UNIVERSITY OF CALIFORNIA,
IRVINE

Safe Learning and Verification of Neural Network Controllers for Autonomous Systems

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Electrical and Computer Engineering

by

Xiaowu Sun

2022

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

I am extremely grateful to my advisor, Prof. Yasser Shoukry, for his support and guidance throughout these years. He offered me a Ph.D. position and showed trust in my abilities when I was new to the field. He was always there for me whenever I needed advice, whether it be technical, professional, personal, or otherwise. I am incredibly lucky to have had Yasser as my advisor, and this means so much to my life.

I would like to extend my sincere appreciation to my committee members, Prof. Mohammad Al Faruque and Prof. Yanning Shen, for reviewing my work and providing insightful comments. I would also like to express my deepest gratitude to Prof. Donald Yeung and Prof. P. S. Krishnaprasad for providing me with research opportunities and advice when I was in Maryland.

I must also thank my colleagues who shared this journey with me in the Resilient Cyber-Physical Systems Lab. A special thanks to Wael Fatnassi, James Ferlez, Haitham Khedr, Momina Sajid, Ulices Santa Cruz Leal, Kohei Tsujio, and Goli Vaisi.

Finally, I want to thank my family for their unconditional love. I would not be the person I am today without their steadfast support and encouragement through the highs and the lows in my life.

VITA

Xiaowu Sun

2013	B.Sc. in Physics, Nanjing University, China
2013 - 2016	Research and Teaching Assistant, Department of Physics and Astronomy, University of Pittsburgh
2016	M.Sc. in Physics, University of Pittsburgh
2018	M.Sc. in Electrical Engineering, University of Maryland, College Park
2018	Dean's Fellowship, University of Maryland, College Park
2021	Finalist in the ACM SIGBED SRC Student Competition at the Cyber-Physical Systems (CPS-IoT) Week 2021
Summer 2022	Software Engineer Intern, Uber Technologies, Inc., San Francisco
2018 - 2022	Research and Teaching Assistant, Department of Electrical Engineering and Computer Science, University of California, Irvine

ABSTRACT OF THE DISSERTATION

Safe Learning and Verification of Neural Network Controllers for Autonomous Systems

By

Xiaowu Sun

Doctor of Philosophy in Electrical and Computer Engineering

University of California, Irvine, 2022

Assistant Professor Yasser Shoukry, Chair

The last decade has witnessed tremendous success in using machine learning (ML) to control physical systems, such as autonomous vehicles, drones, and smart cities. On the one hand, learning-based controller synthesis enjoys the scalability and flexibility benefits offered by purely data-driven architectures. Nevertheless, these end-to-end learning approaches suffer from the lack of safety, reliability, and generalization guarantees. On the other hand, control-theoretic and formal-methods techniques enjoy the guarantees of satisfying high-level specifications. Nevertheless, these algorithms need an explicit model of the dynamic systems and suffer from computational complexity whenever the dynamical models are highly nonlinear and complex. The objective of this dissertation is to develop learning algorithms and verification tools that bridge ideas from symbolic control/reasoning techniques to design ML-controlled autonomous systems with certifiable trust and assurance.

The contributions of this dissertation are multi-fold. (1) We propose a neurosymbolic framework that integrates machine learning and symbolic techniques in training neural network (NN) controllers for robotic systems to satisfy temporal logic specifications. In particular, the trained NN controllers enjoy strong correctness guarantees when applying to unseen tasks, i.e., the exact task (including the environment, specifications, and dynamic constraints of a robot) is unknown during the training of NNs. (2) We introduce the first framework to formally reason about the safety of autonomous systems equipped with a neural network controller that processes LiDAR images to produce control actions. Given a NN-controlled autonomous system that processes the environment with a LiDAR sensor, our framework computes a set of safe initial states such that the autonomous system is guaranteed to be safe when starting from these initial states. (3) We propose a novel approach called NNSynth that uses machine learning techniques to guide the design of abstraction-based controllers. Thanks to the use of ML, NNSynth achieves significant performance improvement compared to traditional controller synthesis while maintaining probabilistic guarantees in the meantime. (4) We consider the problem of automatically designing neural network architectures and exhibit a systematic methodology for choosing NN architectures that are guaranteed to implement a controller that satisfies the given high-level specification. (5) Finally, we present an efficient multi-robot motion planning algorithm for missions captured by temporal logic specifications in the presence of bounded disturbances and denial-of-service (DoS) attacks.

Part I

Safe Learning for Controller Synthesis

Chapter 1

Neurosymbolic Motion and Task Planning for Linear Temporal Logic

This chapter presents a neurosymbolic framework to solve motion planning problems for mobile robots involving temporal goals. The temporal goals are described using temporal logic formulas such as Linear Temporal Logic (LTL) to capture complex tasks. The proposed framework trains Neural Network (NN)-based planners that enjoy strong correctness guarantees when applying to unseen tasks, i.e., the exact task (including workspace, LTL formula, and dynamic constraints of a robot) is unknown during the training of NNs. Our approach to achieving theoretical guarantees and computational efficiency is based on two insights. First, we incorporate a symbolic model into the training of NNs such that the resulting NN-based planner inherits the interpretability and correctness guarantees of the symbolic model. Moreover, the symbolic model serves as a discrete "memory", which is necessary for satisfying temporal logic formulas. Second, we train a library of neural networks offline and combine a subset of the trained NNs into a single NN-based planner at runtime when a task is revealed. In particular, we develop a novel constrained NN training procedure, named formal NN training, to enforce that each neural network in the library represents a "symbol" in the symbolic model. As a result, our neurosymbolic framework enjoys the scalability and flexibility benefits of machine learning and inherits the provable guarantees from control-theoretic and formal-methods techniques. We demonstrate the effectiveness of our framework in both simulations and on an actual robotic vehicle, and show that our framework can generalize to unknown tasks where state-of-the-art meta-reinforcement learning techniques fail.

1.1 Introduction

Developing intelligent machines with a considerable level of cognition dates to the early 1950s. With the current rise of machine learning (ML) techniques, robotic platforms are witnessing a breakthrough in their cognition. Nevertheless, regardless of how many environments they were trained (or programmed) to consider, such intelligent machines will always face new environments which the human designer failed to examine during the training phase. To circumvent the lack of autonomous systems to adapt to new environments, several researchers asked whether we could build autonomous agents that can learn how to learn. In other words, while conventional machine learning focuses on designing agents that can perform one task, the so-called meta-learning aims instead to solve the problem of designing agents that can generalize to different tasks that were not considered during the design or the training of these agents. For example, in the context of meta-Reinforcement Learning (meta-RL), given data collected from a multitude of tasks (e.g., changes in the environments, goals, and robot dynamics), meta-RL aims to combine all such experiences and use them to design agents that can quickly adapt to unseen tasks. While the current successes of meta-RL are undeniable, significant drawbacks of meta-RL in its current form are (i) *the lack of formal guarantees on its ability to generalize to unseen tasks*, (ii) *the lack of formal guarantees with regards to its safety* and (iii) *the lack of interpretability due to the use of black-box deep learning* *techniques.*

In this chapter, we focus on the problem of designing Neural Network (NN)-based task and motion planners that are guaranteed to generalize to unseen tasks, enjoy strong safety guarantees, and are interpretable [139, 143, 142]. We consider agents who need to accomplish temporal goals captured by temporal logic formulas such as Linear Temporal Logic (LTL) [21, 78]. The use of LTL in task and motion planning has been widely studied (e.g., [76, 75, 19, 54, 58, 18, 43, 44, 74, 130, 147, 12]) due to the ability of LTL formulas to capture complex goals such as "eventually visit region A followed by a visit to region B or region C while always avoiding hitting obstacle D." On the one hand, motion and task planning using symbolic techniques enjoy the guarantees of satisfying task specifications in temporal logic. Nevertheless, these algorithms need an explicit model of the dynamic constraints of the robot and suffer from computational complexity whenever such dynamic constraints are highly nonlinear and complex. On the other hand, machine learning approaches are capable of training NN planners without the explicit knowledge of the dynamic constraints and scale favorably to highly nonlinear and complex dynamics. Nevertheless, these data-driven approaches suffer from the lack of safety and generalization guarantees. Therefore, in this work, we aim to design a novel *neurosymbolic* framework for motion and task planning by combining the benefits of symbolic control and machine learning techniques.

At the heart of the proposed framework is using a symbolic model to guide the training of NNs and restricting the behavior of NNs to "symbols" in the symbolic model. Specifically, our framework consists of offline (or training) and online (or runtime) phases. During the offline phase, we assume access to a "nominal" simulator that approximates the dynamic constraints of a robot. We assume no knowledge of the exact task (e.g., workspace, LTL formula, and exact dynamic constraints of a robot). We use this information to train a "library" of NNs through a novel NN training procedure, named formal NN training, which enforces each trained NN to represent a continuous piece-wise affine (CPWA) function from a chosen family of CPWA functions. The exact task becomes available only during the online (or runtime) phase. Given the dynamic constraints of a robot, we compute a finite-state Markov decision process (MDP) as our symbolic model. Thanks to the formal NN training procedure, the symbolic model can be constructed so that each of the trained NNs in the library represents a transition in the MDP (and hence a symbol in this MDP). By analyzing this symbolic model, our framework selects NNs from the library and combines them into a single NN-based planner to perform the task and motion planning.

In summary, the main contributions of this chapter are:

1) We propose a *neurosymbolic* framework that integrates machine learning and symbolic techniques in training NN-based planners for an agent to accomplish *unseen* tasks. Thanks to the use of a symbolic model, the resulting NN-based planners are guaranteed to satisfy the temporal goals described in linear temporal logic formulas, which cannot be satisfied by existing NN training algorithms.

2) We develop a formal training algorithm that restricts the trained NNs to specific local behavior. The training procedure combines classical gradient descent training of NNs with a novel *NN weight projection operator* that modifies the NN weights as little as possible to ensure the trained NN belongs to a chosen family of CPWA functions. We provide theoretical guarantees on the proposed *NN weight projection operator* in terms of correctness and upper bounds on the error between the NN before and after the projection.

3) We provide a theoretical analysis of the overall *neurosymbolic* framework. We show theoretical guarantees that govern the correctness of the resulting NN-based planners when generalizing to *unseen* tasks, including unknown workspaces, unknown temporal logic formulas, and uncertain dynamic constraints.

4) We pursue the high performance of the proposed framework in fast adaptation to

unseen tasks with efficient training. For example, we accelerate the training of NNs by employing ideas from transfer learning and constructing the symbolic model using a data-driven approach. We validate the effectiveness of the proposed framework on an actual robotic vehicle and demonstrate that our framework can generalize to unknown tasks where state-of-the-art meta-RL techniques are known to fail (e.g., when the tasks are chosen from across homotopy classes [24]).

The remainder of the chapter is organized as follows. After the problem formulation in Section 3.2, we present the formal NN training algorithm in Section 1.3. In Section 1.4, we introduce the neurosymbolic framework that uses the formal NN training algorithm to obtain a library of NNs and combines them into a single NN-based planner at runtime. In Section 1.5, we provide theoretical guarantees of the proposed framework. In Section 1.6, we present some key elements for performance improvement while maintaining the same theoretical guarantees. Experimental results are given in Section 4.6.

Related work: The literature on the safe design of ML-based motion and task planners can be classified according to three broad approaches, namely (i) incorporating safety in the training of ML-based planners, (ii) post-training verification of ML models, and (iii) online validation of safety and control intervention. Representative examples of the first approach include reward-shaping [67, 125], Bayesian and robust regression [15, 86, 102], and policy optimization with constraints [2, 155, 162]. Unfortunately, these approaches do not provide provable guarantees about the safety of the trained ML-based planners.

To provide strong safety and reliability guarantees, several works in the literature focus on applying formal verification techniques (e.g., model checking) to verify pre-trained ML models against formal safety properties. Representative examples of this approach include the use of SMT-like solvers [38, 87, 140, 70, 47, 123] and hybrid-system verification [45, 65, 167]. However, these techniques only assess a given ML-based planner's safety rather than design or train a safe agent.

Due to the lack of safety guarantees on the resulting ML-based planners, researchers proposed several techniques to *restrict* the output of the ML models to a set of safe control actions. Such a set of safe actions can be obtained through Hamilton-Jacobi analysis [52, 63] and barrier certificates [1, 28, 30, 115, 150, 160, 171]. Unfortunately, methods of this type suffer from being computationally expensive, specific to certain controller structures, or requiring assumptions on the system model. Other techniques in this domain include synthesizing a safety layer (shield) based on model predictive control with the assumption of safe terminal sets [9, 157, 158], logically-constrained reinforcement learning [60, 8, 4], and Lyapunov methods [16, 33, 34] that focus on providing stability guarantees rather than safety or general temporal logic guarantees.

The idea of learning neurosymbolic models is studied in works [5, 156, 10] that use NNs to guide the synthesis of control policies represented as short programs. The algorithms in [5, 156, 10] train a NN controller, project it to the space of program languages, analyze the short programs, and lift the programs back to the space of NNs for further training. These works focus on tasks given during the training of NNs, and the final controller is a short program. Another line of related work is reported in [161, 25], which study the problem of extracting a finite-state controller from a recurrent neural network. Unlike the above works, we consider temporal logic specifications and unseen tasks, and our final planner is NNs in tandem with a finite-state MDP.

1.2 Problem Formulation

Let $\mathbb{R}$, $\mathbb{R}^+$, $\mathbb{N}$ be the set of real numbers, positive real numbers, and natural numbers, respectively. For a non-empty set S, let 2^S be the power set of S, $\mathbf{1}_S$ be the indicator function of S, and $\mathrm{Int}(S)$ be the interior of S. Furthermore, we use S^n to denote the set of all finite sequences of length $n \in \mathbb{N}$ of elements in S. The product of two sets is defined as $S_1 \times S_2 := \{(s_1, s_2) | s_1 \in S_1, s_2 \in S_2\}$. Let $\|x\|$ be the Euclidean norm of a vector $x \in \mathbb{R}^n$, $\|A\|$ be the induced 2-norm of a matrix $A \in \mathbb{R}^{m \times n}$, and $\|A\|_{\max} = \max\limits_{i,j} |A_{ij}|$ be the max norm of a matrix A. Any Borel space X is assumed to be endowed with a Borel σ-algebra denoted by $\mathcal{B}(X)$.

1.2.1 Assumptions and Information Structure

We consider a meta-RL setting that aims to train neural networks for controlling a robot to achieve tasks that were unseen during training. To be specific, we denote a task by a tuple $\mathcal{T} = (t, \varphi, \mathcal{W}, X_0)$, where t captures the dynamic constraints of a robot (see Section 1.2.2), φ is a Linear Temporal Logic (LTL) formula that defines the mission for a robot to accomplish (see Section 2.2.2), $\mathcal{W}$ is a workspace (or an environment) in which a robot operates, and X_0 contains the initials states of a robot. Furthermore, we use J to denote a cost functional of controllers, and the cost of using a neural network $\mathcal{NN}$ is given by $J(\mathcal{NN})$ (see Section 1.2.5).

During training, we assume the availability of the cost functional J and an approximation of the dynamical model t (see Section 1.2.2 for details). The mission specification φ, the workspace $\mathcal{W}$, and the set of initial states X_0 are unknown during training and only become available at runtime. Despite the limited knowledge of tasks during training, we aim to design provably correct NNs for unseen tasks $\mathcal{T}$ while minimizing some given cost J.

1.2.2 Dynamical Model

We consider robotic systems that can be modeled as stochastic, discrete-time, nonlinear dynamical systems with a transition probability of the form:

$$\Pr(x' \in A|x, u) = \int_A t(dx'|x, u), \tag{1.1}$$

where states of a robot $x \in X$ and control actions $u \in U$ are from continuous state and action spaces $X \subset \mathbb{R}^n$ and $U \subset \mathbb{R}^m$, respectively. In (2.2), we use $t : \mathcal{B}(X) \times X \times U \to [0, 1]$ to denote a stochastic kernel that assigns to any state $x \in X$ and action $u \in U$ a probability measure $t(\cdot|x, u)$. Then, $\Pr(x' \in A|x, u)$ is the probability of reaching a subset $A \in \mathcal{B}(X)$ in one time step from state $x \in X$ under action $u \in U$. We assume that t consists of a priori known nominal model f and an unknown model-error g capturing unmodeled dynamics. As a well-studied technique to learn unknown functions from data, we assume the model-error g can be learned by a Gaussian Process (GP) regression model $\mathcal{GP}(\mu_g, \sigma_g^2)$, where μ_g and σ_g^2 are the posterior mean and variance functions, respectively [113]. Hence, we can re-write (2.2) as:

$$\Pr(x' \in A|x, u) = \int_A \mathcal{N}(dx'|f(x, u) + \mu_g(x, u), \sigma_g^2(x, u)), \tag{1.2}$$

which is an integral of the normal distribution $\mathcal{N}(f(x, u) + \mu_g(x, u), \sigma_g^2(x, u))$ and hence can be easily computed.

We assume the nominal model f is given during the NN training phase, while the model-error g is evaluated at runtime, and hence the exact stochastic kernel t only becomes known at runtime. This allows us to apply the trained NN to various robotic systems with different dynamics captured by the model error g.

Remark: We note that our algorithm does not require the knowledge of the function f in a closed-form/symbolic representation. Access to a simulator would suffice.

1.2.3 Temporal Logic Specification and Workspace

A well-known weakness of RL and meta-RL algorithms is the difficulty in designing reward functions that capture the exact intent of designers [51, 60, 8]. Agent behavior that scores high according to a user-defined reward function may not be aligned with the user's intention, which is often referred to as "specification gaming" [116]. To that end, we adopt the representation of an agent's mission in temporal logic specifications, which have been extensively demonstrated the capability to capture complex behaviors of robotic systems.

In particular, we consider mission specifications defined in either bounded linear temporal logic (BLTL) [21] or syntactically co-safe linear temporal logic (scLTL) [78]. Let AP be a finite set of atomic propositions that describe a robotic system's states with respect to a workspace $\mathcal{W}$. For example, these atomic propositions can describe the location of a robot with respect to the obstacles to avoid and the goal location to achieve. Given AP, any BLTL formula can be generated according to the following grammar:

$$\varphi := \sigma \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \, \mathcal{U}_{[k_1,k_2]} \, \varphi_2$$

where $\sigma \in AP$ and time steps $k_1 < k_2$. Given the above grammar, we can define $\varphi_1 \wedge \varphi_2 = \neg(\neg\varphi_1 \vee \neg\varphi_2)$, $false = \varphi \wedge \neg\varphi$, and $true = \neg false$. Furthermore, the bounded-time *eventually* operator can be derived as $\Diamond_{[k_1,k_2]}\varphi = true \, \mathcal{U}_{[k_1,k_2]} \, \varphi$ and the bounded-time *always* operator is given by $\Box_{[k_1,k_2]}\varphi = \neg\Diamond_{[k_1,k_2]}\neg\varphi$.

Given a set of atomic propositions AP, the corresponding alphabet is defined as $\mathbb{A} := 2^{AP}$, and a finite (infinite) word ω is a finite (infinite) sequence of letters from the alphabet $\mathbb{A}$, i.e., $\omega = \omega^{(0)}\omega^{(1)} \ldots \omega^{(H)} \in \mathbb{A}^{H+1}$. The satisfaction of a word ω to a specification φ can be determined based on the semantics of BLTL [21]. Given a robotic system and an alphabet $\mathbb{A}$, let $L : X \to \mathbb{A}$ be a labeling function that assigns to each state $x \in X$ the subset of atomic propositions $L(x) \in \mathbb{A}$ that evaluate *true* at x. Then, a robotic system's trajectory ξ satisfies a specification φ, denoted by $\xi \models \varphi$, if the corresponding word satisfies φ, i.e., $L(\xi) \models \varphi$, where $\xi = x^{(0)}x^{(1)} \dots x^{(H)} \in X^{H+1}$ and $L(\xi) = L(x^{(0)})L(x^{(1)}) \dots L(x^{(H)}) \in \mathbb{A}^{H+1}$. Similarly, we can consider scLTL specifications interpreted over infinite words based on the fact that any infinite word that satisfies a scLTL formula φ contains a finite "good" prefix such that all infinite words that contain the prefix satisfy φ [78].

Example 1 *(Reach-avoid Specification)*: Consider a robot that navigates a workspace $\mathcal{W} = \{X_{\text{goal}}, O_1, \dots, O_c\}$, where $X_{\text{goal}} \subset X$ is a set of goal states that the robot would like to reach and $O_1, \dots, O_c \subset X$ are obstacles that the robot needs to avoid. The set of atomic propositions is given by $AP = \{x \in X_{\text{goal}}, x \in O_1, \dots, x \in O_c\}$, where x is the state of the robot. Then, a reach-avoid specification can be expressed as $\varphi = \varphi_{\text{liveness}} \wedge \varphi_{\text{safety}}$, where $\varphi_{\text{liveness}} = \Diamond_{[0,H]}(x \in X_{\text{goal}})$ requires the robot to reach the goal X_{goal} in H time steps and $\varphi_{\text{safety}} = \Box_{[0,H]} \bigwedge_{i=1,\dots,c} \neg(x \in O_i)$ specifies to avoid all the obstacles during the time horizon H. Let $\xi = x^{(0)}x^{(1)} \dots x^{(H)}$ be a trajectory of the robot, then the reach-avoid specification φ is interpreted as:

$$\xi \models \varphi_{\text{liveness}} \Longleftrightarrow \exists k \in \{0, \dots H\}, x^{(k)} \in X_{\text{goal}},$$
$$\xi \models \varphi_{\text{safety}} \Longleftrightarrow \forall k \in \{0, \dots H\}, \forall i \in \{1, \dots, c\}, x^{(k)} \notin O_i.$$

1.2.4 Neural Network

To account for the stochastic behavior of a robot, we aim to design a state-feedback neural network $\mathcal{NN} : X \to U$ that can achieve temporal motion and task specifications φ. An F-layer Rectified Linear Unit (ReLU) NN is specified by composing F layer functions (or just layers). A layer l with $\mathfrak{i}_l$ inputs and $\mathfrak{o}_l$ outputs is specified by a weight matrix $W^{(l)} \in \mathbb{R}^{\mathfrak{o}_l \times \mathfrak{i}_l}$ and a bias vector $b^{(l)} \in \mathbb{R}^{\mathfrak{o}_l}$ as follows:

$$L^{\theta^{(l)}} : z \mapsto \max\{W^{(l)} z + b^{(l)}, 0\}, \tag{1.3}$$

where the max function is taken element-wise, and $\theta^{(l)} \triangleq (W^{(l)}, b^{(l)})$ for brevity. Thus, an F-layer ReLU NN is specified by F layer functions $\{L^{\theta^{(l)}} : l = 1, \ldots, F\}$ whose input and output dimensions are composable: that is, they satisfy $\mathfrak{i}_l = \mathfrak{o}_{l-1}$, $l = 2, \ldots, F$. Specifically:

$$\mathcal{NN}^{\theta}(x) = (L^{\theta^{(F)}} \circ L^{\theta^{(F-1)}} \circ \cdots \circ L^{\theta^{(1)}})(x), \tag{1.4}$$

where we index a ReLU NN function by a list of parameters $\theta \triangleq (\theta^{(1)}, \ldots, \theta^{(F)})$. As a common practice, we allow the output layer $L^{\theta^{(F)}}$ to omit the max function. For simplicity of notation, we drop the superscript θ in $\mathcal{NN}^{\theta}$ whenever the dependence on θ is obvious.

1.2.5 Main Problem

We consider training a finite set (or a library) of ReLU NNs (during the offline phase) and designing a selection algorithm (during the online phase) that can select the correct NNs once the exact task $\mathcal{T} = (t, \varphi, \mathcal{W}, X_0)$ is revealed at runtime. Before formalizing the problem under consideration, we introduce the following notion of neural network composition.

Definition 1.1. *Given a set (or a library) of neural networks $\mathfrak{NN} = \{\mathcal{NN}_1, \mathcal{NN}_2, \ldots, \mathcal{NN}_d\}$ along with an activation map $\Gamma : X \to \{1, \ldots, d\}$, the composed neural network $\mathcal{NN}_{[\mathfrak{NN}, \Gamma]}$ is defined as: $\mathcal{NN}_{[\mathfrak{NN}, \Gamma]}(x) = \mathcal{NN}_{\Gamma(x)}(x)$.*

In other words, the activation map Γ selects the NN that needs to be activated at each state $x \in X$. In addition to achieving the motion and task specifications, the neural network needs to minimize a given cost functional J. The cost functional J is defined as:

$$J(\mathcal{NN}_{[\mathfrak{NN},\Gamma]}) = \int_X c(x, \mathcal{NN}_{[\mathfrak{NN},\Gamma]}(x)) d\mu^{\mathcal{NN}}(x), \tag{1.5}$$

where $c : X \times U \to \mathbb{R}$ is a state-action cost function and $\mu^{\mathcal{NN}}$ is the distribution of states induced by the nominal dynamics f in (4.1) under the control of $\mathcal{NN}_{[\mathfrak{NN},\Gamma]}$. As an example, the cost functional can be a controller's energy $J(\mathcal{NN}_{[\mathfrak{NN},\Gamma]}) = \int_X \|\mathcal{NN}_{[\mathfrak{NN},\Gamma]}(x)\|^2 d\mu^{\mathcal{NN}}(x)$.

Let $\xi^x_{\mathcal{NN}_{[\mathfrak{NN},\Gamma]}}$ be a closed-loop trajectory of a robot that starts from the state $x \in X_0$ and evolves under the composed neural network $\mathcal{NN}_{[\mathfrak{NN},\Gamma]}$. We define the problem of interest as follows:

Problem 1.1. *Given a cost functional J, train a library of ReLU neural networks $\mathfrak{NN}$, and compute an activation map Γ at runtime when a task $\mathcal{T} = (t, \varphi, \mathcal{W}, X_0)$ is revealed, such that the composed neural network minimizes the cost $J(\mathcal{NN}_{[\mathfrak{NN},\Gamma]})$ and satisfies the specification φ with probability at least p, i.e.,* $\Pr\left(\xi^x_{\mathcal{NN}_{[\mathfrak{NN},\Gamma]}} \models \varphi\right) \geq p$ *for any $x \in X_0$.*

1.2.6 Overview of the Neurosymbolic Framework

Our approach to designing the NN-based planner $\mathcal{NN}_{[\mathfrak{NN},\Gamma]}$ can be split into two stages: offline training and runtime selection. During the offline training phase, our algorithm obtains a library of networks $\mathfrak{NN}$. At runtime, and to fulfill *unseen* tasks using a *finite* set of neural networks $\mathfrak{NN}$, our neurosymbolic framework bridges ideas from symbolic LTL-based planning and machine learning. Similar to symbolic LTL-based planning, our framework uses a hierarchical approach that consists of a "high-level" discrete planner and a "low-level" continuous controller [130, 43, 18]. The "high-level" discrete planner focuses on ensuring the satisfaction of the LTL specification. At the same time, the "low-level" controllers compute control actions that steer the robot to satisfy the "high-level" plan. Unlike symbolic LTL-based planners, our framework uses neural networks as low-level controllers, thanks to their ability to handle complex nonlinear dynamic constraints. In particular, the "high-level" planner chooses the activation map Γ to activate particular neural networks.

Nevertheless, to ensure the correctness of the proposed framework, it is essential to ensure that each neural network in $\mathfrak{NN}$ satisfies some "formal" property. This "formal" property allows the high-level planner to abstract the capabilities of each of the neural networks in $\mathfrak{NN}$ and hence choose the correct activation map Γ. To that end, in Section 1.3, we formulate the sub-problem of "formal NN training" that guarantees the trained NNs satisfy certain formal properties, and solve it efficiently by introducing a NN weight projection operator. The solution to the formal training is used in Section 1.4.1 to obtain the library of networks $\mathfrak{NN}$ offline. The associated formal property of each NN is used in Section 1.4.2 to design the activation map Γ.

1.3 Formal Training of NNs

In this section, we study the sub-problem of training NNs that are guaranteed to obey certain behaviors. In addition to the classical gradient-descent update of NN weights, we propose a novel "projection" operator that ensures the resulting NN obeys the selected behavior. We provide a theoretical analysis of the proposed projection operator in terms of correctness and computational complexity.

1.3.1 Formulation of Formal Training

We start by recalling that every ReLU NN represents a Continuous Piece-Wise Affine (CPWA) function [95]. Let $\Psi_{\text{CPWA}} : X \to \mathbb{R}^m$ denote a CPWA function of the form:

$$\Psi_{\text{CPWA}}(x) = K_i' x + b_i' \quad \text{if } x \in \mathcal{R}_i,\ i = 1, \ldots, L, \tag{1.6}$$

where the collection of polytopic subsets $\{\mathcal{R}_1, \ldots, \mathcal{R}_L\}$ is a partition of the set $X \subset \mathbb{R}^n$ such that $\bigcup_{i=1}^{L} \mathcal{R}_i = X$ and $\text{Int}(\mathcal{R}_i) \cap \text{Int}(\mathcal{R}_j) = \emptyset$ if $i \neq j$. We call each polytopic subset $\mathcal{R}_i \subset X$ a linear region, and denote by $\mathbb{L}_{\Psi_{\text{CPWA}}}$ the set of linear regions associated to Ψ_{CPWA}, i.e., $\mathbb{L}_{\Psi_{\text{CPWA}}} = \{\mathcal{R}_1, \ldots, \mathcal{R}_L\}$. In this chapter, we confine our attention to CPWA controllers (and hence neural network controllers) that are selected from a bounded polytopic set $\mathcal{P}^K \times \mathcal{P}^b \subset \mathbb{R}^{m \times n} \times \mathbb{R}^m$, i.e., we assume that $K_i' \in \mathcal{P}^K$ and $b_i' \in \mathcal{P}^b$. For simplicity of notation, we use $\mathcal{P}^{K \times b} \subset \mathbb{R}^{m \times (n+1)}$ to denote the polytopic set $\mathcal{P}^K \times \mathcal{P}^b$, and use $K_i(x)$ with a single parameter $K_i \in \mathcal{P}^{K \times b}$ to denote $K_i' x_i + b_i'$ with the pair $(K_i', b_i') = K_i$.

Let $\mathcal{P} \subseteq \mathcal{P}^{K \times b}$ be a bounded polytopic subset of the parameters K_i, then with some abuse of notation, we use the same notation $\mathcal{P}$ to denote the subset of CPWA functions whose parameters K_i are chosen from $\mathcal{P}$. In other words, a CPWA function $\Psi_{\text{CPWA}} \in \mathcal{P}$ if and only if $K_i \in \mathcal{P}$ at all linear regions $\mathcal{R}_i \in \mathbb{L}_{\Psi_{\text{CPWA}}}$, where the CPWA function Ψ_{CPWA} is in the form of (1.6).

Using this notation, we define the formal training problem that ensures the trained NNs belong to subsets of CPWA functions $\mathcal{P} \subseteq \mathcal{P}^{K \times b}$ as follows:

Problem 1.2. *Given a bounded polytopic subset $q \subseteq X$, a bounded subset of CPWA functions $\mathcal{P} \subseteq \mathcal{P}^{K \times b}$, and a cost functional J, find NN weights θ^* such that:*

$$\theta^* = \underset{\theta}{\text{argmin}}\ J(\mathcal{NN}^{\theta}) \quad \text{s.t.} \quad \mathcal{NN}^{\theta}|_q \in \mathcal{P}. \tag{1.7}$$

In Problem 1.2, we use $\mathcal{NN}^{\theta}|_q$ to denote the restriction of $\mathcal{NN}^{\theta}$ to the subset q, i.e., $\mathcal{NN}^{\theta}|_q : q \to \mathbb{R}^m$ is given by $\mathcal{NN}^{\theta}|_q(x) = \mathcal{NN}^{\theta}(x)$ for $x \in q$. Consider the CPWA function $\mathcal{NN}^{\theta}$

is in the form of (1.6), then the constraint $\mathcal{NN}^\theta|_q \in \mathcal{P}$ requires that $K_i \in \mathcal{P}$ whenever the corresponding linear region $\mathcal{R}_i$ intersects the subset q, i.e.:

$$\mathcal{NN}^\theta|_q \in \mathcal{P} \iff K_i \in \mathcal{P},\ \forall \mathcal{R}_i \in \{\mathcal{R} \in \mathbb{L}_{\mathcal{NN}^\theta} | \mathcal{R} \cap q \neq \emptyset\}. \tag{1.8}$$

1.3.2 NN Weight Projection

To solve Problem 1.2, we introduce a NN weight projection operator that can be incorporated into the training of neural networks. Algorithm 1 outlines our procedure for solving Problem 1.2. As a projected-gradient algorithm, Algorithm 1 alternates the gradient descent based training (line 3 in Algorithm 1) and the NN weight projection (line 4-5 in Algorithm 1) up to a pre-specified maximum iteration max_iter. Given a subset of CPWA functions $\mathcal{P} \subseteq \mathcal{P}^{K\times b}$, we denote by $\Pi_{\mathcal{P}}$ the NN weight projection operator that enforces a network $\mathcal{NN}^\theta$ to satisfy $\mathcal{NN}^\theta|_q \in \mathcal{P}$, i.e., the constraints (1.8). In the following, we formulate this NN weight projection operator $\Pi_{\mathcal{P}}$ as an optimization problem.

Algorithm 1 FORAMAL-TRAIN $(q, \mathcal{P}, J)$

```
1: Initialize neural network $\mathcal{NN}^\theta$, $i = 1$
2: while $i \leq$ max_iter do
3:     $\mathcal{NN}^\theta$ = gradient-descent($\mathcal{NN}^\theta, \mathcal{P}, J$)
4:     $\widehat{W}^{(F)}, \widehat{b}^{(F)} = \Pi_{\mathcal{P}}(\mathcal{NN}^\theta)$
5:     Set the output layer weights of $\mathcal{NN}^\theta$ be $\widehat{W}^{(F)}, \widehat{b}^{(F)}$
6:     $i = i + 1$
7: end while
8: Return $\mathcal{NN}^\theta$
```

Consider a neural network $\mathcal{NN}^\theta$ with F layers, including $F-1$ hidden layers and an output layer. Let $W^{(F)}$ and $b^{(F)}$ be the weight matrix and the bias vector of the output layer, respectively, i.e.:

$$\theta = \left(\theta^{(1)}, \dots, \theta^{(F-1)},\ (W^{(F)}, b^{(F)})\right) \tag{1.9}$$

Then, the NN weight projection $\Pi_{\mathcal{P}}$ updates the output layer weights $W^{(F)}$, $b^{(F)}$ to $\widehat{W}^{(F)}$, $\widehat{b}^{(F)}$ (line 4-5 in Algorithm 1). As a result, the projected NN weights $\widehat{\theta}$ are given by:

$$\widehat{\theta} = \left(\theta^{(1)}, \ldots, \theta^{(F-1)},\ (\widehat{W}^{(F)}, \widehat{b}^{(F)})\right). \tag{1.10}$$

We formulate the NN weight projection operator $\Pi_{\mathcal{P}}$ as the following optimization problem:

$$\operatorname*{argmin}_{\widehat{W}^{(F)}, \widehat{b}^{(F)}} \max_{x \in q} \|\mathcal{NN}^{\widehat{\theta}}(x) - \mathcal{NN}^{\theta}(x)\|_1 \tag{1.11}$$

$$\text{s.t. } \widehat{K}_i \in \mathcal{P},\ \forall \mathcal{R}_i \in \{\mathcal{R} \in \mathbb{L}_{\mathcal{NN}^{\theta}} \mid \mathcal{R} \cap q \neq \emptyset\}. \tag{1.12}$$

In the constraints (1.12), we use $\widehat{K}_i$ to denote the affine function parameters of the CPWA function $\mathcal{NN}^{\widehat{\theta}}$.

The optimization problem (1.11)-(1.12) tries to minimize the change of the NN's outputs due to the weight projection, where the change is measured by the largest 1-norm difference between the outputs given by $\mathcal{NN}^{\widehat{\theta}}$ and $\mathcal{NN}^{\theta}$ across the subset $q \subseteq X$, i.e., $\max_{x \in q} \|\mathcal{NN}^{\widehat{\theta}}(x) - \mathcal{NN}^{\theta}(x)\|_1$. In the following two subsections, we first upper bound the objective function (1.11) in terms of the change of the NN's weights, and then show that the optimization problem (1.11)-(1.12) can be solved efficiently.

1.3.3 Bounding the Change of Control Actions

First, we note that it is common to omit the ReLU activation functions from the NN's output layer. Since the proposed projection operator only modifies the output layer weights, it is straightforward to show that the NN weight projection operator does not affect the set of linear regions, i.e., $\mathbb{L}_{\mathcal{NN}^{\widehat{\theta}}} = \mathbb{L}_{\mathcal{NN}^{\theta}}$, but only updates the affine functions defined over these regions. The following proposition shows the relation between the change in the NN's outputs and the change made in the output layer weights.

Proposition 1.3. *Consider two F-layer neural networks $\mathcal{NN}^{\theta}$ and $\mathcal{NN}^{\widehat{\theta}}$ where θ and $\widehat{\theta}$ are as defined in* (1.9)-(1.10). *Then, the largest difference in the NNs' outputs across a subset $q \subseteq X$ is upper bounded as follows:*

$$\begin{aligned} &\max_{x \in q} \|\mathcal{NN}^{\widehat{\theta}}(x) - \mathcal{NN}^{\theta}(x)\|_1 \\ &\leq \max_{x \in \mathrm{Vert}\left(\mathbb{L}_{\mathcal{NN}^{\theta} \cap q}\right)} \sum_{i=1}^{m} \sum_{j=1}^{\mathfrak{o}_{F-1}} |\Delta W_{ij}^{(F)}| h_j(x) + \sum_{i=1}^{m} |\Delta b_i^{(F)}|. \end{aligned} \tag{1.13}$$

In Proposition 1.3, m is the dimension of the NN's output, $\Delta W_{ij}^{(F)}$ and $\Delta b_i^{(F)}$ are the (i, j)-th and the i-th entry of $\Delta W^{(F)} = \widehat{W}^{(F)} - W^{(F)}$ and $\Delta b^{(F)} = \widehat{b}^{(F)} - b^{(F)}$, respectively. With the notation of layer functions (1.3), we use a single function $h : \mathbb{R}^n \to \mathbb{R}^{\mathfrak{o}_{F-1}}$ to represent all the hidden layers, i.e., $h(x) = (L_{\theta^{(F-1)}} \circ L_{\theta^{(F-2)}} \circ \cdots \circ L_{\theta^{(1)}})(x)$, where $\mathfrak{o}_{F-1}$ is the number of neurons in the $(F-1)$-layer (the last hidden layer). Furthermore, we use $\mathbb{L}_{\mathcal{NN}^{\theta} \cap q}$ to denote the intersected regions between the linear regions in $\mathbb{L}_{\mathcal{NN}^{\theta}}$ and the subset $q \subseteq X$, i.e., $\mathbb{L}_{\mathcal{NN}^{\theta} \cap q} = \{\mathcal{R} \cap q | \mathcal{R} \in \mathbb{L}_{\mathcal{NN}^{\theta}}, \mathcal{R} \cap q \neq \emptyset\}$. Let $\mathrm{Vert}(\mathcal{R})$ be the set of vertices of a region $\mathcal{R}$, then $\mathrm{Vert}(\mathbb{L}_{\mathcal{NN}^{\theta} \cap q}) = \bigcup_{\mathcal{R} \in \mathbb{L}_{\mathcal{NN}^{\theta} \cap q}} \mathrm{Vert}(\mathcal{R})$ is the set of vertices of all regions in $\mathbb{L}_{\mathcal{NN}^{\theta} \cap q}$.

Proof. Let $h : \mathbb{R}^n \to \mathbb{R}^{\mathfrak{o}_{F-1}}$ represent all the hidden layers, then the neural networks before and after the change of the output layer weights are given by $\mathcal{NN}^{\theta} : x \mapsto W^{(F)} h(x) + b^{(F)}$ and $\mathcal{NN}^{\widehat{\theta}} : x \mapsto \widehat{W}^{(F)} h(x) + \widehat{b}^{(F)}$, respectively. The change in the NN's outputs is bounded as follows:

$$\begin{aligned}
&\max_{x\in q} \|\mathcal{NN}^{\widehat{\theta}}(x) - \mathcal{NN}^{\theta}(x)\|_1 \qquad (1.14)\\
&= \max_{x\in q} \sum_{i=1}^{m} \Big| \sum_{j=1}^{o_{F-1}} \Delta W_{ij}^{(F)} h_j(x) + \Delta b_i^{(F)} \Big| \qquad (1.15)\\
&\leq \max_{x\in q} \sum_{i=1}^{m} \sum_{j=1}^{o_{F-1}} |\Delta W_{ij}^{(F)}| h_j(x) + \sum_{i=1}^{m} |\Delta b_i^{(F)}| \qquad (1.16)\\
&= \max_{x\in \mathrm{Vert}(\mathbb{L}_{\mathcal{NN}^{\theta}\cap q})} \sum_{i=1}^{m} \sum_{j=1}^{o_{F-1}} |\Delta W_{ij}^{(F)}| h_j(x) + \sum_{i=1}^{m} |\Delta b_i^{(F)}| \qquad (1.17)
\end{aligned}$$

where (1.15) directly follows the form of $\mathcal{NN}^{\theta}$ and $\mathcal{NN}^{\widehat{\theta}}$, (1.16) swaps the order of taking the absolute value and the summation, and uses the fact that the hidden layers satisfy $h(x) \geq 0$ due to the ReLU activation function. When x is restricted to each linear region of $\mathcal{NN}^{\theta}$, the hidden layer function h is affine, and hence (1.16) is a linear program whose optimal solution is attained at extreme points. Therefore, in (1.17), the maximum can be taken over a *finite* set of states that are vertices of the linear regions in $\mathbb{L}_{\mathcal{NN}^{\theta}\cap q}$. $\square$

1.3.4 Efficient Computation of the NN Projection Operator

Now, we focus on how to compute the NN weight projection operator $\Pi_{\mathcal{P}}$ efficiently. In particular, Proposition 1.3 proposes a direct way to solve the intended projection operator. In order to minimize the change of the NN's outputs (1.11) due to the weight projection, we minimize its upper bound given by (1.13). Accordingly, we compute the NN weight projection operator $\Pi_{\mathcal{P}}$ by solving following optimization problem:

$$\operatorname*{argmin}_{\widehat{W}^{(F)},\widehat{b}^{(F)}} \max_{x\in \mathrm{Vert}\left(\mathbb{L}_{\mathcal{NN}^{\theta}\cap q}\right)} \sum_{i=1}^{m}\sum_{j=1}^{o_{F-1}} \left|\Delta W_{ij}^{(F)}\right| h_j(x) + \sum_{i=1}^{m} \left|\Delta b_i^{(F)}\right| \qquad (1.18)$$

$$\text{s.t. } \widehat{K}_i \in \mathcal{P},\ \forall \mathcal{R}_i \in \{\mathcal{R} \in \mathbb{L}_{\mathcal{NN}^{\theta}} \mid \mathcal{R}\cap q \neq \emptyset\}. \qquad (1.19)$$

The next result establishes the computational complexity of solving the optimization problem above.

Proposition 1.4. *The optimization problem* (1.18)-(1.19) *is a linear program.*

While Proposition 1.4 ensures that solving the optimization problem can be done efficiently, we note that identifying the set of linear regions $\mathbb{L}_{\mathcal{NN}^\theta}$ of a ReLU neural network $\mathcal{NN}^\theta$ needs to enumerate the hyperplanes represented by $\mathcal{NN}^\theta$. For shallow NNs and other special NN architectures, this can be done in polynomial time (e.g., [49] uses a poset for the enumeration). For general NNs, identifying linear regions may not be polynomial time, but there exist efficient tools such as `NNENUM` [6] that uses star sets to enumerate all the linear regions. Moreover, as we will show in the following sections, each NN in the library $\mathfrak{NN}$ can contain a limited number of weights (and hence a limited number of linear regions), but their combination leads to NNs with a large number of linear regions and hence capable of implementing complex functions.

Proof. We write the optimization problem (1.18)-(1.19) in its equivalent epigraph form:

$$\min_{\widehat{W}^{(F)},\widehat{b}^{(F)},t,s_{ij},v_i} \quad t \quad \text{such that}$$

$$\sum_{i=1}^{m}\sum_{j=1}^{\mathfrak{o}_{F-1}} s_{ij}h_j(x) + \sum_{i=1}^{m} v_i \leq t,\ \forall x \in \operatorname{Vert}(\mathbb{L}_{\mathcal{NN}^\theta \cap q}) \tag{1.20}$$

$$|\widehat{W}_{ij}^{(F)} - W_{ij}^{(F)}| \leq s_{ij},\ i = 1,\ldots,m,\ j = 1,\ldots,\mathfrak{o}_{F-1} \tag{1.21}$$

$$|\widehat{b}_i^{(F)} - b_i^{(F)}| \leq v_i,\ i = 1,\ldots,m \tag{1.22}$$

$$\widehat{K}_i \in \mathcal{P},\ \forall \mathcal{R}_i \in \{\mathcal{R} \in \mathbb{L}_{\mathcal{NN}^\theta} \mid \mathcal{R} \cap q \neq \emptyset\}. \tag{1.23}$$

The inequalities in (1.20) are affine since the hidden layer function h is known and does not depend on the optimization variables. The number of inequalities in (1.20) is finite since the set of vertices $\operatorname{Vert}(\mathbb{L}_{\mathcal{NN}^\theta \cap q})$ is finite. To see the constraints (1.23) are affine, consider the neural network $\mathcal{NN}^{\widehat{\theta}} : x \mapsto \widehat{W}^{(F)}h(x) + \widehat{b}^{(F)}$ with the output layer weights $\widehat{W}^{(F)}$, $\widehat{b}^{(F)}$ and the hidden layer function h. The CPWA function $\mathcal{NN}^{\widehat{\theta}}$ can also be written in the form of (1.6), i.e., $\mathcal{NN}^{\widehat{\theta}} : x \mapsto \widehat{K}_i(x)$ at each linear region $\mathcal{R}_i \in \mathbb{L}_{\mathcal{NN}^\theta}$, where we use the notation $\widehat{K}_i(x)$ to denote $\widehat{K}_i'x + \widehat{b}_i'$. Since the hidden-layer function h restricted to each linear region $\mathcal{R}_i \in \mathbb{L}_{\mathcal{NN}^\theta}$ is a known affine function of x, the parameters $\widehat{K}_i$ affinely depend on $\widehat{W}^{(F)}$ and $\widehat{b}^{(F)}$. Therefore, the constraints $\widehat{K}_i \in \mathcal{P}$ are affine constraints of $\widehat{W}^{(F)}$ and $\widehat{b}^{(F)}$. $\square$

We conclude this section with the following result whose proof follows directly from Proposition 1.4 and the equivalence in (1.8).

Theorem 1.5. *Given a bounded polytopic subset $q \subseteq X$ and a bounded subset of CPWA functions $\mathcal{P} \subseteq \mathcal{P}^{K\times b}$. Consider a neural network $\mathcal{NN}^\theta$ whose output layer weights are given by the NN weight projection operator $\Pi_{\mathcal{P}}$ (i.e., the solution to* (1.18)-(1.19)*). Then, the network $\mathcal{NN}^\theta$ satisfies the constraint in* (1.7)*, i.e., $\mathcal{NN}^\theta|_q \in \mathcal{P}$. Furthermore, the optimization problem* (1.18)-(1.19) *is a linear program.*

1.4 Neurosymbolic Learning Framework

As discussed in Section 1.2.6, our approach to designing the NN-based planner $\mathcal{NN}_{[\mathfrak{NN},\Gamma]}$ and solving Problem 2.1 is split into two stages: offline training and runtime selection. During the offline training phase, our algorithm obtains a library of networks $\mathfrak{NN}$, where each NN is trained using the formal training Algorithm 1. At runtime, when the exact task $\mathcal{T} = (t, \varphi, \mathcal{W}, X_0)$ is observed, we use dynamic programming (DP) to compute an activation map Γ, which selects a subset of the trained NNs and combines them into a single planner. We provide details on these two stages in the following two subsections separately.

1.4.1 Offline Training of a Library of NNs

Similar to standard LTL-based motion planners [43, 44, 74, 130, 147, 12], we partition the continuous state space $X \subset \mathbb{R}^n$ into a finite set of abstract states $\mathbb{X} = \{q_1, \ldots, q_N\}$, where each abstract state $q_i \in \mathbb{X}$ is an infinity-norm ball in $\mathbb{R}^n$ with a pre-specified diameter $\eta_q \in \mathbb{R}^+$ (see Section 1.6 for the choice of η_q). The partitioning satisfies $X = \bigcup_{q \in \mathbb{X}} q$ and $\text{Int}(q_i) \cap \text{Int}(q_j) = \emptyset$ if $i \neq j$. Let $\text{abs} : X \to \mathbb{X}$ map a state $x \in X$ to the abstract state $\text{abs}(x) \in \mathbb{X}$ that contains x, i.e., $x \in \text{abs}(x)$, and $\text{ct}_{\mathbb{X}} : \mathbb{X} \to X$ map an abstract state $q \in \mathbb{X}$ to its center $\text{ct}_{\mathbb{X}}(q) \in X$, which is well-defined since abstract states are inifinity-norm balls. With some abuse of notation, we denote by q both an abstract state, i.e., $q \in \mathbb{X}$, and a subset of states, i.e., $q \subseteq X$.

As mentioned in the above section, we consider CPWA controllers (and hence neural network controllers) selected from a bounded polytopic set (namely a controller space) $\mathcal{P}^{K\times b} \subset \mathbb{R}^{m\times(n+1)}$. We partition the controller space $\mathcal{P}^{K\times b} \subset \mathbb{R}^{m\times(n+1)}$ into a finite set of controller partitions $\mathbb{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_M\}$ with a pre-specified grid size $\eta_{\mathcal{P}} \in \mathbb{R}^+$ (see Section 1.6 for the choice of $\eta_{\mathcal{P}}$). Each controller partition $\mathcal{P}_i \in \mathbb{P}$ is an infinity-norm ball centered around some $K_i \in \mathcal{P}^{K\times b}$ such that $\mathcal{P}^{K\times b} = \bigcup_{\mathcal{P} \in \mathbb{P}} \mathcal{P}$ and $\text{Int}(\mathcal{P}_i) \cap \text{Int}(\mathcal{P}_j) = \emptyset$ if $i \neq j$. Let $\text{ct}_{\mathbb{P}} : \mathbb{P} \to \mathcal{P}^{K\times b}$ map a controller partition $\mathcal{P} \in \mathbb{P}$ to its center $\text{ct}_{\mathbb{P}}(\mathcal{P}) \in \mathcal{P}^{K\times b}$. As mentioned in Section 1.3.1, we use the same notation $\mathcal{P}$ to denote both a subset of the parameters $K_i \in \mathcal{P}^{K\times b}$ and a subset of CPWA functions whose parameters K_i are chosen from $\mathcal{P}$.

Algorithm 2 outlines the training of a library of neural networks $\mathfrak{NN}$. Without knowing the exact robot dynamics (i.e., the stochastic kernel t), the workspace $\mathcal{W}$, and the specification φ, we use the formal training Algorithm 1 to train one neural network $\mathcal{NN}^{\theta}_{(q,\mathcal{P})}$ corresponding to each combination of controller partitions $\mathcal{P} \in \mathbb{P}$ and abstract states $q \in \mathbb{X}$ (line 4 in Algorithm 2). Thanks to the NN weight projection operator $\Pi_{\mathcal{P}}$, the neural networks $\mathcal{NN}^{\theta}_{(q,\mathcal{P})}$ satisfy the constraint in (1.7), i.e., $\mathcal{NN}^{\theta}_{(q,\mathcal{P})}|_q \in \mathcal{P}$. In the following, we use the notation $\mathcal{NN}_{(q,\mathcal{P})}$

by dropping the superscript θ for simplicity and refer to each neural network $\mathcal{NN}_{(q,\mathcal{P})}$ a local network.

To minimize the cost functional J, we implement the training approach `gradient-descent` (line 3 in Algorithm 1) based on Proximal Policy Optimization (PPO) [127] with the reward function as follows:

$$r(x, u) = -w_1 c(x, u) - w_2 \|u - \kappa(x)\|, \tag{1.24}$$

where $\kappa = \mathrm{ct}_{\mathbb{P}}(\mathcal{P})$ is the center of the assigned controller partition $\mathcal{P} \in \mathbb{P}$, $w_1, w_2 \in \mathbb{R}^+$ are pre-specified weights, and the state-action cost function $c : X \times U \to \mathbb{R}$ is from the definition of J in (1.5). Maximizing the above reward minimizes the cost $c(x, u)$ and encourages choosing controllers from the assigned controller partition $\mathcal{P}$. We assume access to a "nominal" simulator (i.e., the nominal dynamics f in (4.1)) for updating the robot states. Algorithm 2 returns a library $\mathfrak{NN}$ of $M \times N$ local networks, where M and N are the number of abstract states and the number of controller partitions, respectively. In Section 1.6, we reduce the number of local networks that need to be trained by employing transfer learning.

Algorithm 2 TRAIN-LIBRARY-NNS $(\mathbb{X}, \mathbb{P}, J)$

```
1: $\mathfrak{NN} = \{\}$
2: for $q \in \mathbb{X}$ do
3:     for $\mathcal{P} \in \mathbb{P}$ do
4:         $\mathcal{NN}_{(q,\mathcal{P})} = \texttt{Formal-Train}(q, \mathcal{P}, J)$
5:         $\mathfrak{NN} = \mathfrak{NN} \cup \{\mathcal{NN}_{(q,\mathcal{P})}\}$
6:     end for
7: end for
8: Return $\mathfrak{NN}$
```

1.4.2 Runtime Selection of Local NNs

In this subsection, we present our selection algorithm used at runtime when an arbitrary task $\mathcal{T} = (t, \varphi, \mathcal{W}, X_0)$ is given. The selection algorithm assigns one local neural network in the set $\mathfrak{NN}$ to each abstract state $\{q_1, \ldots, q_N\}$ in order to satisfy the given specification φ. Given a stochastic kernel t, our algorithm first computes a finite-state Markov Decision Process (MDP) that captures the closed-loop behavior of the robot under *all* possible CPWA controllers. Transitions in this finite-state MDP correspond to different subsets of CPWA functions in $\mathbb{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_M\}$. Thanks to the fact that the neural networks in the library $\mathfrak{NN}$ were trained using the formal training algorithm (Algorithm 1), *each neural network now represents a transition (symbol) in the finite-state MDP*. In other words, although neural networks are hard to interpret due to their construction, the formal training algorithm ensures the one-to-one mapping between these black-box neural networks and the transitions in the finite-state symbolic model.

Next, we use standard techniques in LTL-based motion planning to construct a finite-state automaton that captures the satisfaction of mission specifications φ. By analyzing the product between the finite-state MDP (that abstracts the robot dynamics) and the automaton corresponding to the specification φ, our algorithm decides which local networks in the set $\mathfrak{NN}$ need to be activated. We present details on the selection algorithm in the three steps below.

Step 1: Compute Symbolic Model

We construct a finite-state Markov decision process (MDP) $\hat{\Sigma} = (\mathbb{X}, \mathbb{X}_0, \mathbb{P}, \hat{t})$ of the robotic system $\Sigma = (X, X_0, U, t)$ as:

- $\mathbb{X} = \{q_1, \ldots, q_N\}$ is the set of abstract states;
- $\mathbb{X}_0 = \{q \in \mathbb{X} \mid q \subseteq X_0\}$ is the set of initial states;
- $\mathbb{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_M\}$ is the set of controller partitions;
- The transition probability from state $q \in \mathbb{X}$ to state $q' \in \mathbb{X}$ with label $\mathcal{P} \in \mathbb{P}$ is given

by:

$$\hat{t}(q'|q,\mathcal{P}) = \int_{q'} t(dx'|z,\kappa(z)) \tag{1.25}$$

where $z = \text{ct}_{\mathbb{X}}(q)$, $\kappa = \text{ct}_{\mathbb{P}}(\mathcal{P})$.

As explained in Section 1.2.2, the integral (1.25) can be easily computed since the stochastic kernel $t(\cdot|x,u)$ is a normal distribution, and we show techniques to accelerate the construction of the symbolic model $\hat{\Sigma}$ in Section 1.6. Such finite symbolic models have been used heavily in state-of-the-art LTL-based controller synthesis. Nevertheless, and unlike state-of-the-art LTL-based controllers, the control alphabet in $\hat{\Sigma}$ is *controller partitions* (i.e., subsets of CPWA functions). This is in contrast to LTL-based controllers in the literature (e.g., [147, 12]) that use subsets of control signals as their control alphabet.

We emphasize that our trained NN controllers are used to control the robotic system Σ with continuous state and action spaces, and the theoretical guarantees that we provide in Section 1.5 are also for the robotic system Σ, not for the finite-state MDP $\hat{\Sigma}$. As the motivation to introduce the symbolic model $\hat{\Sigma}$, our approach provides correctness guarantees for the NN-controlled robotic system Σ through (i) analyzing the behavior of the finite-state MDP $\hat{\Sigma}$ (in this section), and (ii) bounding the difference in behavior between the finite-state MDP $\hat{\Sigma}$ and the NN-controlled robotic system Σ (in Section 1.5). Critical to the latter step is the ability to restrict the NN's behavior thanks to the formal training proposed in Section 1.3.

Step 2: Construct Product MDP

Given a mission specification φ encoded in BLTL or scLTL formula, we construct the equivalent deterministic finite-state automaton (DFA) $\mathcal{A}_{\varphi} = (S, S_0, \mathbb{A}, G, \delta)$ as follows:

- S is a finite set of states;
- $S_0 \subseteq S$ is the set of initial states;
- $\mathbb{A}$ is an alphabet;
- $G \subseteq S$ is the accepting set;
- $\delta : S \times \mathbb{A} \to S$ is a transition function.

Such translation of BLTL and scLTL specifications to the equivalent DFA can be done using off-the-shelf tools (e.g., [81, 56]).

Given the finite-state MDP capturing the robot dynamics $\hat{\Sigma} = (\mathbb{X}, \mathbb{X}_0, \mathbb{P}, \hat{t})$ and the DFA $\mathcal{A}_\varphi = (S, S_0, \mathbb{A}, G, \delta)$ of the mission specification φ, we construct the product MDP $\hat{\Sigma} \otimes \mathcal{A}_\varphi = (\mathbb{X}^\otimes, \mathbb{X}_0^\otimes, \mathbb{P}, \mathbb{X}_G^\otimes, \hat{t}^\otimes)$ as follows:

- $\mathbb{X}^\otimes = \mathbb{X} \times S$ is a finite set of states;
- $\mathbb{X}_0^\otimes = \{(q_0, \delta(s_0, \hat{L}(q_0)) | q_0 \in \mathbb{X}_0, s_0 \in S_0\}$ is the set of initial states, where $\hat{L} : \mathbb{X} \to \mathbb{A}$ is the labeling function that assigns to each abstract state $q \in \mathbb{X}$ the subset of atomic propositions $\hat{L}(q) \in \mathbb{A}$ that evaluate *true* at q;
- $\mathbb{P}$ is the set of controller partitions;
- $\mathbb{X}_G^\otimes = \mathbb{X} \times G$ is the accepting set;
- The transition probability from state $(q, s) \in \mathbb{X}^\otimes$ to state $(q', s') \in \mathbb{X}^\otimes$ under $\mathcal{P} \in \mathbb{P}$ is given by:

$$\hat{t}^\otimes(q', s' | q, s, \mathcal{P}) = \begin{cases} \hat{t}(q' | q, \mathcal{P}) & \text{if } s' = \delta(s, \hat{L}(q')) \\ 0 & \text{else.} \end{cases}$$

Step 3: Select Local NNs by Dynamic Programming

Once constructed the product MDP $\hat{\Sigma} \otimes \mathcal{A}_\varphi$, the next step is to assign one local network $\mathcal{NN}_{(q,\mathcal{P})} \in \mathfrak{NN}$ to each abstract state $q \in \mathbb{X}$. In particular, the selection of NNs aims to maximize the probability of the finite-state MDP $\hat{\Sigma}$ satisfying the given specification φ. This can be formulated as finding the optimal policy that maximizes the probability of reaching the accepting set $\mathbb{X}_G^\otimes$ in the product MDP $\hat{\Sigma} \otimes \mathcal{A}_\varphi$. To that end, we define the optimal value functions $\hat{V}_k^* : \mathbb{X}^\otimes \to [0,1]$ that map a state $(q,s) \in \mathbb{X}^\otimes$ to the maximum probability of reaching the accepting set $\mathbb{X}_G^\otimes$ in $H-k$ steps from the state (q,s). When $k=0$, the optimal value function $\hat{V}_0^*$ yields the maximum probability of reaching the accepting set $\mathbb{X}_G^\otimes$ in H steps, i.e., the maximum probability of $\hat{\Sigma}$ satisfying φ. The optimal value functions can be solved by the following dynamic programming recursion:

$$\hat{Q}_k(q,s,\mathcal{P}) = \mathbf{1}_G(s) + \mathbf{1}_{S\setminus G}(s) \sum_{(q',s')\in\mathbb{X}^\otimes} \hat{V}_{k+1}^*(q',s')\hat{t}^\otimes(q',s'|q,s,\mathcal{P}) \tag{1.26}$$

$$\hat{V}_k^*(q,s) = \max_{\mathcal{P}\in\mathbb{P}} \hat{Q}_k(q,s,\mathcal{P}) \tag{1.27}$$

with the initial condition $\hat{V}_H^*(q,s) = \mathbf{1}_G(s)$ for all $(q,s) \in \mathbb{X}^\otimes$, where $k = H-1,\dots,0$.

Algorithm 3 summarizes the above three steps for selecting local NNs. Given a task $\mathcal{T} = (t,\varphi,\mathcal{W},X_0)$ at runtime, Algorithm 3 first computes the symbolic model $\hat{\Sigma}$ based on the stochastic kernel t, translates the mission specification φ to a DFA $\mathcal{A}_\varphi$ using off-the-shelf tools, and constructs the product MDP $\hat{\Sigma}\otimes\mathcal{A}_\varphi$ (line 1-3 in Algorithm 3). Then, Algorithm 3 solves the optimal policy for the product MDP $\hat{\Sigma} \otimes \mathcal{A}_\varphi$ using the DP recursion (2.8)-(2.9) (line 1-20 in Algorithm 3). At time step k, the optimal controller partition $\mathcal{P}^*$ at state $(q,s) \in \mathbb{X}^\otimes$ is given by the maximizer of $\hat{Q}_k(q,s,\mathcal{P})$ (line 16 in Algorithm 3). The last step is to assign a corresponding neural network to be applied given the robot states $x \in X$ and the DFA states $s \in S$. To that end, let:

$$\Gamma_k(x, s) = \hat{\Gamma}_k(\text{abs}(x), s),$$

where $\hat{\Gamma}_k$ maps the product MDP's states $(q, s) \in \mathbb{X}^{\otimes}$ to neural network's indices $(q, \mathcal{P}^*)$ (line 17 in Algorithm 3). In other words, given the robot states $x \in X$ and the DFA states $s \in S$ at time step k, we first find the abstract state $q \in \mathbb{X}$ that contains x, i.e., $q = \text{abs}(x)$, and then use the neural network $\mathcal{NN}_{(q,\mathcal{P}^*)} \in \mathfrak{NN}$ to control the robot at x, where $\hat{\Gamma}_k(q, s) = (q, \mathcal{P}^*)$. Recall that the neural networks in $\mathfrak{NN}$ are indexed as $(q, \mathcal{P})$ and hence the function $\Gamma(x, s) = \hat{\Gamma}_k(\text{abs}(x), s)$ computes such indices.

Algorithm 3 RUNTIME-SELECT ($\mathcal{T} = (t, \varphi, \mathcal{W}, X_0)$)

```
1: Compute the symbolic model $\hat{\Sigma} = (\mathbb{X}, \mathbb{X}_0, \mathbb{P}, \hat{t})$
2: Translate $\varphi$ to a DFA $\mathcal{A}_\varphi = (S, S_0, \mathbb{A}, G, \delta)$
3: Construct the product MDP $\hat{\Sigma} \otimes \mathcal{A}_\varphi$
4: for $(q, s) \in \mathbb{X}^{\otimes}$ do
5:     $\hat{V}^*_H(q, s) = \mathbf{1}_G(s)$
6: end for
7: $k = H - 1$
8: while $k \geq 0$ do
9:     for $(q, s) \in \mathbb{X}^{\otimes}$ do
10:        if $s \in G$ then
11:            $\hat{Q}_k(q, s, \mathcal{P}) = 1$
12:        else
13:            $\hat{Q}_k(q, s, \mathcal{P}) = \sum_{(q', s') \in \mathbb{X}^{\otimes}} \hat{V}^*_{k+1}(q', s') \hat{t}^{\otimes}(q', s' | q, s, \mathcal{P})$
14:        end if
15:        $\hat{V}^*_k(q, s) = \max_{\mathcal{P} \in \mathbb{P}} \hat{Q}_k(q, s, \mathcal{P})$
16:        $\mathcal{P}^* = \operatorname{argmax}_{\mathcal{P} \in \mathbb{P}} \hat{Q}_k(q, s, \mathcal{P})$
17:        $\hat{\Gamma}_k(q, s) = (q, \mathcal{P}^*)$
18:    end for
19:    $k = k - 1$
20: end while
21: Return $\{\hat{\Gamma}_k\}_{k \in \{0, \ldots, H-1\}}, \hat{V}^*_0, \hat{\Sigma} \otimes \mathcal{A}_\varphi$
```

1.4.3 Toy Example

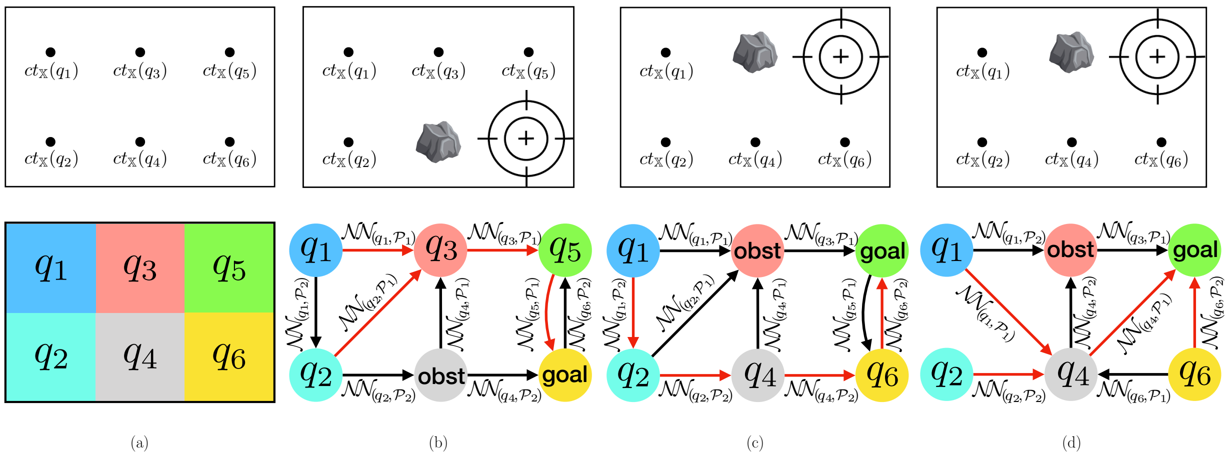


Figure 1.1: A toy example of a robot that navigates a two-dimensional workspace and needs to satisfy reach-avoid specifications $\varphi = \varphi_{\text{liveness}} \wedge \varphi_{\text{safety}}$ (see more details in Section 1.4.3).

We conclude this section by providing a toy example in Figure 1.1. Consider a mobile robot that navigates a two-dimensional workspace. We partition the state space $X \subset \mathbb{R}^2$ into six abstract states $\mathbb{X} = \{q_1, \ldots, q_6\}$ and discretize the controller space $\mathcal{P}^{K\times b}$ into two controller partitions $\mathbb{P} = \{\mathcal{P}_1, \mathcal{P}_2\}$. Figure 1.1 (a) shows the state space (top) and the abstract states $q_1, \ldots, q_6$ resulted from the partitioning (bottom), where the centers of abstract states are $\text{ct}_{\mathbb{X}}(q_1), \ldots, \text{ct}_{\mathbb{X}}(q_6)$.

During the offline training (Section 1.4.1), we use the formal training Algorithm 1 to obtain a library $\mathfrak{NN}$ consisting of 12 neural networks, i.e., $\mathfrak{NN} = \{\mathcal{NN}_{(q_i,\mathcal{P}_j)} | i \in \{1, \ldots, 6\}, j \in \{1, 2\}\}$.

We consider three different tasks $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$ that only become available at runtime after all the neural networks in $\mathfrak{NN}$ have been trained. Figure 1.1 (b), (c), and (d) show the workspaces for these three tasks, respectively. The specifications for these three tasks are $\varphi_1 = \Diamond_{[0,3]} (x \in q_6) \wedge \Box_{[0,3]} \neg (x \in q_4)$, $\varphi_2 = \Diamond_{[0,4]} (x \in q_5) \wedge \Box_{[0,4]} \neg (x \in q_3)$, and $\varphi_3 = \Diamond_{[0,3]} (x \in q_5) \wedge \Box_{[0,3]} \neg (x \in q_3)$, respectively. Finally, the three tasks have different robot dynamics t. Figure 1.1 (b)-(d) also depict the transitions in the resulting symbolic models, where we assume that all the transition probabilities $\hat{t}$ are 1 for simplicity (the transition probabilities $\hat{t}$ are computed as the integral of t in (1.25)). Thanks to the formal training Algorithm 1, the neural networks in $\mathfrak{NN}$ are guaranteed to be members of the CPWA functions in $\{\mathcal{P}_1, \mathcal{P}_2\}$. Hence, we label the transitions in the MDPs in Figure 1.1 (b)-(d) using $\mathcal{NN}_{(q_i,\mathcal{P}_j)}$ instead of $\{\mathcal{P}_1, \mathcal{P}_2\}$. While the transitions in the MDPs in Figure 1.1 (b) and (c) are the same, the MDP in Figure 1.1 (d) is different from that in Figure 1.1 (b) and (c) due to the difference in the robot dynamics in this task.

When the tasks $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$ become available, we use the runtime selection algorithm (Algorithm 3) to obtain the selection functions Γ_k. In Figure 1.1 (b)-(d), the selected NNs are the labels of the transitions marked in red. For example, in Figure 1.1 (b), our algorithm selects $\mathcal{NN}_{(q_1,\mathcal{P}_1)}$ to be used at all states $x \in q_1$. It is clear from the figures that the selected NNs are guaranteed to satisfy the given specifications φ_1, φ_2, and φ_3, respectively, regardless of the difference in the workspaces and robot dynamics.

1.5 Theoretical Guarantees

In this section, we study the theoretical guarantees of the proposed approach. We first provide a probabilistic guarantee for our NN-based planners on satisfying mission specifications given at runtime, then bound the difference between the NN-based planner and the optimal controller that maximizes the probability of satisfying the given specifications.

1.5.1 Generalization to Unseen Tasks

For an arbitrary task $\mathcal{T} = (t, \varphi, \mathcal{W}, X_0)$, let $\mathcal{NN}_{[\mathfrak{NN},\Gamma]}$ be the corresponding NN-based planner, where the library of networks $\mathfrak{NN}$ is trained by Algorithm 2 without knowing the task $\mathcal{T}$, and the activation map Γ denotes the time-dependent functions Γ_k obtained from Algorithm 3. As a key feature of $\mathcal{NN}_{[\mathfrak{NN},\Gamma]}$, the activation map Γ selects NNs based on both the robot states and the states of the $\mathcal{A}_\varphi$ DFA. This allows the NN-based planner $\mathcal{NN}_{[\mathfrak{NN},\Gamma]}$ to take into account the specification φ by tracking states of the DFA $\mathcal{A}_\varphi$. In comparison, a single state-feedback neural network $\mathcal{NN}: X \to U$ is not able to track the DFA states and hence cannot be trained to satisfy BLTL or scLTL specifications in general.

We denote by $\xi^{(x,s)}_{\mathcal{NN}_{[\mathfrak{NN},\Gamma]}}$ the closed-loop trajectory of a robot under the NN-based planner $\mathcal{NN}_{[\mathfrak{NN},\Gamma]}$ with the robot starting from state $x \in X_0$ and the DFA $\mathcal{A}_\varphi$ starting from state $s \in S_0$. Notice that though the symbolic model $\hat{\Sigma}$ is a finite-state MDP, the NN-based planner $\mathcal{NN}_{[\mathfrak{NN},\Gamma]}$ is used to control the robotic system Σ with continuous state and action spaces. The following theorem provides a probabilistic guarantee for the NN-controlled robotic system to satisfy mission specifications given at runtime.

Theorem 1.6. *Let $\hat{V}_0^*$ be the optimal value function returned by Algorithm 3. For arbitrary states $x \in X_0$ and $s \in S_0$, the probability of the closed-loop trajectory $\xi^{(x,s)}_{\mathcal{NN}_{[\mathfrak{NN},\Gamma]}}$ satisfying the given mission specification φ is bounded as follows:*

$$\left|\Pr\left(\xi^{(x,s)}_{\mathcal{NN}_{[\mathfrak{NN},\Gamma]}} \models \varphi\right) - \hat{V}_0^*(q,s)\right| \leq HZ\Delta^{\mathcal{NN}} \tag{1.28}$$

where $q = \mathrm{abs}(x)$ and

$$\Delta^{\mathcal{NN}} = \max_{i \in \{1,\dots,N\}} \left(\Lambda_i \eta_q + B_i L_i \eta_q + \sqrt{m(n+1)}\mathcal{L}_X B_i \eta_{\mathcal{P}}\right). \tag{1.29}$$

Recall that η_q and $\eta_{\mathcal{P}}$ are the grid sizes used for partitioning the state space and the controller space, respectively. The upper bound $HZ\Delta^{\mathcal{NN}}$ in Theorem 1.6 can be arbitrarily small by tuning the grid sizes η_q and $\eta_{\mathcal{P}}$. In (1.28)-(1.29), H is the time horizon, $N = |\mathbb{X}|$ is the number of abstract states, and $Z = |S|$ is the number of the $\mathcal{A}_\varphi$ DFA states. The parameters Λ_i and B_i are given by $\Lambda_i = \int_X \lambda_i(y)\mu(dy)$ and $B_i = \int_X \beta_i(y)\mu(dy)$, where $\lambda_i(y)$ and $\beta_i(y)$ are the Lipschitz constants of the stochastic kernel $t : \mathcal{B}(X) \times X \times U \to [0,1]$, i.e., $\forall x, x' \in q_i$, $\forall u \in U$: $|t(dy|x',u) - t(dy|x,u)| \leq \lambda_i(y)\|x' - x\|\mu(dy)$, and $\forall x \in q_i$, $\forall u, u' \in U$:

$|t(dy|x,u') - t(dy|x,u)| \leq \beta_i(y)\|u'-u\|\mu(dy)$. Furthermore, L_i is the Lipschitz constant of the local neural networks at abstract state $q_i \in \mathbb{X}$, i.e., $\forall \mathcal{P} \in \mathbb{P}$, $\forall x, x' \in q_i$:

$$\|\mathcal{NN}_{(q_i,\mathcal{P})}(x) - \mathcal{NN}_{(q_i,\mathcal{P})}(x')\| \leq L_i\|x - x'\|.$$

Finally, $\sup_{x\in X}\|x\| \leq \mathcal{L}_X$, $\sup_{K\in\mathcal{P}^{K\times b}}\|K\| \leq \mathcal{L}_{\mathcal{P}}$, and n, m are the dimensions of $X \subset \mathbb{R}^n$, $U \subset \mathbb{R}^m$, respectively.

We now provide proofs of Theorem 1.6 and Theorem 1.7 in Section 1.5. Let $\Sigma = (X, X_0, U, t)$ be a robotic system with continuous state and action spaces and $\mathcal{A}_\varphi = (S, S_0, \mathbb{A}, G, \delta)$ be the DFA of a mission specification φ. Similar to the product MDP $\hat{\Sigma} \otimes \mathcal{A}_\varphi$, the product between Σ and $\mathcal{A}_\varphi$ is given by $\Sigma \otimes \mathcal{A}_\varphi = (X^\otimes, X_0^\otimes, U, X_G^\otimes, t^\otimes)$, where:

- $X^\otimes = X \times S$ is the state space;
- $X_0^\otimes = \{(x_0, \delta(s_0, L(x_0))|x_0 \in X_0, s_0 \in S_0\}$ is the set of initial states, where $L : X \to \mathbb{A}$ is the labeling function that assigns to each state $x \in X$ the subset of atomic propositions $L(x) \in \mathbb{A}$ that evaluate *true* at x;
- $U \subset \mathbb{R}^m$ is the control action space;
- $X_G^\otimes = X \times G$ is the accepting set;
- The stochastic kernel $t^\otimes$ is given by:

$$t^\otimes(dx', s'|x, s, u) = \begin{cases} t(dx'|x,u) & \text{if } s' = \delta(s, L(x')) \\ 0 & \text{else.} \end{cases}$$

Proof. Given the NN-based planner $\mathcal{NN}_{[\mathfrak{M}\mathfrak{M},\Gamma]}$ obtained using our framework, we define functions $V_k^{\mathcal{NN}} : X^\otimes \to [0,1]$ that map a state $(x,s) \in X^\otimes$ to the probability of reaching the accepting set $X_G^\otimes$ in $H-k$ steps from the state (x,s) and under the control of $\mathcal{NN}_{[\mathfrak{M}\mathfrak{M},\Gamma]}$.

With this notation, we have $V_0^{\mathcal{NN}}(x,s) = \Pr\left(\xi^{(x,s)}_{\mathcal{NN}_{[\mathfrak{NN},\Gamma]}} \models \varphi\right)$ since reaching the accepting set $X_G^{\otimes}$ in H steps in the product MDP $\Sigma \otimes \mathcal{A}_\varphi$ is equivalent to Σ satisfying φ. In the following, we show that for any $x \in q$ and $k = 0, \dots, H$:

$$|V_k^{\mathcal{NN}}(x,s) - \hat{V}_k^*(q,s)| \leq (H-k)Z\Delta^{\mathcal{NN}}, \tag{1.30}$$

which yields (1.28) by letting $k = 0$. By the definition of $V_k^{\mathcal{NN}}$, the probabilities of reaching the accepting set $X_G^{\otimes}$ under the NN-based planner $\mathcal{NN}_{[\mathfrak{NN},\Gamma]}$ can be expressed as:

$$V_k^{\mathcal{NN}}(x,s) = \mathbf{1}_G(s) + \mathbf{1}_{S\setminus G}(s) \sum_{s' \in S} \int_X V_{k+1}^{\mathcal{NN}}(x',s') t^{\otimes}(dx', s' | x, s, \mathcal{NN}(x)) \tag{1.31}$$

with the initial condition $V_H^{\mathcal{NN}}(x,s) = \mathbf{1}_G(s)$. In the stochastic kernel $t^{\otimes}$ in (1.31), we use $\mathcal{NN}$ to denote the local network selected by the activation map Γ_{k+1} at the state (x,s) for simplicity. Though solving (1.31) is intractable due to the continuous state space, we can bound the difference between $V_k^{\mathcal{NN}}$ and $\hat{V}_k^*$ as (1.30) by induction.

For the base case $k = H$, (1.30) trivially holds since $V_H^{\mathcal{NN}}(x,s) = \mathbf{1}_G(s)$ and $\hat{V}_H^*(q,s) = \mathbf{1}_G(s)$. For the induction hypothesis, suppose for $k+1$ it holds that:

$$|V_{k+1}^{\mathcal{NN}}(x,s) - \hat{V}_{k+1}^*(q,s)| \leq (H-k-1)Z\Delta^{\mathcal{NN}}. \tag{1.32}$$

Let $\bar{V}_k^*$ be a piecewise constant interpolation of $\hat{V}_k^*$ defined by $\bar{V}_k^*(x,s) = \hat{V}_k^*(q,s)$ for any $x \in q$ and any $s \in S$. Then,

$$|V_k^{\mathcal{NN}}(x,s) - \hat{V}_k^*(q,s)| \leq |V_k^{\mathcal{NN}}(x,s) - V_k^{\mathcal{NN}}(z,s)| + |V_k^{\mathcal{NN}}(z,s) - \bar{V}_k^*(z,s)| \tag{1.33}$$

where $z = \mathrm{ct}_{\mathbb{X}}(q)$ and $x \in q$. For the first term on the RHS:

$$\begin{aligned}
&|V_k^{\mathcal{NN}}(x,s) - V_k^{\mathcal{NN}}(z,s)| \\
&= |\mathbf{1}_G(s) + \mathbf{1}_{S\setminus G}(s) \sum_{s'\in S} \int_X V_{k+1}^{\mathcal{NN}}(x',s') t^{\otimes}(dx',s'|x,s,\mathcal{NN}(x)) \\
&\quad - \mathbf{1}_G(s) + \mathbf{1}_{S\setminus G}(s) \sum_{s'\in S} \int_X V_{k+1}^{\mathcal{NN}}(x',s') t^{\otimes}(dx',s'|z,s,\mathcal{NN}(z))| \\
&\le \sum_{s'\in S} \int_X V_{k+1}^{\mathcal{NN}}(x',s') |t^{\otimes}(dx',s'|x,s,\mathcal{NN}(x)) - t^{\otimes}(dx',s'|z,s,\mathcal{NN}(z)| \\
&\le Z \int_X |t(dx'|x,\mathcal{NN}(x)) - t(dx'|z,\mathcal{NN}(z)| \\
&\le Z \int_X |t(dx'|x,\mathcal{NN}(x)) - t(dx'|z,\mathcal{NN}(x))| + |t(dx'|z,\mathcal{NN}(x)) - t(dx'|z,\mathcal{NN}(z))| \\
&\le Z\Lambda_i \|x - z\| + ZB_i \|\mathcal{NN}(x) - \mathcal{NN}(z)\| \\
&\le Z\Lambda_i \eta_q + ZB_i L_i \eta_q. && (1.34)
\end{aligned}$$

For the second term on the RHS of (1.33):

$$\begin{aligned}
&|V_k^{\mathcal{NN}}(z,s) - \bar{V}_k^{*}(z,s)| \\
&= |\mathbf{1}_G(s) + \mathbf{1}_{S\setminus G}(s) \sum_{s'\in S} \int_X V_{k+1}^{\mathcal{NN}}(x',s') t^{\otimes}(dx',s'|z,s,\mathcal{NN}(z)) \\
&\quad - \mathbf{1}_G(s) + \mathbf{1}_{S\setminus G}(s) \max_{\mathcal{P}\in\mathbb{P}} \sum_{(q',s')\in\mathbb{X}^{\otimes}} \hat{V}_{k+1}^{*}(q',s') \hat{t}^{\otimes}(q',s'|q,s,\mathcal{P})| && (1.35) \\
&\le |\sum_{s'\in S} \int_X V_{k+1}^{\mathcal{NN}}(x',s') t^{\otimes}(dx',s'|z,s,\mathcal{NN}(z)) - \sum_{s'\in S}\sum_{q'\in\mathbb{X}} \hat{V}_{k+1}^{*}(q',s') \hat{t}^{\otimes}(q',s'|q,s,\mathcal{P}^*)| && (1.36) \\
&\le |\sum_{s'\in S} \int_X V_{k+1}^{\mathcal{NN}}(x',s') t^{\otimes}(dx',s'|z,s,\mathcal{NN}(z)) - \sum_{s'\in S} \int_X \bar{V}_{k+1}^{*}(x',s') t^{\otimes}(dx',s'|z,s,\mathrm{ct}_{\mathbb{P}}(\mathcal{P}^*)(z))| \\
& && (1.37) \\
&\le \sum_{s'\in S} \int_X |V_{k+1}^{\mathcal{NN}}(x',s') - \bar{V}_{k+1}^{*}(x',s')| t^{\otimes}(dx',s'|z,s,\mathcal{NN}(z)) \\
&+ \sum_{s'\in S} \int_X \bar{V}_{k+1}^{*}(x',s') |t^{\otimes}(dx',s'|z,s,\mathcal{NN}(z)) - t^{\otimes}(dx',s'|z,s,\mathrm{ct}_{\mathbb{P}}(\mathcal{P}^*)(z))| && (1.38) \\
&\le (H-k-1) Z \Delta^{\mathcal{NN}} + Z\sqrt{m(n+1)} \mathcal{L}_X B_i \eta_{\mathcal{P}} && (1.39)
\end{aligned}$$

where (1.35) uses the DP recursion (2.8)-(2.9), in (1.36) $\mathcal{P}^*$ denotes the maximizer, and (1.37) uses the definition of $\hat{t}$ in (1.25) with $z = \mathrm{ct}_{\mathbb{X}}(q)$. In (1.39), we use the induction hypothesis (1.32), and the inequality $\|K(x) - K'(x)\| \leq \|K - K'\| \|x\| \leq \sqrt{m(n+1)} \|K - K'\|_{\max} \mathcal{L}_X \leq \sqrt{m(n+1)} \eta_{\mathcal{P}} \mathcal{L}_X$, where $\|K - K'\|_{\max} \leq \eta_{\mathcal{P}}$ since the local network $\mathcal{NN}$ selected by the activation map Γ represents a CPWA function from the maximizer $\mathcal{P}^\star$, i.e., $K, K' \in \mathcal{P}^\star \subset \mathbb{R}^{m \times (n+1)}$. Substitute (1.34) and (1.39) into (1.33) yields (1.30). $\square$

1.5.2 Optimality Guarantee

Next, we compare our NN-based planner $\mathcal{NN}_{[\mathfrak{NN},\Gamma]}$ with the optimal controller (not necessarily a neural network) that maximizes the probability of satisfying the given specification φ. To that end, we provide an upper bound on the difference in the probabilities of satisfying φ without explicit computing of the optimal controller. Let $\mathcal{C}_\varphi^* : X \times S \to U$ be the optimal controller and $\xi_{\mathcal{C}_\varphi^*}^{(x,s)}$ be the closed-loop trajectory of the robotic system $\Sigma = (X, X_0, U, t)$ controlled by $\mathcal{C}_\varphi^*$. Similar to the NN-based planner $\mathcal{NN}_{[\mathfrak{NN},\Gamma]}$, the optimal controller $\mathcal{C}_\varphi^*$ applies to the robotic system Σ with continuous state and action spaces, and takes the DFA states $s \in S$ into consideration when computing control actions. Synthesizing the optimal controller $\mathcal{C}_\varphi^*$ for a mission specification φ is computationally prohibitive due to the continuous state and action spaces. Without explicitly computing $\mathcal{C}_\varphi^*$, the following theorem tells how close our NN-based planner $\mathcal{NN}_{[\mathfrak{NN},\Gamma]}$ is to the optimal controller $\mathcal{C}_\varphi^*$ in terms of satisfying the specification φ. By tuning the grid sizes η_q and $\eta_{\mathcal{P}}$, our NN-based planner $\mathcal{NN}_{[\mathfrak{NN},\Gamma]}$ can be arbitrarily close to the optimal controller $\mathcal{C}_\varphi^*$.

Theorem 1.7. *For arbitrary states $x \in X_0$ and $s \in S_0$, the difference in the probabilities of the closed-loop trajectories $\xi_{\mathcal{NN}_{[\mathfrak{NN},\Gamma]}}^{(x,s)}$ and $\xi_{\mathcal{C}_\varphi^*}^{(x,s)}$ satisfying the given mission specification φ is upper bounded as follows:*

$$\left| \Pr\left(\xi_{\mathcal{NN}_{[\mathfrak{NN},\Gamma]}}^{(x,s)} \models \varphi \right) - \Pr\left(\xi_{\mathcal{C}_\varphi^*}^{(x,s)} \models \varphi \right) \right| \leq HZ(\Delta^{\mathcal{NN}} + \Delta^*) \tag{1.40}$$

where $\Delta^{\mathcal{NN}}$ *is given by* (1.29) *and*

$$\Delta^* = \max_{i \in \{1,\dots,N\}} \left(\Lambda_i \eta_q + B_i \mathcal{L}_{\mathcal{P}} \eta_q + 2\sqrt{m(n+1)} \mathcal{L}_X B_i \eta_{\mathcal{P}} \right). \tag{1.41}$$

Proof. Let functions $V_k^* : X^{\otimes} \to [0,1]$ map a state $(x,s) \in X^{\otimes}$ to the probability of reaching the accepting set $X_G^{\otimes}$ in $H-k$ steps from the state (x,s) and under the optimal controller $\mathcal{C}_\varphi^* : X \times S \to U$. Then, $V_0^*(x,s) = \Pr\left(\xi_{\mathcal{C}_\varphi^*}^{(x,s)} \models \varphi\right)$ since reaching the accepting set $X_G^{\otimes}$ in H steps in the product MDP $\Sigma \otimes \mathcal{A}_\varphi$ is equivalent to Σ satisfying φ. The optimal probabilities of reaching the accepting set $X_G^{\otimes}$ can be expressed using DP recursion:

$$Q_k(x,s,u) = \mathbf{1}_G(s) + \mathbf{1}_{S \setminus G}(s) \sum_{s' \in S} \int_X V_{k+1}^*(x',s') t^{\otimes}(dx', s' | x, s, u) \tag{1.42}$$

$$V_k^*(x,s) = \max_{u \in U} Q_k(x,s,u) \tag{1.43}$$

Though solving V_k^* and the corresponding optimal controller $\mathcal{C}_\varphi^*$ is intractable due to the continuous state and action spaces, we can bound the difference between V_k^* and $\hat{V}_k^*$ by induction similar to the proof of (1.30) in Theorem 1.6. We skip the details and directly give the following bound:

$$|V_k^*(x,s) - \hat{V}_k^*(q,s)| \le (H-k) Z \Delta^*, \tag{1.44}$$

where $x \in q$ and Δ^* is given by (1.41). With (1.30) and (1.44), we have:

$$\begin{aligned} |V_k^{\mathcal{NN}}(x,s) - V_k^*(x,s)| &\le |V_k^{\mathcal{NN}}(x,s) - \hat{V}_k^*(q,s)| + |V_k^*(x,s) - \hat{V}_k^*(q,s)| \\ &\le (H-k) Z (\Delta^{\mathcal{NN}} + \Delta^*), \end{aligned} \tag{1.45}$$

which yields (1.40) by letting $k = 0$. □

1.6 Effective Adaptation

In this section, we focus on practical issues of the proposed approach and present some key elements for performance improvement while maintaining the same theoretical guarantees as Section 1.5. Firstly, we show that the proposed composition of neural networks leads to an effective way to adapt previous learning experiences to unseen tasks. In particular, instead of training the whole library of neural networks $\mathfrak{N}\mathfrak{N}$ in Algorithm 2, we only train a subset of networks $\mathfrak{N}\mathfrak{N}_{\text{part}} \subseteq \mathfrak{N}\mathfrak{N}$ based on tasks provided for training. Obtaining this subset $\mathfrak{N}\mathfrak{N}_{\text{part}}$ can be viewed as a systematic way to store learning experiences, which are adapted to unseen tasks via transfer learning (see Section 1.6.1). Secondly, we propose a data-driven approach to accelerate the construction of the symbolic model $\hat{\Sigma}$ (see Section 1.6.2). Finally, we comment on the choice of grid sizes η_q and $\eta_{\mathcal{P}}$ for partitioning the state and action spaces (see Section 1.6.3).

1.6.1 Accelerate by Transfer Learning

Consider a meta-RL problem with a set of training tasks $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_d\}$ that are provided for training neural networks in the hope of fast adaptation to unseen tasks $\mathcal{T}_{\text{test}}$ during the test phase, where each task is a tuple $\mathcal{T} = (t, \varphi, \mathcal{W}, X_0)$ as defined before. We consider the problem of how to leverage the learning experiences from the training tasks to accelerate the learning of the unseen test tasks. Our intuition is that when the training tasks have enough variety, the local behavior for fulfilling a test task $\mathcal{T}_{\text{test}}$ should be close to the local behavior for fulfilling some training task $\mathcal{T}_{\text{train}} \in \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_d\}$. In other words, the controller needed by a robot to fulfill the test task $\mathcal{T}_{\text{test}}$ should be close to the controller used for fulfilling some training task $\mathcal{T}_{\text{train}} \in \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_d\}$, where the training task $\mathcal{T}_{\text{train}}$ can be *different* in different subsets of the state space X. This is more general than the prevalent assumption in the meta-RL literature that the test task's controller is close to the *same* training task's controller everywhere in the state space. As a result, our approach requires less variety of the training tasks $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_d\}$ for fast adaptation to unseen tasks.

The form of the composed NN-based planner $\mathcal{NN}_{[\mathfrak{N}\mathfrak{N},\Gamma]}$ provides a systematic way to store learning experiences from all the training tasks and enables to select which training task should be adapted to the test task based on the current state of the robot. Given a set of training tasks $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_d\}$, Algorithm 4 trains a subset of local networks $\mathfrak{N}\mathfrak{N}_{\text{part}} \subseteq \mathfrak{N}\mathfrak{N}$ suggested by the training tasks. For each training task $\mathcal{T}_{\text{train}} \in \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_d\}$, Algorithm 4 first calls `Runtime-Select` (i.e., Algorithm 3) to compute the corresponding activation maps $\hat{\Gamma}_k$ (line 3 in Algorithm 4). The activation maps $\hat{\Gamma}_k$ are then used to determine which local networks $\mathcal{NN}_{(q,\mathcal{P})}$ need to be trained at each state $(q, s) \in \mathbb{X}^{\otimes}$ of the product MDP $\hat{\Sigma} \otimes \mathcal{A}_{\varphi}$ (line 5 in Algorithm 4). The local neural networks are trained using the method `Formal-Train` given by Algorithm 1 (line 7 in Algorithm 4). Compared to Algorithm 2 that trains all the neural networks to obtain the library $\mathfrak{N}\mathfrak{N}$, Algorithm 4 reduces the number of NNs need to be trained by leveraging the training tasks $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_d\}$.

During the test phase, we adapt previous learning experiences stored in the subset of networks $\mathfrak{N}\mathfrak{N}_{\text{part}}$ to test tasks $\mathcal{T}_{\text{test}}$ by employing transfer learning. In particular, if a local NN needed by the test task $\mathcal{T}_{\text{test}}$ has not been trained, we fast learn it by fine-tuning the "closest" NN to it in the subset $\mathfrak{N}\mathfrak{N}_{\text{part}}$. Thanks to the fact that each local network $\mathcal{NN}_{(q,\mathcal{P})}$ is associated with an abstract state $q \in \mathbb{X}$ and a controller partition $\mathcal{P} \in \mathbb{P}$, we can define the distance between two local networks $\mathcal{NN}_{(q_1,\mathcal{P}_1)}$ and $\mathcal{NN}_{(q_2,\mathcal{P}_2)}$ as follows:

$$\text{Dist}\left(\mathcal{NN}_{(q_1,\mathcal{P}_1)}, \mathcal{NN}_{(q_2,\mathcal{P}_2)}\right) = \alpha_1 \|\text{ct}_{\mathbb{X}}(q_1) - \text{ct}_{\mathbb{X}}(q_2)\| + \alpha_2 \|\text{ct}_{\mathbb{P}}(\mathcal{P}_1) - \text{ct}_{\mathbb{P}}(\mathcal{P}_2)\|_{\max} \quad (1.46)$$

with pre-specified weights $\alpha_1, \alpha_2 \in \mathbb{R}^+$. Given a test task $\mathcal{T}_{\text{test}}$, Algorithm 5 first computes the corresponding activation maps $\hat{\Gamma}_k$ (line 1 in Algorithm 5), and then selects local networks $\mathcal{NN}_{(q,\mathcal{P})}$ to be applied at each time step until reaching the product MDP's accepting set $\mathbb{X}^{\otimes}_{G}$

(line 3-4 in Algorithm 5). If the needed network $\mathcal{NN}_{(q,\mathcal{P})}$ has not been trained, Algorithm 5 initializes the missing network $\mathcal{NN}_{(q,\mathcal{P})}$ using the weights of the closest network $\mathcal{NN}_{(q^*,\mathcal{P}^*)}$ to it in the subset $\mathfrak{NN}_{\text{part}}$, where the distance metric between neural networks is given by (1.46) (line 5-7 in Algorithm 5). After that, the algorithm trains the missing network $\mathcal{NN}_{(q,\mathcal{P})}$ using PPO with only a few episodes for fine-tuning (line 8 in Algorithm 5). Thanks to the NN weight projection operator $\Pi_{\mathcal{P}}$, the resulting NN-based planner enjoys the same theoretical guarantees presented in Section 1.5 (line 9-10 in Algorithm 5).

Algorithm 4 TRAIN-TRANSFER $(\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_d\}, J)$

```
1: $\mathfrak{NN}_{\text{part}} = \{\}$
2: for $\mathcal{T}_{\text{train}} \in \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_d\}$ do
3:     $\hat{\Gamma}_k, \hat{V}_0^*, \hat{\Sigma} \otimes \mathcal{A}_\varphi = \texttt{Runtime-Select}(\mathcal{T}_{\text{train}})$
4:     for $(q, s) \in \mathbb{X}^{\otimes}$, $k \in \{0, \ldots, H-1\}$ do
5:         $(q, \mathcal{P}) = \hat{\Gamma}_k(q, s)$
6:         if $\mathcal{NN}_{(q,\mathcal{P})} \notin \mathfrak{NN}_{\text{part}}$ then
7:             $\mathcal{NN}_{(q,\mathcal{P})} = \texttt{Formal-Train}(q, \mathcal{P}, J)$
8:             $\mathfrak{NN}_{\text{part}} = \mathfrak{NN}_{\text{part}} \cup \{\mathcal{NN}_{(q,\mathcal{P})}\}$
9:         end if
10:    end for
11: end for
12: Return $\mathfrak{NN}_{\text{part}}$
```

1.6.2 Data-Driven Symbolic Model

Recall that in Algorithm 3, after knowing the robot dynamics (i.e., the stochastic kernel t), the first step is to construct the symbolic model $\hat{\Sigma} = (\mathbb{X}, \mathbb{X}_0, \mathbb{P}, \hat{t})$ (line 1 in Algorithm 3). The construction of $\hat{\Sigma}$ requires to compute the transition probabilities $\hat{t}(q'|q, \mathcal{P}) = \int_{q'} t(dx'|z, \kappa(z))$ with all controller partitions $\mathcal{P} \in \mathbb{P}$ at each abstract state $q \in \mathbb{X}$, where $z = \text{ct}_{\mathbb{X}}(q)$, $\kappa = \text{ct}_{\mathbb{P}}(\mathcal{P})$. Reducing the computation of transition probabilities is tempting when the number of controller partitions is large, especially if the stochastic kernel $t(\cdot|x, u)$ is not a normal distribution and needs numerical integration. In this subsection, we accelerate the construction of $\hat{\Sigma}$ in a data-driven manner.

Algorithm 5 RUNTIME-TRANSFER ($\mathcal{T}_{\text{test}}, \mathfrak{NN}_{\text{part}}, J, x, s$)

```
1:  $\hat{\Gamma}_k, \hat{V}_0^*, \hat{\Sigma} \otimes \mathcal{A}_\varphi = \texttt{Runtime-Select}(\mathcal{T}_{\text{test}})$
2:  $k = 0$, $q = \text{abs}(x)$
3:  **while** $(q, s) \notin \mathbb{X}_G^{\otimes}$ **do**
4:      $(q, \mathcal{P}) = \hat{\Gamma}_k(q, s)$
5:      **if** $\mathcal{NN}_{(q,\mathcal{P})} \notin \mathfrak{NN}_{\text{part}}$ **then**
6:          $\mathcal{NN}_{(q^*,\mathcal{P}^*)} = \underset{\mathcal{NN}_{(q_1,\mathcal{P}_1)} \in \mathfrak{NN}_{\text{part}}}{\text{argmin}} \text{Dist}\left(\mathcal{NN}_{(q_1,\mathcal{P}_1)}, \mathcal{NN}_{(q,\mathcal{P})}\right)$
7:          $\mathcal{NN}_{(q,\mathcal{P})} = \texttt{initialize}(\mathcal{NN}_{(q^*,\mathcal{P}^*)})$
8:          $\mathcal{NN}_{(q,\mathcal{P})} = \texttt{PPO-update}(\mathcal{NN}_{(q,\mathcal{P})}, J)$
9:          $\widehat{W}^{(F)}, \widehat{b}^{(F)} = \Pi_{\mathcal{P}}(\mathcal{NN}_{(q,\mathcal{P})})$
10:         Set $\mathcal{NN}_{(q,\mathcal{P})}$ output layer weights be $\widehat{W}^{(F)}, \widehat{b}^{(F)}$
11:         $\mathfrak{NN}_{\text{part}} = \mathfrak{NN}_{\text{part}} \cup \{\mathcal{NN}_{(q,\mathcal{P})}\}$
12:     **end if**
13:     $u = \mathcal{NN}_{(q,\mathcal{P})}(x)$
14:     Apply action $u$, observe the new state $x$
15:     $q = \text{abs}(x)$, $s = \delta(s, L(x))$
16:     $k = k + 1$
17: **end while**
```

For a given task $\mathcal{T}$, we consider our algorithm has access to a set of expert-provided trajectories $\mathcal{D} = \{\xi_1, \xi_2, \dots, \xi_c\}$, such as human demonstrations that fulfill the task $\mathcal{T}$. Instead of computing all the transition probabilities $\hat{t}(q'|q, \mathcal{P})$, we use the set of expert trajectories $\mathcal{D}$ to guide the computation of transitions. The resulting symbolic model can be viewed as a symbolic representation of the expert trajectories in $\mathcal{D}$.

In Algorithm 6, we first use imitation learning to train a neural network $\mathcal{NN}$ by imitating the expert trajectories in $\mathcal{D}$ (line 1 in Algorithm 6). Though the neural network $\mathcal{NN}$ trained using a limited dataset $\mathcal{D}$ may not always fulfill the task $\mathcal{T}$, the network $\mathcal{NN}$ contains relevant control actions that can be used to obtain the final controller. In particular, at each abstract state $q \in \mathbb{X}$, we only compute transition probabilities $\hat{t}(q'|q, \mathcal{P})$ with controller partitions $\mathcal{P}$ suggested by the network $\mathcal{NN}$. To be specific, let u^* be the control actions given by the network $\mathcal{NN}$ at the centers of abstract states $q \in \mathbb{X}$ (line 3 in Algorithm 6). Then, Algorithm 6 selects a subset $P_q \subseteq \mathbb{P}$ consists of I controller partitions that yield control actions close to the NN's output u^*, where $I \in \mathbb{N}$ is a user-defined parameter (line 4-8 in

Algorithm 6). Finally, Algorithm 6 computes a symbolic model $\hat{\Sigma}$ with only transitions under the controller partitions in the subset P_q (line 9 in Algorithm 6). The symbolic model $\hat{\Sigma}$ contains more transitions by increasing the parameter I at the cost of computational efficiency. The choice of I can be adaptively determined as discussed in the next subsection.

Algorithm 6 CONSTRUCT-SYMBOL-MODEL $(\mathcal{T}, \mathcal{D}, \mathbb{X}, \mathbb{P}, I)$

```
 1: $\mathcal{NN}$ = imitation-learning($\mathcal{D}$)
 2: for $q \in \mathbb{X}$ do
 3:     $u^* = \mathcal{NN}(z)$, where $z = \mathrm{ct}_{\mathbb{X}}(q)$
 4:     $P_q = \{\}$
 5:     for $i = 1, \ldots, I$ do
 6:         $\mathcal{P}^* = \underset{\mathcal{P} \in \mathbb{P} \setminus P_q}{\mathrm{argmin}} \|\kappa(z) - u^*\|$, s.t. $\kappa = \mathrm{ct}_{\mathbb{P}}(\mathcal{P}), z = \mathrm{ct}_{\mathbb{X}}(q)$
 7:         $P_q = P_q \cup \{\mathcal{P}^*\}$
 8:     end for
 9:     Compute $\hat{t}(q'|q, \mathcal{P})$ with $\mathcal{P} \in P_q$
10: end for
11: Return $\hat{\Sigma}$
```

1.6.3 Adaptive Partitioning

Recall that during the offline training, we partition the state space $X \subset \mathbb{R}^n$ and the controller space $\mathcal{P}^{K \times b} \subset \mathbb{R}^{m \times (n+1)}$ using the pre-specified parameters η_q and $\eta_{\mathcal{P}}$, respectively (see Section 1.4.1). In this subsection, we comment on the choice of the grid sizes η_q and $\eta_{\mathcal{P}}$. In particular, our framework can directly incorporate the discretization techniques from the literature of abstraction-based controller synthesis (e.g. [40, 62]). To that end, we provide a simple yet efficient example of adaptive partitioning in Algorithm 7, which enables the update of gird sizes η_q and $\eta_{\mathcal{P}}$ at runtime using transfer learning.

The first part of Algorithm 7 aims to partition the state and controller spaces such that the resulting probabilities $\hat{V}_0^*(q, s)$ of satisfying the specification φ are greater than the pre-specified threshold p at all initial states $(q, s) \in \mathbb{X}_0 \times S_0$ (line 1-7 in Algorithm 7). In particular, if the probability $\hat{V}_0^*(q, s)$ is less than p at some state $(q, s) \in \mathbb{X}_0 \times S_0$, Algorithm 7 decreases the current grid sizes η_q and $\eta_{\mathcal{P}}$ by half and increases the parameter I (line 6 in Algorithm 7). After having such a partitioning of the state and controller spaces, Algorithm 7 trains the corresponding locals networks by fine-tuning the NNs in the provided library of networks $\mathfrak{NN}_{\text{part}}$ (line 8-18 in Algorithm 7). The following theoretical guarantee for the resulting NN-based planner to satisfy the given specification φ directly follows Theorem 1.6.

Corollary 1.8. *Consider Algorithm 7 returns a library of local networks $\mathfrak{NN}_{part}$ and an activation map Γ (denoting the functions $\hat{\Gamma}_k$). Then, the NN-based planner $\mathcal{NN}_{[\mathfrak{NN}_{part},\Gamma]}$ satisfying $\Pr\left(\xi^{(x,s)}_{\mathcal{NN}_{[\mathfrak{NN}_{part},\Gamma]}} \models \varphi\right) \geq p - \varepsilon$ for any $x \in X_0$ and $s \in S_0$, where $\varepsilon = HZ\Delta^{\mathcal{NN}}$ and $\Delta^{\mathcal{NN}}$ is given by* (1.29).

Algorithm 7 Adapt-Partition $(\mathcal{T}, \mathcal{D}, \mathfrak{NN}_{\text{part}}, J, \eta_q, \eta_{\mathcal{P}}, I)$

```
1: while $\hat{V}^*_{\min} < p$ do
2:     $\mathbb{X} = \texttt{partition}(X, \eta_q)$, $\mathbb{P} = \texttt{partition}(\mathcal{P}^{K\times b}, \eta_{\mathcal{P}})$
3:     $\hat{\Sigma} = \texttt{Construct-Symbol-Model}(\mathcal{T}, \mathcal{D}, \mathbb{X}, \mathbb{P}, I)$
4:     $\hat{\Gamma}_k, \hat{V}^*_0, \hat{\Sigma} \otimes \mathcal{A}_\varphi = \texttt{Runtime-Select}(\mathcal{T})$
5:     $\hat{V}^*_{\min} = \min_{(q,s)\in\mathbb{X}_0\times S_0} \hat{V}^*_0(q,s)$
6:     $\eta_q = \eta_q/2,\ \eta_{\mathcal{P}} = \eta_{\mathcal{P}}/2,\ I = 2I$
7: end while
8: for $(q,s) \in \mathbb{X}^{\otimes}$, $k \in \{0,\dots,H-1\}$ do
9:     $(q,\mathcal{P}) = \hat{\Gamma}_k(q,s)$
10:    if $\mathcal{NN}_{(q,\mathcal{P})} \notin \mathfrak{NN}_{\text{part}}$ then
11:        $\mathcal{NN}_{(q^*,\mathcal{P}^*)} = \operatorname{argmin}_{\mathcal{NN}_{(q_1,\mathcal{P}_1)}\in\mathfrak{NN}_{\text{part}}} \text{Dist}\left(\mathcal{NN}_{(q_1,\mathcal{P}_1)}, \mathcal{NN}_{(q,\mathcal{P})}\right)$
12:        $\mathcal{NN}_{(q,\mathcal{P})} = \texttt{initialize}(\mathcal{NN}_{(q^*,\mathcal{P}^*)})$
13:        $\mathcal{NN}_{(q,\mathcal{P})} = \texttt{PPO-update}(\mathcal{NN}_{(q,\mathcal{P})}, J)$
14:        $\widehat{W}^{(F)}, \widehat{b}^{(F)} = \Pi_{\mathcal{P}}(\mathcal{NN}_{(q,\mathcal{P})})$
15:        Set $\mathcal{NN}_{(q,\mathcal{P})}$ output layer weights be $\widehat{W}^{(F)}, \widehat{b}^{(F)}$
16:        $\mathfrak{NN}_{\text{part}} = \mathfrak{NN}_{\text{part}} \cup \{\mathcal{NN}_{(q,\mathcal{P})}\}$
17:    end if
18: end for
19: Return $\mathfrak{NN}_{\text{part}}, \{\hat{\Gamma}_k\}_{k\in\{0,\dots,H-1\}}$
```

1.7 Results

We evaluated the proposed framework both in simulation and on a robotic vehicle. All experiments were executed on a single Intel Core i9 2.4-GHz processor with 32 GB of memory. Our open-source implementation of the proposed neurosymbolic framework can be found at `https://github.com/rcpsl/Neurosymbolic_planning`.

1.7.1 Controller Performance in Simulation

Consider a wheeled robot with the state vector $x = [\zeta_x, \zeta_y, \theta]^\top \in X \subset \mathbb{R}^3$, where ζ_x, ζ_y denote the coordinates of the robot and θ is the heading direction. The priori known nominal model f in the form of (4.1) is given by:

$$\begin{aligned} \zeta_x^{(t+\Delta t)} &= \zeta_x^{(t)} + \Delta t \; v \; \cos(\theta^{(t)}) \\ \zeta_y^{(t+\Delta t)} &= \zeta_y^{(t)} + \Delta t \; v \; \sin(\theta^{(t)}) \\ \theta^{(t+\Delta t)} &= \theta^{(t)} + \Delta t \; u^{(t)} \end{aligned} \tag{1.47}$$

where the speed $v = 0.3$m/s and the time step $\Delta t = 1$s. We train NNs to control the robot, i.e., $u^{(t)} = \text{NN}(x^{(t)})$, $\text{NN} \in \mathcal{P}^{K\times b} \subset \mathbb{R}^{1\times 4}$ with the controller space $\mathcal{P}^{K\times b}$ being a hyperrectangle.

As the first step of our framework, we discretized the state space $X \subset \mathbb{R}^3$ and the controller space $\mathcal{P}^{K\times b} \subset \mathbb{R}^{1\times 4}$ as described in Section 1.4.1. Specifically, we partitioned the range of heading direction $\theta \in [0, 2\pi)$ uniformly into 8 intervals, and the partitions in the x, y dimensions are shown as the dashed lines in Figure 1.2. We uniformly partitioned the controller space $\mathcal{P}^{K\times b}$ into 240 hyperrectangles.

Study#1: Comparison against standard NN training for a fixed task. The objective of this study is to compare the proposed framework against standard NN training when the task is known during training time. We aim to show the ability of our framework to guarantee the safety and correctness of achieving the task compared with standard NN training. To that end, we considered the workspace shown in Figure 1.2 and a simple reach-avoid specification, i.e., reach the goal area (green) while avoiding the obstacles (blue).

We collected data by observing the control actions of an expert controller operating in this workspace while varying the initial position of the robot. We trained several NNs using imitation learning for a wide range of NN architectures and a number of episodes to achieve the best performance.

We then trained a library of neural networks $\mathfrak{NN}$ using Algorithm 2, and we used the dataset—used to train NNs with imitation learning—to accelerate the runtime selection as detailed in Algorithm 6 (recall that line 1 in Algorithm 6 uses imitation-learning).

We report the trajectories of the proposed neurosymbolic framework in the first row of Figure 1.2 and the results of the top performing NNs obtained from imitation learning in the second row of Figure 1.2. As shown in the figure, we were able to find initial states from which the imitation-learning-based NNs failed to guarantee the safety of the robot (and hence failed to satisfy the mission goals). However, as shown in the figure (and supported by our theoretical analysis in Theorem 1.7), our framework was capable of always achieving the mission goals and steering the robot safely to the goal.

Study#2: Generalization to unknown workspace/tasks using transfer learning. This experiment aims to study our framework's ability to generalize to unseen tasks even when the library of neural networks is not complete. In other words, the trained local networks in $\mathcal{NN}$ cannot cover all possible transitions in the symbolic model, and hence a transfer learning needs to be performed during the runtime selection phase.

During the offline training, we trained a subset of local networks $\mathfrak{NN}_{\text{part}}$ by following Algo-

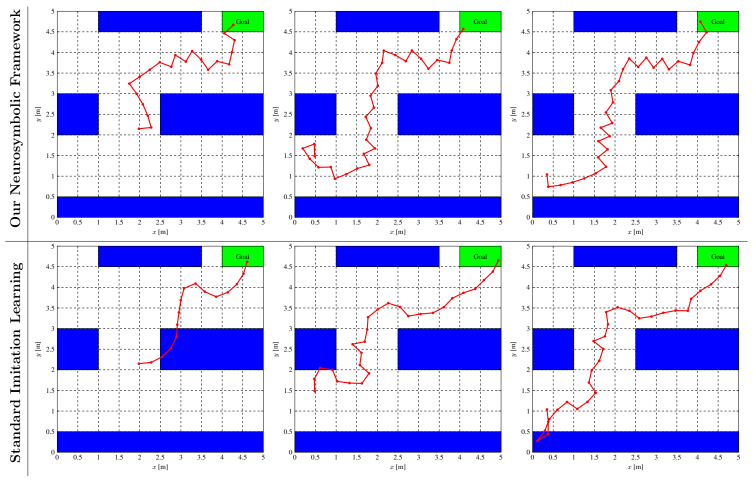


Figure 1.2: The upper row shows trajectories resulting from NN-based planners trained using our framework. The lower row shows trajectories under the control of NNs trained by standard imitation learning, where the NN architectures are (left) 2 hidden layers with 10 neurons per layer, (middle) 2 hidden layers with 64 neurons per layer, and (right) 3 hidden layers with 128 neurons per layer. With the same initial states (two subfigures in the same column), only NN-based planners trained by our framework lead to collision-free trajectories.

rithm 4 in Section 1.6.1. Specifically, the local NNs are trained in the workspace $\mathcal{W}_1$ (the first subfigure in the upper row of Figure 1.3). The set $\mathfrak{NN}_{\text{part}}$ consists of 658 local NNs, where each local NN has only one hidden layer with 6 neurons. We used Proximal Policy Optimization (PPO) implemented in Keras [31] to train each local NN for 800 episodes, and projected the NN weights at the end of training. The total time for training and projecting weights of the 658 local networks in $\mathfrak{NN}_{\text{part}}$ is 2368 seconds.

At runtime, we tested the trained NN-based planner in five unseen workspaces $\mathcal{W}_i$, $i = 2, \ldots, 6$, and the corresponding trajectories are shown in Figure 1.3. For each of the workspaces, our framework computes an activation map Γ that assigns a controller partition $\mathcal{P} \in \mathbb{P}$ to each abstract state $q \in \mathbb{X}$ through dynamical programming (Algorithm 3 in Section 1.4.2). The local NNs corresponding to the assigned controller partitions may not have been trained offline. If this was the case, we follow Algorithm 5 that employs transfer learning to learn the missing NNs at runtime efficiently. Specifically, after initializing a missing NN using its closest NN in the set $\mathfrak{NN}_{\text{part}}$, we trained it for 80 episodes, which is much less than the number of episodes used in the offline training. For example, for the workspace $\mathcal{W}_2$ (the first subfigure in the lower row of Figure 1.3), the length of the corresponding trajectory is 35 steps, and 28 local NNs used along the trajectory are not in the set $\mathfrak{NN}_{\text{part}}$. Our algorithm efficiently trains these 28 local NNs in 10.5 seconds, which shows the capability of our framework in real-time applications.

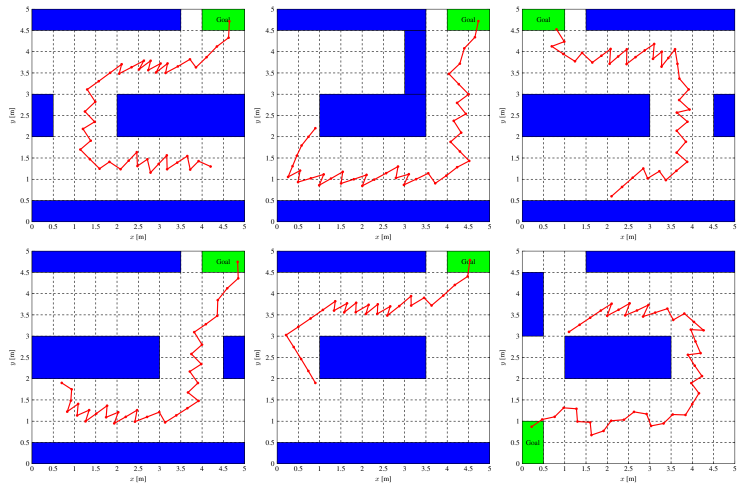


Figure 1.3: The upper row shows trajectories in workspaces $\mathcal{W}_1$, $\mathcal{W}_3$, $\mathcal{W}_5$, and the lower row corresponds to workspaces $\mathcal{W}_2$, $\mathcal{W}_4$, $\mathcal{W}_6$. The subset of local networks $\mathfrak{NN}_{\text{part}}$ is trained in workspace $\mathcal{W}_1$ and the rest five workspaces are given at runtime. Trajectories in all the workspaces satisfy both the safety specification φ_{safety} (blue areas are obstacles) and the liveness specification $\varphi_{\text{liveness}}$ for reaching the goal (green area).

1.7.2 Actual Robotic Vehicle

We tested the proposed framework on a small robotic vehicle called PiCar, which carries a Raspberry Pi that runs the NNs trained by our framework. We used a Vicon motion capture system to measure the states of the PiCar in real-time. Figure 1.4 (left) shows the PiCar and our experimental setup. We modeled the PiCar's dynamics using the rear-wheel bicycle drive [72] and used GP regression to learn the model-error.

Study#3: Dynamic changes in the workspace. We study the ability of our framework to adapt, at runtime, to changes in the workspace. This is critical in cases when the workspace is dynamic and changes over time. To that end, we trained NNs in the workspace shown in Figure 1.4 (right). The part of the obstacle colored in striped blue was considered an obstacle during the training, but was removed at runtime after the PiCar finished running the first loop. Thanks to the DP recursion that selects the optimal NNs at runtime (Algorithm 3 in Section 1.4.2), the PiCar was capable of updating its optimal selection of neural networks and found a better trajectory to achieve the mission.

Study#4: Comparison against meta-RL in terms of generalization to unknown workspace/tasks. The objective of this study is to show the ability of our framework to generalize to unseen tasks, even in scenarios that are known to be hard for state-of-the-art meta-RL algorithms. We conducted our second experiment with the workspaces in Figure 1.5. In particular, the four subfigures in the first row of Figure 1.5 are the workspaces considered for training. These four training workspaces differ in the y-coordinate of the two obstacles (blue areas). During runtime, we use the workspaces shown in the second/third row of Figure 1.5. Specifically, the first subfigure in the second/third rows of Figure 1.5 corresponds to a workspace that has appeared in training. The rest three subfigures in the second/third row of Figure 1.5 are unseen workspaces, i.e., they are not present in training and only become known at runtime. Indeed, as demonstrated in [24], existing meta-RL algorithms are limited by the ability to adapt across homotopy classes (in Figure 1.5, the training tasks and the unseen tasks are in different homotopy classes since trajectories satisfying a training task cannot be continuously deformed to trajectories satisfying an unseen task without intersecting the obstacles).

We show the PiCar's trajectories under the NN-based planner trained by our neurosymbolic framework in the second row of Figure 1.5. By following Algorithm 5 with transfer learning, the PiCar's trajectories satisfy the reach-avoid specifications in all four workspaces, including the three unseen ones. Thanks to the fact that our NN-based planner is composed of local networks, our framework enables easy adaptation across homotopy classes by updating the activation map Γ based on the revealed task (Algorithm 3).

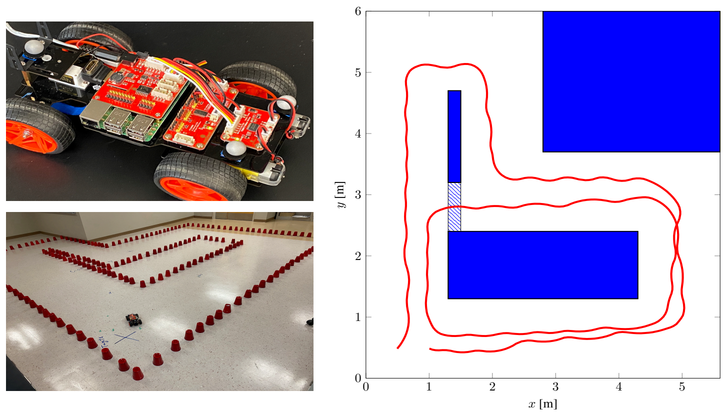


Figure 1.4: (Left) PiCar and workspace. (Right) The PiCar's trajectory (red) for two loops, where the striped blue obstacle is removed after the first loop.

As a comparison, we assessed NN controllers trained by a state-of-the-art meta-RL algorithm PEARL [111] in the above workspaces. Given the four training workspaces (the first row of Figure 1.5), we use PEARL to jointly learn a probabilistic encoder [71] (3 hidden layers with 20 neurons per layer) and a NN controller (3 hidden layers with 30 neurons per layer). The probabilistic encoder accumulates information about tasks into a vector of probabilistic context variables $z \in \mathbb{R}^5$, and the NN controller $\mathcal{NN}$ takes both the robot states x and the context variables z as input and outputs control actions $\mathcal{NN}(x, z)$.

When applying the trained NN controller to a task (either a training task or an unseen task) at runtime, PEARL needs to first update the posterior distribution of the context variables $z \in \mathbb{R}^5$ by collecting trajectories from the corresponding task. The third row of Figure 1.5 shows trajectories under the control of neural networks trained by PEARL. Specifically, the first subfigure in the third row of Figure 1.5 corresponds to a workspace that has appeared in training, and the presented trajectory is obtained after updating the posterior distribution of z with 2 trajectories collected from this workspace. The rest three subfigures in the third row of Figure 1.5 show trajectories in unseen workspaces, where the trajectories cannot be safe even after updating the posterior distribution of z with 100 trajectories collected from the corresponding unseen workspace. By comparing trajectories resulting from our

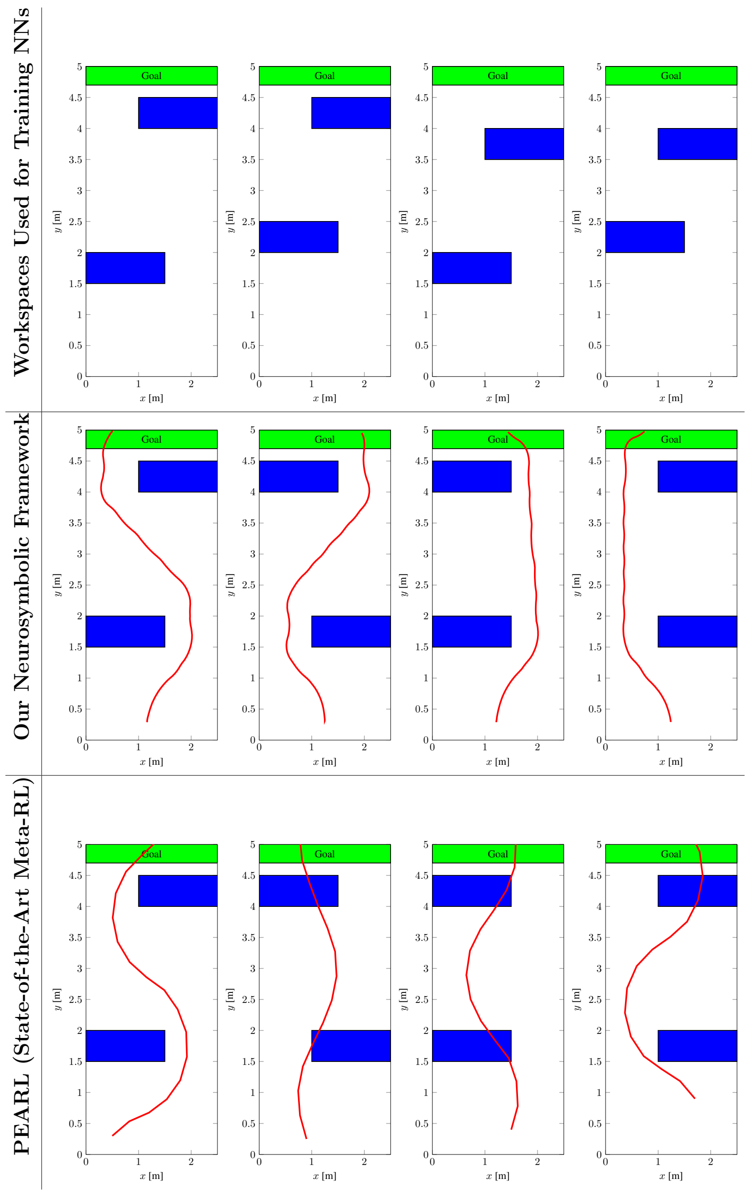


Figure 1.5: Performance comparison between our neurosymbolic framework and a state-of-the-art meta-RL algorithm PEARL. The first row shows the four workspaces used for training NNs. The second row shows the PiCar's trajectories under the NN-based planner trained by our neurosymbolic framework. All the trajectories satisfy reach-avoid specifications even in unseen workspaces. The third row shows trajectories resulting from NN controllers trained by PEARL, where the trajectory is only safe in the training workspace (the first subfigure in the third row) but unsafe in the three unseen workspaces (the rest three subfigures in the third row).

neurosymbolic framework and PEARL (the second and third rows in Figure 1.5), NN-based planners trained by our algorithm show the capability of adapting to unseen tasks that can be very different from training tasks.

1.7.3 Scalability Study

We study the scalability of our framework with respect to both partition granularity and system dimension. In this experiment, we construct the symbolic models $\hat{\Sigma}$ and assign controller partitions by following Algorithm 3. Table 1.1 reports the execution time that grows with the increasing number of abstract states and controller partitions. In Table 1.2, we show the scalability by increasing the system dimension n. To conveniently increase the system dimension, we consider a chain of integrators represented as the linear system $x^{(t+1)} = Ax^{(t)} + Bu^{(t)}$, where $A \in \mathbb{R}^{n \times n}$ is the identity matrix and $u^{(t)} \in \mathbb{R}^2$. Note that our algorithms is not aware of the linearity of the dynamics constraints nor is exploiting this fact. The algorithm has access to a simulator (the function f in (4.1)) that it can use to construct the symbolic model $\hat{\Sigma}$.

To construct the symbolic models $\hat{\Sigma}$ efficiently, we adopt Algorithm 6 and only consider local controller partitions by setting the range parameter I be 25. The execution time show that our algorithm can handle a high-dimensional system in a reasonable amount of time. Although we conducted all the experiments on a single CPU core, we note that our framework is highly parallelizable. For example, both computing transition probabilities in the symbolic model $\hat{\Sigma}$ and training local networks $\mathcal{NN}_{(q,\mathcal{P})}$ can be parallelized.

Table 1.1: Scalability with respect to Partition Granularity

Number of Abstract States	**Number of Controller Partitions**	**Build Symbolic Model $\hat{\Sigma}$ [s]**	**Assign Controller Partitions [s]**
1000	100	10.1	21.8
1000	324	11.3	69.8
1000	900	13.3	193.2
2197	100	41.6	74.2
2197	324	44.7	227.5
2197	900	51.3	673.45
4096	100	145.6	383.8
4096	324	151.2	1210.64
4096	900	164.6	3444.43

Table 1.2: Scalability with respect to System Dimension

System Dimension n	**Number of Abstract States**	**Build Symbolic Model $\hat{\Sigma}$ [s]**	**Assign Controller Partitions [s]**
2	324	2.1	1.8
4	1296	9.4	10.4
6	4096	70.3	62.9
8	16384	311.2.	158.4
10	59049	1581.9	441.7

Chapter 2

NNSynth: Neural Network Guided Abstraction-based Controller Synthesis for Stochastic Systems

In this chapter, we introduce NNSynth, a new framework that uses machine learning techniques to guide the design of abstraction-based controllers with correctness guarantees. NNSynth utilizes neural networks (NNs) to guide the search over the space of controllers. The trained neural networks are "projected" and used for constructing a "local" abstraction of the system. An abstraction-based controller is then synthesized from such "local" abstractions. If a controller that satisfies the specifications is not found, then the best found controller is "lifted" to a neural network for additional training. Our experiments show that this neural network-guided synthesis leads to more than 50$\times$ or even 100$\times$ speedup in high dimensional systems compared to the state-of-the-art.

2.1 Introduction

Abstraction-based control synthesis techniques have gained considerable attention in the past decade. These techniques provide tools for automated, correct-by-construction controller synthesis from complex specifications, typically given in the form of a Linear Temporal Logic (LTL) formulae [147]. It is then unsurprising the vast amount of developed software tools that can handle a wide variety of nonlinear control systems including Pessoa [92], CoSyMa [96], SCOTS [120], QUEST [66], FAUST [136], StocHy [26], and AMYTISS [82]. At the heart of all these tools is the need to obtain discrete abstraction of continuous-time dynamical systems using various quantization methods for state and input spaces. The resulting discrete abstraction is then traversed to search for a feedback controller that conforms to the required LTL specification. While performing the search for the feedback controller over the quantized system is motivated by the availability of tools from the computer science literature that can find such controllers, a significant drawback is the vast number of combinations of quantized states and inputs that needs to be considered. The problem is exacerbated in high-dimensional state and input spaces, leading to the so-called *curse of dimensionality.*

Motivated by the recent success of machine learning techniques in efficiently searching over the space of feedback controllers (e.g., imitation learning and reinforcement learning), we ask the following question: *Can machine learning techniques be used to accelerate the process of synthesizing abstraction-based controllers from LTL specifications?* On the one hand, machine learning techniques enjoy favorable scalability properties and eliminate the dependency on state-space quantization. On the other hand, these learning-based feedback controllers (or policies) do not come with the guarantee that they conform to the LTL specifications. This motivates the need to closely integrate the scalability of learning-based techniques with the provable guarantees provided by the abstraction-based techniques.

Toward this end, we propose NNSynth, a new framework for synthesizing abstraction-based controllers from LTL specifications [144]. Unique to NNSynth is the use of machine learning techniques to train a neural network (NN) based controller, which will guide the synthesis of the final abstraction-based controller. The advantages of the proposed NN guided abstraction-based controller synthesis is multi-fold. First, it utilizes the empirically proven advantages of machine learning algorithms to search the space of feedback controllers without relying on expensive quantizations of state and input spaces. Second, it limits the search over the quantized spaces only to local control actions within the neighborhood of the controller proposed by the NN training. That is, our approach uses NN training to guide the search over the quantized abstract system and eliminates the need to consider all combinations of quantized states and inputs. Third, the use of neural networks to guide the design of the abstraction-based controller opens the door to encode the human's preferences for how a dynamical system should act. Such human's preference is crucial for several real-world settings in which a human user or operator interacts with an autonomous dynamical system [32]. Current research found that human preferences can be efficiently captured using expert demonstrations and preference-based learning which can be hard to be accurately capture in the form of a logical formulae or a reward function [99]. These advantages are demonstrated using several key applications showing that NNSynth scales more favorably compared to the state-of-the-art techniques while achieving more than 50$\times$ or even 100$\times$ speedup in high dimensional systems.

Related Work. The closest results to our work are those reported in [5, 156] which proposes a neurosymbolic framework to train control policies that can be represented as short programs in a symbolic language while ensuring the generated policies are safe. Similar to our approach, the work in [5, 156] trains a NN controller, project it to the space of symbolic controllers, analyze the symbolic controller and lift it back to the space of NN policies for further training. Differently, our approach focuses on designing a finite-state, abstraction-based controller instead of short programs in a symbolic language. This difference (short programs versus finite-state controller) manifests itself in all the framework steps, particularly the NN training, projection, and lifting. We confine our focus on synthesizing finite-state controllers due to the extensive literature on analyzing such controllers in tandem with the controlled physical systems [147]. Another line of related work is reported in [161, 25] which studies the problem of extracting a finite-state controller from a recurrent neural network controller. We note that our framework uses the NN policy to guide the search for abstraction-based controllers and not as the final produced controller.

2.2 Problem Formulation

Let $\mathbb{R}$, $\mathbb{R}^+$, $\mathbb{N}$ be the set of real numbers, positive real numbers, and natural numbers, respectively. For a non-empty set S, let $|S|$ be the cardinality of S, 2^S be the power set of S, $\mathbf{1}_S$ be the indicator function of S, and $\text{Int}(S)$ be the interior of S. Furthermore, we use S^n to denote the set of all finite sequences of length $n \in \mathbb{N}$ of elements in S. The product of two sets is defined as $S_1 \times S_2 = \{(s_1, s_2) | s_1 \in S_1, s_2 \in S_2\}$. Given a natural number $H \in \mathbb{N}$, let $\bar{H} = \{0, 1, \ldots, H\}$. Let $\|x\|$ be the Euclidean norm of a vector $x \in \mathbb{R}^n$ and $x^\top$ be the transpose of $x \in \mathbb{R}^n$. Let the inner product of two functions $h_1 : X \to \mathbb{R}^m$ and $h_2 : X \to \mathbb{R}^m$ be defined as $\langle h_1, h_2 \rangle = \int_X h_1(x)^\top h_2(x) dx$, which induces a norm $\|h_1\| = \sqrt{\langle h_1, h_1 \rangle}$. We use ∇J to denote the Fréchet gradient of a functional J, and use the big O notation for upper bounds. Any Borel space X is assumed to be endowed with a Borel σ-algebra denoted by $\mathcal{B}(X)$.

2.2.1 Dynamical Model

We consider discrete-time nonlinear stochastic systems with continuous state and action spaces:

$$x^{(t+1)} = f(x^{(t)}, u^{(t)}) + \zeta^{(t)}, \tag{2.1}$$

where $x^{(t)} \in X \subset \mathbb{R}^n$ is the state and $u^{(t)} \in U \subset \mathbb{R}^m$ is the control action at time step $t \in \mathbb{N}$, respectively. The dynamical model (4.1) consists of a nominal model f and an addictive noise $\zeta^{(t)}$. We consider the nominal model f can be evaluated using a simulator (we do not require the function f in a closed-form/symbolic representation), and the noise $\zeta^{(t)}$ is sampled from a given distribution.

The dynamical model (4.1) can be equivalently expressed using a stochastic kernel $\tau : \mathcal{B}(X) \times X \times U \to [0, 1]$ that assigns to any $x \in X$ and $u \in U$ a probability measure $\tau(\cdot|x, u)$ such that for any subset $A \in \mathcal{B}(X)$:

$$\Pr(x^{(t+1)} \in A | x^{(t)}, u^{(t)}) = \int_A \tau(dx^{(t+1)} | x^{(t)}, u^{(t)}). \tag{2.2}$$

The stochastic kernel τ captures the evolution of system (4.1) and can be uniquely determined by the nominal model f and the noise $\zeta^{(t)}$ in (4.1).

Example 1 Consider a nonlinear dynamical system of the form:

$$x^{(t+1)} = f(x^{(t)}, u^{(t)}) + g(x^{(t)}, u^{(t)}), \tag{2.3}$$

where f is a priori known nominal model and g captures the unknown model-error. As a well-studied technique to learn unknown functions from data, we assume the model-error g can be learned by a Gaussian Process (GP) regression model $\mathcal{GP}(\mu_g, \sigma_g^2)$, where μ_g and

σ_g^2 are the posterior mean and variance functions, respectively [113]. Then, the nonlinear system (2.3) with the model-error g learned by $\mathcal{GP}(\mu_g, \sigma_g^2)$ can be treated as a nonlinear stochastic system in the form of (4.1), where the noise $\zeta^{(t)}$ is sampled from the normal distribution $\mathcal{N}(\mu_g(x^{(t)}, u^{(t)}), \sigma_g^2(x^{(t)}, u^{(t)}))$. Furthermore, the stochastic kernel $\tau(\cdot|x^{(t)}, u^{(t)})$ is given by the normal distribution $\mathcal{N}(f(x^{(t)}, u^{(t)}) + \mu_g(x^{(t)}, u^{(t)}), \sigma_g^2(x, u))$, and hence the integral (2.2) can be easily computed as an integral of normal distribution.

2.2.2 Temporal Logic Specification

We consider bounded linear temporal logic (BLTL) [21] and syntactically co-safe linear temporal logic (scLTL) [78] specifications, which have been extensively demonstrated the capability to capture complex behaviors of dynamical systems. Let AP be a finite set of atomic propositions that describe the states of a dynamical system with respect to the environment. Given AP, any BLTL formula can be generated according to the following grammar:

$$\varphi := \sigma \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \, \mathcal{U}_{[t_1,t_2]} \, \varphi_2$$

where $\sigma \in AP$ and time steps $t_1 < t_2$. Given the above grammar, we can define $\varphi_1 \wedge \varphi_2 = \neg(\neg\varphi_1 \vee \neg\varphi_2)$, $false = \varphi \wedge \neg\varphi$, and $true = \neg false$. Furthermore, the bounded-time *eventually* operator can be derived as $\Diamond_{[t_1,t_2]}\varphi = true \, \mathcal{U}_{[t_1,t_2]} \, \varphi$ and the bounded-time *always* operator is given by $\Box_{[t_1,t_2]}\varphi = \neg\Diamond_{[t_1,t_2]}\neg\varphi$.

Given a set of atomic propositions AP, the corresponding alphabet is defined as $\mathbb{A} := 2^{AP}$, and a finite (infinite) word ω is a finite (infinite) sequence of letters from the alphabet $\mathbb{A}$, i.e., $\omega = \omega^{(0)}\omega^{(1)} \ldots \omega^{(H)} \in \mathbb{A}^{H+1}$. The satisfaction of a word ω to a specification φ can be determined based on the semantics of BLTL [21]. Given the dynamical system (4.1) and an alphabet $\mathbb{A}$, let $L : X \to \mathbb{A}$ be a labeling function that assigns to each state $x \in X$ the subset of atomic propositions $L(x) \in \mathbb{A}$ that evaluate *true* at x. Then, a system's trajectory

ξ satisfies a specification φ, denoted by $\xi \models \varphi$, if the corresponding word satisfies φ, i.e., $L(\xi) \models \varphi$, where $\xi = x^{(0)}x^{(1)} \dots x^{(H)} \in X^{H+1}$ and $L(\xi) = L(x^{(0)})L(x^{(1)}) \dots L(x^{(H)}) \in \mathbb{A}^{H+1}$. Similarly, we can consider scLTL specifications interpreted over infinite words based on the fact that any infinite word that satisfies a scLTL formula φ contains a finite "good" prefix such that all infinite words that contain the prefix satisfy φ [78].

Every BLTL or scLTL formula φ defined over an alphabet $\mathbb{A}$ can be translated to an equivalent deterministic finite-state automaton (DFA) $\mathcal{A}_\varphi = (S, S_0, \mathbb{A}, G, \rho)$, where:

- S is a finite set of states;
- $S_0 \subseteq S$ is the set of initial states;
- $\mathbb{A} := 2^{AP}$ is the alphabet;
- $G \subseteq S$ is the accepting set;
- $\rho : S \times \mathbb{A} \to S$ is a transition function.

Such translation of BLTL and scLTL specifications to the equivalent DFA can be done using off-the-shelf tools (e.g., [81, 56]).

Example 2 *(Reach-avoid Specification)*: Consider an agent that navigates an environment characterized by a goal $X_{\text{goal}} \subset X$ that the agent would like to reach and a set of obstacles $O_1, \dots, O_c \subset X$ that the agent needs to avoid. The set of atomic propositions is given by $AP = \{x \in X_{\text{goal}}, x \in O_1, \dots, x \in O_c\}$, where x is the state of the agent. Then, a reach-avoid specification can be expressed as $\varphi = \varphi_{\text{liveness}} \wedge \varphi_{\text{safety}}$, where $\varphi_{\text{liveness}} = \Diamond_{[0,H]}(x \in X_{\text{goal}})$ requires the agent to reach the goal X_{goal} in H time steps and $\varphi_{\text{safety}} = \Box_{[0,H]} \bigwedge_{i=1,\dots,c} \neg(x \in O_i)$ specifies to avoid all the obstacles during the time horizon H. Let $\xi = x^{(0)}x^{(1)} \dots x^{(H)}$

be a trajectory of the agent, then the reach-avoid specification φ is interpreted as:

$$\xi \models \varphi_{\text{liveness}} \Longleftrightarrow \exists t \in \{0, \dots H\}, x^{(t)} \in X_{\text{goal}},$$

$$\xi \models \varphi_{\text{safety}} \Longleftrightarrow \forall t \in \{0, \dots H\}, \forall i \in \{1, \dots, c\}, x^{(t)} \notin O_i.$$

2.2.3 Main Problem

The goal of this chapter is to synthesize a controller $\Psi : X \times S \times \bar{H} \to U$ for the dynamical system (4.1) to satisfy a given specification φ while minimizing some cost functional J. To take into account the BLTL or scLTL formula φ, the controller Ψ takes as input the system's states $x \in X$, states $s \in S$ of the DFA $\mathcal{A}_\varphi$, and times steps $t \in \bar{H}$. The cost functional J is defined as:

$$J(\Psi) = \sum_{t \in \bar{H}} \int_{X \times S} c(x^{(t)}, \Psi(x^{(t)}, s^{(t)}, t)) d\mu^{\Psi}(x^{(t)}, s^{(t)}),$$

where $c : X \times U \to \mathbb{R}$ is a state-action cost function and μ^{Ψ} is the distribution of the system's states and the DFA's states induced by the controller Ψ. Let $X_0 \subseteq X$ be the set of initial states of the dynamical system (4.1) and ξ_{Ψ}^{x} denote a closed-loop trajectory of the system that starts from the state $x \in X_0$ and evolves under the controller Ψ. We define the problem of interest as follows:

Problem 2.1. *Given a nonlinear stochastic system* (4.1)*, a high-level specification* φ*, and a cost functional* J*, synthesize a controller* $\Psi : X \times S \times \bar{H} \to U$ *that minimizes the cost* $J(\Psi)$ *and satisfies the specification* φ *with a pre-specified probability threshold* p*, i.e.,* $\Pr(\xi_{\Psi}^{x} \models \varphi) \geq p$*,* $\forall x \in X_0$*.*

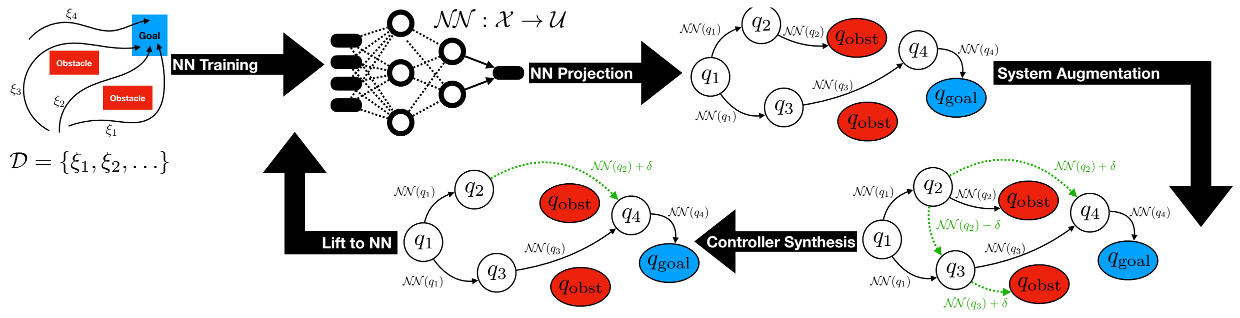

ξ_4
Goal
Obstacle
Obstacle
ξ_3
ξ_2
ξ_1
$\mathcal{D} = \{\xi_1, \xi_2, \dots\}$
NN Training
$\mathcal{NN} : \mathcal{X} \to \mathcal{U}$
NN Projection
System Augmentation
Controller Synthesis
Lift to NN
q_{obst}
q_{goal}
$\mathcal{NN}(q_2) + \delta$
$\mathcal{NN}(q_2) - \delta$
$\mathcal{NN}(q_3) + \delta$

Figure 2.1: A cartoon summarizing the NNSynth framework. NNSynth starts by training a neural network controller $\mathcal{NN}$ using the dataset $\mathcal{D}$ provided by an expert. The obtained neural network is then projected to a symbolic model by evaluating the neural network at the representative points of abstract states, i.e. using the control actions $\mathcal{NN}(\text{ct}(q), s, t)$. The obtained symbolic model is then augmented with control actions in the neighborhood of the actions proposed by the neural network $\mathcal{NN}(\text{ct}(x), s, t) \pm i\delta$. A controller is then synthesized from the augmented symbolic model. In case that a controller was not found, the "best" controller so far is then lifted to a neural network which is further trained using the expert dataset $\mathcal{D}$ to obtain a new $\mathcal{NN}$. The loop continues until a controller with correctness guarantees is found.

2.3 NNSynth Framework

Our framework is featured by the use of neural networks to guide the synthesis of controllers in symbolic representation. Controller synthesis using symbolic techniques enjoy the guarantees of satisfying temporal logic specifications φ. However, these symbolic techniques suffer from computational complexity whenever the dynamical models are highly nonlinear and complex. Moreover, controllers in symbolic representation are hard to be optimized in terms of minimizing the cost functional J. To that end, NNSynth incorporates neural networks in the synthesis loop of symbolic controllers. The benefit of using NNs are two-fold: (i) we use NNs to limit the search space of symbolic controllers and hence improve the computational efficiency; (ii) we use the gradient of NNs to optimize the performance of symbolic controllers (i.e., minimizing the cost functional J) by "projecting" and "lifting" between NNs and symbolic controllers. In this way, even though the gradient of a symbolic controller does not exist in general, we can improve NN controllers using gradient-based approaches and "project" the improvement to symbolic controllers.

Algorithm 8 NNSYNTH ($\mathcal{D}_{\text{exp}}$, φ, p, ε, η)

```
 1: Translate $\varphi$ to a DFA $\mathcal{A}_\varphi = (S, S_0, \mathbb{A}, G, \rho)$
 2: Initialize $\mathcal{NN}_{\text{init}}$ with random weights
 3: $\mathcal{NN}_0 = $ UPDATE$(\mathcal{NN}_{\text{init}}, \mathcal{D}_{\text{exp}}, \eta)$
 4: $\Psi_0, V_{\min} = $ PROJECT-BY-SYNTH$(\mathcal{NN}_0, \mathcal{A}_\varphi)$
 5: for $k = 0, \ldots, K-1$ do
 6:     if $V_{\min} \geq p + \varepsilon$ then
 7:         Return $\Psi_k, V_{\min}$
 8:     end if
 9:     $\mathcal{NN}_k = $ LIFT$(\Psi_k)$
10:     $\mathcal{NN}_{k+1} = $ UPDATE$(\mathcal{NN}_k, \mathcal{D}_{\text{exp}}, \eta)$
11:     $\Psi_{k+1}, V_{\min} = $ PROJECT-BY-SYNTH$(\mathcal{NN}_{k+1}, \mathcal{A}_\varphi)$
12: end for
13: Return $\Psi_K, V_{\min}$
```

We first give an overview of our framework, and then present each step separately in the following subsections. The overview of the proposed NNSynth is depicted in Figure 2.1. Algorithm 8 outlines the framework. After translating the given specification φ to the equivalent DFA $\mathcal{A}_\varphi$, NNSynth trains a neural network $\mathcal{NN}_0$ using gradient-based approaches such as imitation learning of the expert dataset $\mathcal{D}_{\text{exp}}$ with learning rate η (line 3 in Algorithm 8). The trained neural network is then projected to a symbolic controller Ψ_0 through the procedure PROJECT-BY-SYNTH presented in Algorithm 9 (line 4 in Algorithm 8). If the resulting symbolic controller Ψ_k does not satisfy the specification φ with probability at least $p + \varepsilon$ (line 6 in Algorithm 8), the controller Ψ_k is lifted to a neural network $\mathcal{NN}_k$ for further training (line 9-10 in Algorithm 8). This synthesis loop iterates until a symbolic controller with the desired correctness guarantee is found.

2.3.1 Step 1: NN Training

The first step of NNSynth is to train a NN controller $\mathcal{NN}\colon X \times S \times \bar{H} \to U$ that minimizes the cost functional J. The NN controller can be trained using either imitation learning or reinforcement learning. For imitation learning, the training dataset is given by a set of expert-provided trajectories $\mathcal{D}_{\text{exp}} = \{(x_j^{(t)}, s_j^{(t)}, u_j^{(t)})\}$, where the trajectory index $j = 1, 2, \ldots, M$

and the time step $t = 0, 1, \ldots, H$. Alternatively, the NN controller can be trained by reinforcement learning, which requires the expert to provide the state-action cost function $c : X \times U \rightarrow \mathbb{R}$ instead of the dataset $\mathcal{D}_{\text{exp}}$. Neural networks are highly parameterized and can be updated using gradient-based approaches $\mathcal{NN}_{k+1} = \mathcal{NN}_k - \eta \nabla J(\mathcal{NN}_k)$, where $\eta \in \mathbb{R}^+$ is the learning rate. Let $\mathcal{NN}_k^\theta$ denote a neural network parameterized by weights θ, then the gradient $\nabla J(\mathcal{NN}_k^\theta)$ can be approximated using sampled trajectories:

$$\nabla J(\mathcal{NN}_k^\theta) \approx \frac{1}{M} \sum_{i=1}^{M} \sum_{t=0}^{H} \nabla_\theta \mathcal{NN}_k^\theta(u_i^{(t)} | x_i^{(t)}, s_i^{(t)}, t) \widehat{Q}_i^{(t)} \tag{2.4}$$

where M is the number of trajectories, H is the bounded time horizon, and $\widehat{Q}_i^{(t)}$ is the estimated cost-to-go. Detailed optimality analysis of the gradient-based update is given in Section IV.

2.3.2 Step 2: NN Projection

Regardless of the use of imitation learning or reinforcement learning, the resulting neural network $\mathcal{NN}$ is not guaranteed to satisfy the specification φ and hence can not be used directly as a controller. Nevertheless, the neural network contains relevant control actions that can be used to obtain the final controller. To that end, NNSynth projects the trained neural network $\mathcal{NN}$ to a symbolic model and synthesizes a symbolic controller Ψ based on the symbolic model.

To construct the symbolic model, we first partition the continuous state space $X \subset \mathbb{R}^n$ into a finite set of abstract states $\widehat{X} = \{q_1, \ldots, q_N\}$, where each abstract state $q_i \in \widehat{X}$ is an infinity-norm ball in $\mathbb{R}^n$ with a pre-specified diameter $\lambda \in \mathbb{R}^+$ (see Section 2.4 for the choice of λ). The partitioning satisfies $X = \bigcup_{q \in \widehat{X}} q$ and $\text{Int}(q_i) \cap \text{Int}(q_j) = \emptyset$ if $i \neq j$. Let $\text{abs} : X \rightarrow \widehat{X}$ map a state $x \in X$ to the abstract state $\text{abs}(x) \in \widehat{X}$ that contains x, i.e.,

$x \in \text{abs}(x)$, and $\text{ct} : \widehat{X} \to X$ map an abstract state $q \in \widehat{X}$ to its center $\text{ct}(q) \in X$, which is well-defined since abstract states are inifinity-norm balls. With some abuse of notation, we denote by q both an abstract state, i.e., $q \in \widehat{X}$, and a subset of states, i.e., $q \subseteq X$.

Given the dynamical model (4.1), the DFA $\mathcal{A}_\varphi = (S, S_0, \mathbb{A}, G, \rho)$ of the specification φ, and a state space partitioning $\widehat{X}$, we project the trained NN controller $\mathcal{NN}$ to the symbolic model $\Sigma_\varphi^{\mathcal{NN}} = (X^\otimes, X_0^\otimes, \widehat{U}^{\mathcal{NN}}, X_G^\otimes, T^{\mathcal{NN}})$ as follows:

- $X^\otimes = \widehat{X} \times S$ is a finite set of states;
- $X_0^\otimes = \{(q_0, \delta(s_0, \hat{L}(q_0)) \mid q_0 \in \widehat{X}_0, s_0 \in S_0\}$ is the set of initial states, where $\widehat{X}_0 = \{q \in \widehat{X} \mid q \subseteq X_0\}$ and $\hat{L} : \widehat{X} \to \mathbb{A}$ is the labeling function that assigns to each abstract state $q \in \widehat{X}$ the subset of atomic propositions $\hat{L}(q) \in \mathbb{A}$ that evaluate *true* at q;
- $\widehat{U}^{\mathcal{NN}} = \{\mathcal{NN}(\text{ct}(q), s, t) \mid q \in \widehat{X}, s \in S, t \in \bar{H}\}$ is a finite set of control actions by evaluating $\mathcal{NN}$ at the center of each abstract state;
- $X_G^\otimes = \widehat{X} \times G$ is the accepting set;
- The transition probability from state $(q, s) \in X^\otimes$ to state $(q', s') \in X^\otimes$ under action $u \in U$ is given by:

$$T^{\mathcal{NN}}(q', s' | q, s, u) = \begin{cases} \Pr(q' | \text{ct}(q), u) & \text{if } s' = \rho(s, \hat{L}(q')) \text{ and} \\ & u \in \{\mathcal{NN}(\text{ct}(q), s, t) \mid t \in \bar{H}\} \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

In (2.5), the transition probability $\Pr(q' | \text{ct}(q), u)$ can be computed as the integral (2.2). The symbolic model $\Sigma_\varphi^{\mathcal{NN}}$ considers only control actions taken by the trained NN, i.e, $u \in \widehat{U}^{\mathcal{NN}}$. Computing such symbolic model $\Sigma_\varphi^{\mathcal{NN}}$ is straightforward and entails evaluating the NN controller at the center of each each abstract state and computing the transition probabilities associated with these actions.

2.3.3 Step 3: System Augmentation

The symbolic model $\Sigma_\varphi^{\mathcal{NN}}$ may contain transitions that violate the given specification φ since the trained neural network $\mathcal{NN}$ lacks correctness guarantees. Therefore, the next step is to "augment" $\Sigma_\varphi^{\mathcal{NN}}$ with additional transitions corresponding to control actions that are close to those taken by $\mathcal{NN}$. This augmentation will provide the controller synthesis algorithm with the freedom to choose control actions not contained in the set $\widehat{U}^{\mathcal{NN}}$. Given a precision $\delta \in \mathbb{R}^+$ and a range parameter $I \in \mathbb{N}$ (see the choice of δ and I in Section 2.4), we construct the augmented symbolic model $\Sigma_\varphi^{\mathcal{NN}+\delta} = (X^\otimes, X_0^\otimes, \widehat{U}^{\mathcal{NN}+\delta}, X_G^\otimes, T^{\mathcal{NN}+\delta})$, where $X^\otimes$, $X_0^\otimes$, and $X_G^\otimes$ are the same as those in $\Sigma_\varphi^{\mathcal{NN}}$, with a finite set of actions $\widehat{U}^{\mathcal{NN}+\delta}$ and transition probabilities $T^{\mathcal{NN}+\delta}$ as follows:

$$\widehat{U}^{\mathcal{NN}+\delta} = \{\mathcal{NN}(\mathrm{ct}(q), s, t) \pm i\delta \mid q \in \widehat{X}, s \in S, t \in \bar{H}, i \in \bar{I}\} \tag{2.6}$$

$$T^{\mathcal{NN}+\delta}(q', s'|q, s, u) = \begin{cases} \Pr(q'|\mathrm{ct}(q), u) & \text{if } s' = \rho(s, \hat{L}(q')) \text{ and} \\ & u \in \{\mathcal{NN}(\mathrm{ct}(q), s, t) \pm i\delta \mid t \in \bar{H}, i \in \bar{I}\} \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

where with some abuse of notation, we use $\mathcal{NN}(\mathrm{ct}(q), s, t) \pm i\delta$ to denote $\mathcal{NN}(\mathrm{ct}(q), s, t) + [\pm i_1\delta, \pm i_2\delta, \ldots, \pm i_m\delta]^\top$ with $i_1, i_2, \ldots i_m \in \{0, 1, \ldots, I\}$. In other words, the augmented symbolic model $\Sigma_\varphi^{\mathcal{NN}+\delta}$ takes into account all the control actions that are $\delta, 2\delta, \ldots I\delta$ away from those given by the neural network $\mathcal{NN}$, where the distance is considered for each dimension of the control input $u \in \mathbb{R}^m$.

2.3.4 Step 4: Controller Synthesis

The next step is to synthesize a controller $\Psi : X \times S \times \bar{H} \to U$ using the augmented symbolic model $\Sigma_{\varphi}^{\mathcal{NN}+\delta}$. Specifically, we first synthesize a controller $\widehat{\Psi} : \widehat{X} \times S \times \bar{H} \to U$ that maximizes the probability of reaching the accepting set $X_G^{\otimes}$ in $\Sigma_{\varphi}^{\mathcal{NN}+\delta}$. Then, the controller $\Psi : X \times S \times \bar{H} \to U$ can be obtained by letting $\Psi(x, s, t) = \widehat{\Psi}(\text{abs}(x), s, t)$, i.e., applying the same control action $\widehat{\Psi}(q, s, t)$ at all states $x \in q$, where $q \in \widehat{X}$. In Section 2.4, we will show that such a controller Ψ maximizes the probability of satisfying the given specification φ for the dynamical system (4.1) with continuous state and action spaces.

We use dynamic programming (DP) to synthesize the controller $\widehat{\Psi} : \widehat{X} \times S \times \bar{H} \to U$ that maximizes the probability of reaching the accepting set $X_G^{\otimes}$ in the augmented symbolic model $\Sigma_{\varphi}^{\mathcal{NN}+\delta}$. To that end, we define the optimal value functions $V_t^* : X^{\otimes} \to [0, 1]$ that map a state $(q, s) \in X^{\otimes}$ to the maximum probability of reaching the accepting set $X_G^{\otimes}$ in $H - t$ steps from the state (q, s). When $t = 0$, the optimal value function V_0^* yields the maximum probability of reaching the accepting set $X_G^{\otimes}$ in H steps, i.e., the maximum probability of the dynamical system (4.1) satisfying φ. The optimal value functions can be solved by the following dynamic programming recursion:

$$Q_t(q, s, u) = \mathbf{1}_G(s) + \mathbf{1}_{S \setminus G}(s) \sum_{(q', s') \in X^{\otimes}} V_{t+1}^*(q', s') T^{\mathcal{NN}+\delta}(q', s' | q, s, u) \tag{2.8}$$

$$V_t^*(q, s) = \max_{u \in \{\mathcal{NN}(\text{ct}(q), s, t) \pm i\delta | i \in \bar{I}\}} Q_t(q, s, u) \tag{2.9}$$

with the initial condition $V_H^*(q, s) = \mathbf{1}_G(s)$ for all $(q, s) \in X^{\otimes}$, where the transition probability matrix $T^{\mathcal{NN}+\delta}$ is given by (2.7) and $t = H - 1, \ldots, 0$.

Critical to the speedup of NNSynth is that the entries $T^{\mathcal{NN}+\delta}(q', s' | q, s, u)$ are nonzero only when $u \in \hat{U}^{\mathcal{NN}+\delta}$, i.e., the control actions are close to that suggested by the neural network. This avoids computing all the transition probabilities from any state $(q, s) \in X^{\otimes}$ to any state $(q', s') \in X^{\otimes}$ under any control action $u \in U$, which is usually the computational bottlenecks for abstraction-based controller synthesis. Further speedup is achieved by limiting the search of the optimal action to the neighborhood of the actions suggested by $\mathcal{NN}$, i.e., the maximization over $\{\mathcal{NN}(\mathrm{ct}(q), s, t) \pm i\delta \mid i \in \bar{I}\}$ in (2.9).

Algorithm 9 presents details on using the NN controller $\mathcal{NN}$ to guide the synthesis of the symbolic controller Ψ by summarizing Subsections 2.3.2, 2.3.3, and 2.3.4. Given a neural network $\mathcal{NN}$ and a DFA $\mathcal{A}_\varphi$, NNSynth first projects $\mathcal{NN}$ to the augmented symbolic model $\Sigma_\varphi^{\mathcal{NN}+\delta}$ (line 6-11 of Algorithm 9), where the transition probability matrix $T^{\mathcal{NN}+\delta}$ is defined as (2.7). In particular, the entries $T^{\mathcal{NN}+\delta}(q', s'|q, s, u)$ are computed only if the control action u is close to that given by $\mathcal{NN}$ and u has not been considered before at $(q, s) \in X^{\otimes}$, i.e., $u \notin U_{\mathrm{buffer}}(q, s)$ (line 7 of Algorithm 9). The optimal control action at each state $(q, s) \in X^{\otimes}$ is determined as the maximizer of the Q-function (line 12-21 in Algorithm 9). Unique to NNSynth, the optimal action is searched over the local action space $\{\mathcal{NN}(\mathrm{ct}(q), s, t) \pm i\delta \mid i \in \bar{I}\}$ at each state $(q, s) \in X^{\otimes}$. In line 9 of Algorithm 9, $B_r(f(\mathrm{ct}(q), u))$ denotes the subset of abstract states that are in a ball centered at $f(\mathrm{ct}(q), u)$ with radius r, where r is a user-defined probability cut-off (i.e., when probability is smaller than the cut-off, the probability is treated as zero), which allows further speedup by discarding transitions with small enough probabilities [82]. The resulting controller $\Psi : X \times S \times \bar{H} \to U$ applies the same control action $\widehat{\Psi}(q, s, t)$ at all states $x \in q$ (line 20 of Algorithm 9).

2.3.5 Step 5: Lift to NN

To further minimize the cost $J(\Psi_k)$, NNSynth "lifts" the symbolic controller Ψ_k obtained in the above step to a neural network $\mathcal{NN}_k$, which allows us to employ the well-developed deep policy gradient approaches to update the controller. Such lifting can be done by imitation learning with sampled trajectories of the dynamical system (4.1) controlled by

Algorithm 9 PROJECT-BY-SYNTH ($\mathcal{NN}$, $\mathcal{A}_\varphi$)

```
 1: for $(q, s) \in \mathbb{X}^\otimes$ do
 2:     $V_H^*(q, s) = \mathbf{1}_G(s)$
 3: end for
 4: $U_{\text{buffer}}(q, s) = \text{set}()$ for all $(q, s) \in X^\otimes$
 5: for $t = H-1, \dots, 0$ do
 6:     for $(q, s) \in X^\otimes$ do
 7:         for $u \in \{\mathcal{NN}(\text{ct}(q), s, t) \pm i\delta | i \in \bar{I}\} \setminus U_{\text{buffer}}(q, s)$ do
 8:             $U_{\text{buffer}}(q, s) = U_{\text{buffer}}(q, s) \cup \{u\}$
 9:             Compute $T^{\mathcal{NN}+\delta}(q', s'|q, s, u), \forall q' \in B_r(f(\text{ct}(q), u))$
10:         end for
11:     end for
12:     for $(q, s) \in \mathbb{X}^\otimes$ do
13:         if $s \in G$ then
14:             $Q_t(q, s, \mathcal{P}) = 1$
15:         else
16:             $Q_t(q, s, \mathcal{P}) = \sum_{(q', s') \in X^\otimes} V_{t+1}^*(q', s') T^{\mathcal{NN}+\delta}(q', s'|q, s, u)$
17:         end if
18:         $V_t^*(q, s) = \max_{u \in \{\mathcal{NN}(\text{ct}(q), s, t) \pm i\delta | i \in \bar{I}\}} Q_t(q, s, u)$
19:         $\widehat{\Psi}(q, s, t) = \operatorname{argmax}_{u \in \{\mathcal{NN}(\text{ct}(q), s, t) \pm i\delta | i \in \bar{I}\}} Q_t(q, s, u)$
20:         $\Psi(x, s, t) = \widehat{\Psi}(q, s, t)$ for all $x \in q$
21:     end for
22: end for
23: $V_{\min} = \min_{(q,s) \in X_0^\otimes} V_0^*(q, s)$
24: Return $\Psi, V_{\min}$
```

Ψ_k. The obtained neural network is then used as an initialization for further training by either reinforcement learning or imitation learning of the expert dataset $\mathcal{D}_{\text{exp}}$. In Section 2.4, we analyze the performance of the synthesized controllers by taking into account the error due to the lift.

2.4 Theoretical Analysis

2.4.1 Correctness Guarantee on Specification Satisfaction

We provide theoretical guarantees of NNSynth on both satisfying the given specification φ and minimizing the cost functional J in this section. The satisfaction of φ with the pre-specified probability is correct-by-construction. In particular, the procedure PROJECT-BY-SYNTH (Algorithm 9) maximizes the probability of reaching the accepting set $X_G^{\otimes}$ in the augmented symbolic model $\Sigma_{\varphi}^{\mathcal{NN}+\delta}$. We show that the resulting controller maximizes the probability of satisfying the given specification φ for the dynamical system (4.1), and then bound the probability difference between the symbolic model $\Sigma_{\varphi}^{\mathcal{NN}+\delta}$ and the dynamical system (4.1). Let H be the bounded time horizon in φ, $\mathcal{A}$ be the Lebesgue measure of the state space X, L_τ be the Lipschitz constant of the stochastic kernel τ (in (2.2)), and λ be the grid size used for partitioning the state space X when constructing the symbolic model $\Sigma_{\varphi}^{\mathcal{NN}}$. Then, the correctness guarantee on satisfying φ is the following:

Theorem 2.2. *Consider Algorithm 8 returns a symbolic controller Ψ_k with a probability $V_{min} \geq p + \varepsilon$, where $\varepsilon = \lambda H \mathcal{A} L_\tau$. Then, the dynamical system* (4.1) *controlled by Ψ_k is guaranteed to satisfy the given specification φ with probability at least p, i.e.,* $\Pr\left(\xi_{\Psi_k}^{x} \models \varphi\right) \geq p$*, $\forall x \in X_0$.*

Proof. By the construction, the augmented symbolic model $\Sigma_{\varphi}^{\mathcal{NN}+\delta} = (X^{\otimes}, X_0^{\otimes}, \widehat{U}^{\mathcal{NN}+\delta}, X_G^{\otimes}, T^{\mathcal{NN}+\delta})$ is the product of the DFA $\mathcal{A}_\varphi = (S, S_0, \mathbb{A}, G, \rho)$ and the finite-state automaton $\mathcal{F} = (\widehat{X}, \widehat{X}_0, \widehat{U}^{\mathcal{NN}+\delta}, T_{\mathcal{F}})$, where the transition probabilities are given by:

$$T_{\mathcal{F}}(q'|q,u) = \begin{cases} \Pr(q'|\mathrm{ct}(q), u) & \text{if } u \in \{\mathcal{NN}(\mathrm{ct}(q), s, t) \pm i\delta | t \in \bar{H}, i \in \bar{I}\} \\ 0 & \text{otherwise.} \end{cases}$$

As a property of the product automaton, the probability of the finite-state automaton $\mathcal{F}$ satisfying the given specification φ equals the probability of the product automaton $\Sigma_{\varphi}^{\mathcal{NN}+\delta} = \mathcal{F} \otimes \mathcal{A}_{\varphi}$ reaching the accepting set $X_G^{\otimes}$. Then, we have $\Pr\left(\xi_{\widehat{\Psi}_k}^{q} \models \varphi\right) \geq p$ for all $q \in \widehat{X}_0$, where $\xi_{\widehat{\Psi}_k}^{q}$ is a trajectory of $\mathcal{F}$ starting from q under the control of $\widehat{\Psi}_k$. Finally, we use the fact that the difference in the satisfaction probabilities between the finite-state automaton $\mathcal{F}$ and the dynamical system (4.1) is upper bounded as follows [83, Theorem 2.1]:

$$\left|\Pr\left(\xi_{\widehat{\Psi}_k}^{q} \models \varphi\right) - \Pr\left(\xi_{\Psi_k}^{x} \models \varphi\right)\right| \leq \lambda H \mathcal{A} L_{\tau}, \ \forall x \in q.$$

$\square$

2.4.2 Projection and Lift Error

Now, we focus on the performance analysis of NNSynth, i.e., the optimality of controllers returned by Algorithm 8 in terms of minimizing the cost functional J. To circumvent evaluating the gradient of symbolic controllers $\nabla J(\Psi_k)$, each iteration of Algorithm 8 lifts a symbolic controller Ψ_k to a neural network $\mathcal{NN}_k$ (line 9 in Algorithm 8), updates the neural network using its gradient $\nabla J(\mathcal{NN}_k)$ (line 10 in Algorithm 8), and projects the updated neural network $\mathcal{NN}_{k+1}$ back to a symbolic controller Ψ_{k+1} (line 11 in Algorithm 8). In this subsection, we focus on the lift and projection procedures, and present the overall performance guarantee in the next subsection.

Recall that the symbolic controller $\Psi : X \times S \times \bar{H} \to U$ returned by NNSynth is given by $\Psi(x, s, t) = \widehat{\Psi}(q, s, t)$ for all $x \in q$ (line 20 of Algorithm 9). Such a controller Ψ is known as an abstraction-based controller, which is featured by applying the same control action at all states in the same abstract state $q \in \widehat{X}$ (when the DFA's states $s \in S$ and the time steps

$t \in \bar{H}$ are fixed). We use $\mathcal{C}_{\text{abs}}$ to denote the set of all abstraction-based controllers:

$$\begin{aligned}\mathcal{C}_{\text{abs}} = \{\Psi : X \times S \times \bar{H} \to U \mid \Psi(x_1, s, t) = \Psi(x_2, s, t) \\ \text{if } \text{abs}(x_1) = \text{abs}(x_2), \forall s \in S, \forall t \in \bar{H}\}.\end{aligned} \tag{2.10}$$

With the notation of $\mathcal{C}_{\text{abs}}$, our framework can be viewed via the lens of mirror-descent [], i.e., NNSynth updates the controllers in the neural network space and projects the trained NNs back to the set of abstraction-based controllers $\mathcal{C}_{\text{abs}}$ in each iteration of the synthsis loop. Given a neural network $\mathcal{NN}_{k+1}$, the procedure Project-by-Synth (Algorithm 9) projects $\mathcal{NN}_{k+1}$ to an abstraction-based controller $\Psi_{k+1} \in \mathcal{C}_{\text{abs}}$ while maximizing the probability of satisfying the specification φ at the meantime. This leads to the projection error compared the abstraction-based controller $\Psi^*_{k+1} \in \mathcal{C}_{\text{abs}}$ that minimizes the distance to the neural network, i.e., $\Psi^*_{k+1} = \text{argmin}_{\Psi \in \mathcal{C}_{\text{abs}}} \|\Psi - \mathcal{NN}_{k+1}\|$. In Proposition 2.4, we upper bound the difference between the abstraction-based controller Ψ_{k+1} returned by the projection procedure Project-by-Synth and the actual minimizer Ψ^*_{k+1} of the distance to $\mathcal{NN}_{k+1}$.

Proposition 2.3. *Let $\Upsilon : X \times S \times \bar{H} \to U$ be an arbitrary controller and $\Psi^* = argmin_{\Psi \in \mathcal{C}_{abs}} \|\Psi - \Upsilon\|^2$. Consider an arbitrary abstract state $q \in \widehat{X}$, a DFA's state $s \in S$, and a time step $t \in \bar{H}$. If $\exists c \in \mathbb{R}^+$ such that $\|\Upsilon(x_1, s, t) - \Upsilon(x_2, s, t)\| \le c$ for all $x_1, x_2 \in q$, then $\exists y \in q$ such that $\|\Upsilon(y, s, t) - \Psi^*(\text{ct}(q), s, t)\| \le c$.*

Proof. By evaluating Ψ at the centers $\text{ct}(q)$, we have

$$\begin{aligned}\Psi^* &= \underset{\Psi \in \mathcal{C}_{\text{abs}}}{\text{argmin}} \|\Psi - \Upsilon\|^2 \\ &= \underset{\Psi \in \mathcal{C}_{\text{abs}}}{\text{argmin}} \sum_{t \in \bar{H}} \sum_{s \in S} \int_X \|\Psi(x, s, t) - \Upsilon(x, s, t)\|^2 dx \\ &= \underset{\Psi \in \mathcal{C}_{\text{abs}}}{\text{argmin}} \sum_{t \in \bar{H}} \sum_{s \in S} \sum_{q \in \widehat{X}} \int_q \|\Psi(\text{ct}(q), s, t) - \Upsilon(x, s, t)\|^2 dx.\end{aligned} \tag{2.11}$$

Since the value of $\Psi^*(\text{ct}(q), s, t)$ can be chosen independently at different $(q, s, t) \in \widehat{X} \times S \times \bar{H}$, (2.11) yields:

$$\Psi^*(\text{ct}(q), s, t) = \operatorname*{argmin}_{u \in U} \int_q \|u - \Upsilon(x, s, t)\|^2 dx. \tag{2.12}$$

Now, we prove the proposition by contradiction. Assume that $\forall x \in q$, $\|\Upsilon(x, s, t) - \Psi^*(\text{ct}(q), s, t)\| > c$, then $\int_q \|\Upsilon(x, s, t) - \Psi^*(\text{ct}(q), s, t)\|^2 dx > c^2 \mathcal{A}_q$, where $\mathcal{A}_q$ is the Lebesgue measure of the abstract state q. This along with (2.12) yields:

$$\min_{u \in U} \int_q \|u - \Upsilon(x, s, t)\|^2 dx > c^2 \mathcal{A}_q. \tag{2.13}$$

Since $\|\Upsilon(x_1, s, t) - \Upsilon(x_2, s, t)\| \leq c$ for all $x_1, x_2 \in q$, by choosing $u = \Upsilon(x', s, t)$ with an arbitrary $x' \in q$, we have $\int_q \|u - \Upsilon(x, s, t)\|^2 dx \leq c^2 \mathcal{A}_q$, which contradicts (2.13). □

Proposition 2.4. *At an arbitrary iteration $k \in \{0, \ldots, K-1\}$ in Algorithm 8, let Ψ_{k+1} be the abstraction-based controller returned by the procedure* Project-by-Synth *(line 11 in Algorithm 8), and $\Psi^*_{k+1} = argmin_{\Psi \in \mathcal{C}_{abs}} \|\Psi - \mathcal{NN}_{k+1}\|$, where $\mathcal{NN}_{k+1}$ is the updated NN (line 10 in Algorithm 8). Then, the difference between Ψ_{k+1} and Ψ^*_{k+1} is upper bounded as follows:*

$$\|\Psi_{k+1} - \Psi^*_{k+1}\| = O\left(\delta I + \lambda L_{nn}\right). \tag{2.14}$$

In the above proposisition, L_{nn} is the Lipshitz constant of the neural network $\mathcal{NN}_{k+1} : X \times S \times \bar{H} \to U$, i.e.:

$$\|\mathcal{NN}_{k+1}(x_1, s, t) - \mathcal{NN}_{k+1}(x_2, s, t)\| \leq L_{nn} \|x_1 - x_2\| \tag{2.15}$$

for all $x_1, x_2 \in X$, $s \in S$, and $t \in \bar{H}$. The parameter λ is the grid size in partitioning the state space X, δ and I are the precision and range parameters in system augmentation, respectively (see (2.7)).

Proof. Since $\Psi_{k+1}, \Psi^*_{k+1} \in \mathcal{C}_{\text{abs}}$, we evaluate their values at the center $\text{ct}(q) \in X$ of each abstract state $q \in \widehat{X}$:

$$\begin{aligned}
&\|\Psi_{k+1} - \Psi^*_{k+1}\|^2 \\
&= \sum_{t\in\bar{H}} \sum_{s\in S} \int_X \|\Psi_{k+1}(x,s,t) - \Psi^*_{k+1}(x,s,t)\|^2 dx \\
&= \sum_{t\in\bar{H}} \sum_{s\in S} \sum_{q\in\widehat{X}} \mathcal{A}_q \|\Psi_{k+1}(\text{ct}(q),s,t) - \Psi^*_{k+1}(\text{ct}(q),s,t)\|^2 \\
&\leq |S|H\mathcal{A} \max_{\substack{(q,s,t) \\ \in \widehat{X}\times S\times\bar{H}}} \|\Psi_{k+1}(\text{ct}(q),s,t) - \Psi^*_{k+1}(\text{ct}(q),s,t)\|^2, \qquad (2.16)
\end{aligned}$$

where $\mathcal{A}_q$ and $\mathcal{A}$ are the Lebesgue measure of the abstract state q and the state space X, respectively. Consider an arbitrary choice of $q \in \widehat{X}$, $s \in S$, and $t \in \bar{H}$. By Proposition 2.3, since $\|\mathcal{NN}_{k+1}(x_1,s,t) - \mathcal{NN}_{k+1}(x_2,s,t)\| \leq \lambda L_{nn}$ for all $x_1, x_2 \in q$, there exists $y \in q$ such that:

$$\|\mathcal{NN}_{k+1}(y,s,t) - \Psi^*_{k+1}(\text{ct}(q),s,t)\| \leq \lambda L_{nn}. \qquad (2.17)$$

With this choice of y, we have:

$$\begin{aligned}
&\|\Psi^*_{k+1}(\text{ct}(q),s,t) - \Psi_{k+1}(\text{ct}(q),s,t)\| \\
&\leq \|\Psi^*_{k+1}(\text{ct}(q),s,t) - \mathcal{NN}_{k+1}(y,s,t)\| + \|\mathcal{NN}_{k+1}(\text{ct}(q),s,t) \\
&- \mathcal{NN}_{k+1}(y,s,t)\| + \|\Psi_{k+1}(\text{ct}(q),s,t) - \mathcal{NN}_{k+1}(\text{ct}(q),s,t)\| \\
&\leq \sqrt{m}\delta I + 2\lambda L_{nn}, \qquad (2.18)
\end{aligned}$$

where the last step uses (2.17) and $\|\Psi_{k+1}(\text{ct}(q),s,t) - \mathcal{NN}_{k+1}(\text{ct}(q),s,t)\| \leq \sqrt{m}\delta I$. The later equation holds since Ψ_{k+1} is the projection of $\mathcal{NN}_{k+1}$ through the procedure PROJECT-BY-SYNTH, which first evaluates $\mathcal{NN}_{k+1}$ at the centers $\text{ct}(q)$, and then augments local actions within the radius δI in each of the m dimensions of $U \subset \mathbb{R}^m$. Since (2.18) holds for an arbitrary choice of $(q, s, t) \in \widehat{X} \times S \times \bar{H}$, substituting (2.18) into (2.16) yields (2.14). $\square$

The Lift procedure (line 9 in Algorithm 8) trains a neural network $\mathcal{NN}_k$ whose output is close to that of the abstraction-based controller Ψ_k. In particular, we use the training approach [173] to memorize the outputs of Ψ_k at the centers of abstract states, i.e., $\Psi_k(\mathrm{ct}(q), s, t) = \mathcal{NN}_k(\mathrm{ct}(q), s, t)$ for all $q \in \widehat{X}$, $s \in S$, and $t \in \bar{H}$. The following proposition provides an upper bound for the lift error.

Proposition 2.5. *Consider the neural network $\mathcal{NN}_k$ is given by lifting an abstraction-based controller Ψ_k, i.e., $\mathcal{NN}_k = \text{Lift}(\Psi_k)$ (line 9 in Algorithm 8), and the Lipschitz constant of $\mathcal{NN}_k$ is L_{nn}, then $\|\mathcal{NN}_k - \Psi_k\| = O(\lambda L_{nn})$.*

Proof. By evaluating Ψ_k at the centers $\mathrm{ct}(q)$, we have:

$$\begin{aligned}
&\|\mathcal{NN}_k - \Psi_k\|^2 \\
&= \sum_{t\in\bar{H}} \sum_{s\in S} \int_X \|\mathcal{NN}_k(x, s, t) - \Psi_k(x, s, t)\|^2 dx \\
&= \sum_{t\in\bar{H}} \sum_{s\in S} \sum_{q\in\widehat{X}} \int_q \|\mathcal{NN}_k(x, s, t) - \Psi_k(\mathrm{ct}(q), s, t)\|^2 dx \\
&\leq |S| H \mathcal{A} (\lambda L_{nn})^2 \qquad (2.19)
\end{aligned}$$

where $\mathcal{A}$ is the Lebesgue measure of the state space X. The last step of (2.19) is due to:

$$\begin{aligned}
&\|\mathcal{NN}_k(x, s, t) - \Psi_k(\mathrm{ct}(q), s, t)\| \leq \|\mathcal{NN}_k(x, s, t) \\
&- \mathcal{NN}_k(\mathrm{ct}(q), s, t)\| + \|\mathcal{NN}_k(\mathrm{ct}(q), s, t) - \Psi_k(\mathrm{ct}(q), s, t)\| \\
&\leq \lambda L_{nn} + c, \qquad (2.20)
\end{aligned}$$

for all $x \in q$, where c is constant given by:

$$c = \max_{(q,s,t)\in\widehat{X}\times S\times\bar{H}} \|\mathcal{NN}_k(\mathrm{ct}(q), s, t) - \Psi_k(\mathrm{ct}(q), s, t)\|$$

which can be zero by training NN to memorize the outputs of Ψ_k at all the centers of abstract states. $\square$

2.4.3 Overall Regret

In Algorithm 8, the procedure UPDATE (line 10 in Algorithm 8) improves the neural network $\mathcal{NN}_k$ using its gradient, i.e., $\mathcal{NN}_{k+1} = \mathcal{NN}_k - \eta\nabla J(\mathcal{NN}_k)$, where η is the learning rate and $\nabla J(\mathcal{NN}_k)$ can be evaluated as (2.4). This can be treated as an approximation of updating the abstraction-based controller $\Psi_k \in \mathcal{C}_{\text{abs}}$ directly through $\Upsilon_{k+1} = \Psi_k - \eta\nabla J(\Psi_k)$, where $\Upsilon_{k+1} : X \times S \times \bar{H} \to U$ is not necessarily an abstraction-based controller and needs to be projected back to the set $\mathcal{C}_{\text{abs}}$. We take into account this gradient approximation error, along with the projection and lift errors in the previous subsection, to provide the overall performance guarantee of NNSynth in terms of regret as follows:

Theorem 2.6. *Consider the synthesis loop (line 5-11 in Algorithm 8) executes K iterations and the abstraction-based controller obtained at the end of each iteration is Ψ_k, $k = 1, \dots, K$. Let Ψ^* be the optimal abstraction-based controller, i.e., $\Psi^* = \mathrm{argmin}_{\Psi\in\mathcal{C}_{abs}} J(\Psi)$. Then, the regret over K iterations is upper bounded as follows:*

$$\frac{1}{K}\sum_{k=1}^{K} J(\Psi_k) - J(\Psi^*) = O\left(\frac{1}{\eta K} + \frac{\delta I + \lambda L_{nn}}{\eta} + \lambda L_{nn} + \eta\right). \qquad (2.21)$$

In the above theorem, by choosing the learning rate $\eta = \sqrt{\frac{1}{K} + \delta I + \lambda L_{nn}}$, (2.21) becomes:

$$\frac{1}{K}\sum_{k=1}^{K} J(\Psi_k) - J(\Psi^*) = O\left(\lambda L_{nn} + \sqrt{\frac{1}{K} + \delta I + \lambda L_{nn}}\right),$$

which shows that when the precision parameters λ and δ approach zero, the regret can be arbitrarily small by increasing the number of iterations K. In general, the choice of parameters λ and δ depends on the satisfaction probability and regret that need to be achieved, and these parameters can be determined based on Theorem 2.2 and Theorem 2.6.

In the proof of Theorem 2.6, we use $D : \mathcal{U} \times \mathcal{U} \to \mathbb{R}$ to denote the distance between two controllers for simplicity of notation, i.e., $D(\Upsilon_1, \Upsilon_2) = \frac{1}{2}\|\Upsilon_1 - \Upsilon_2\|^2$, where $\mathcal{U} = \{\Upsilon : X \times S \times \bar{H} \to U\}$. We will use the identity that for any $\Upsilon_1, \Upsilon_2, \Upsilon_3 \in \mathcal{U}$ it holds that:

$$\langle \Upsilon_1 - \Upsilon_2, \Upsilon_1 - \Upsilon_3 \rangle = D(\Upsilon_1, \Upsilon_2) + D(\Upsilon_3, \Upsilon_1) - D(\Upsilon_3, \Upsilon_2). \tag{2.22}$$

Proof. Let β_k be the error due to the gradient approximation using neural networks, i.e., $\mathcal{NN}_{k+1} = \Upsilon_{k+1} + \beta_k$, where $\mathcal{NN}_{k+1} = \mathcal{NN}_k - \eta\nabla J(\mathcal{NN}_k)$ and $\Upsilon_{k+1} = \Psi_k - \eta\nabla J(\Psi_k)$. Due to the convexity of J over $\mathcal{C}_{\text{abs}}$, we have that for any $\Psi \in \mathcal{C}_{\text{abs}}$:

$$J(\Psi_k) - J(\Psi) \leq \langle \nabla J(\Psi_k), \Psi_k - \Psi \rangle. \tag{2.23}$$

We now bound the RHS of (2.23):

$$\langle \nabla J(\Psi_k), \Psi_k - \Psi \rangle = \frac{1}{\eta}\langle \Psi_k - \Upsilon_{k+1}, \Psi_k - \Psi \rangle \tag{2.24}$$

$$= \frac{1}{\eta}\langle \Psi_k - \mathcal{NN}_{k+1}, \Psi_k - \Psi \rangle + \frac{1}{\eta}\langle \beta_k, \Psi_k - \Psi \rangle \tag{2.25}$$

$$= \frac{1}{\eta}(D(\Psi, \Psi_k) - D(\Psi, \mathcal{NN}_{k+1}) + D(\Psi_k, \mathcal{NN}_{k+1})) + \frac{1}{\eta}\langle \beta_k, \Psi_k - \Psi \rangle \tag{2.26}$$

$$\begin{aligned}\le \frac{1}{\eta}(D(\Psi, \Psi_k) - D(\Psi, \Psi^*_{k+1}) - D(\Psi^*_{k+1}, \mathcal{NN}_{k+1}) \\ + D(\Psi_k, \mathcal{NN}_{k+1})) + \frac{1}{\eta}\langle \beta_k, \Psi_k - \Psi \rangle\end{aligned} \tag{2.27}$$

$$\le \frac{1}{\eta}(D(\Psi, \Psi_k) - D(\Psi, \Psi_{k+1}) + \varepsilon_k + \gamma_k) + \frac{1}{\eta}\langle \beta_k, \Psi_k - \Psi \rangle, \tag{2.28}$$

where the projection error $\varepsilon_k \triangleq D(\Psi, \Psi_{k+1}) - D(\Psi, \Psi^*_{k+1})$ and $\gamma_k \triangleq D(\Psi_k, \mathcal{NN}_{k+1}) - D(\Psi^*_{k+1}, \mathcal{NN}_{k+1})$ is the relative improvement. In the above, (2.26) uses the identity (2.22) above; (2.27) is due to the generalized Pythagorean theorem: if $\Psi^*_{k+1} = \text{argmin}_{\Psi' \in \mathcal{C}_{\text{abs}}} D(\Psi', \mathcal{NN}_{k+1})$, then it holds that $D(\Psi, \mathcal{NN}_{k+1}) \ge D(\Psi, \Psi^*_{k+1}) + D(\Psi^*_{k+1}, \mathcal{NN}_{k+1})$ for all $\Psi \in \mathcal{C}_{\text{abs}}$.

The projection error ε_k can be bounded as follows:

$$\varepsilon_k \triangleq D(\Psi, \Psi_{k+1}) - D(\Psi, \Psi^*_{k+1}) \tag{2.29}$$

$$\le \langle \Psi_{k+1} - \Psi^*_{k+1}, \Psi_{k+1} - \Psi \rangle \tag{2.30}$$

$$\le \|\Psi_{k+1} - \Psi^*_{k+1}\| \|\Psi_{k+1} - \Psi\| \tag{2.31}$$

$$\le d\|\Psi_{k+1} - \Psi^*_{k+1}\|, \tag{2.32}$$

where the diameter $d \triangleq \sup_{\Psi, \Psi' \in \mathcal{C}_{\text{abs}}} \|\Psi - \Psi'\|$. In the above, (2.30) uses the identity (2.22) and the fact that the distance defined by D is nonnegative; (2.31) is due to Cauchy–Schwarz inequality.

The relative improvement γ_k can be bounded as follows:

$$\gamma_k \triangleq D(\Psi_k, \mathcal{NN}_{k+1}) - D(\Psi^*_{k+1}, \mathcal{NN}_{k+1}) \tag{2.33}$$

$$= \frac{1}{2}\|\Psi_k\|^2 - \frac{1}{2}\|\Psi^*_{k+1}\|^2 + \langle \mathcal{NN}_{k+1}, \Psi^*_{k+1} - \Psi_k\rangle \tag{2.34}$$

$$\leq \langle \mathcal{NN}_{k+1} - \Psi_k, \Psi^*_{k+1} - \Psi_k\rangle - \frac{1}{2}\|\Psi_k - \Psi^*_{k+1}\|^2 \tag{2.35}$$

$$\leq \langle \Upsilon_{k+1} - \Psi_k, \Psi^*_{k+1} - \Psi_k\rangle - \frac{1}{2}\|\Psi_k - \Psi^*_{k+1}\|^2 + \langle \beta_k, \Psi^*_{k+1} - \Psi_k\rangle$$

$$\leq -\eta\langle \nabla J(\Psi_k), \Psi^*_{k+1} - \Psi_k\rangle - \frac{1}{2}\|\Psi_k - \Psi^*_{k+1}\|^2 + \langle \beta_k, \Psi^*_{k+1} - \Psi_k\rangle$$

$$\leq \frac{1}{2}\eta^2 L_J^2 + d\|\beta_k\|, \tag{2.36}$$

where L_J is the Lipschitz constant of J; (2.35) is due to the strong convexity $\frac{1}{2}\|\Psi^*_{k+1}\|^2 \geq \frac{1}{2}\|\Psi_k\|^2 + \langle \Psi_k, \Psi^*_{k+1} - \Psi_k\rangle + \frac{1}{2}\|\Psi_k - \Psi^*_{k+1}\|^2$; (2.36) is because $az - bz^2 \leq \frac{a^2}{4b}$, $\forall z \in \mathbb{R}$ and uses Cauchy–Schwarz inequality.

The error β_k can also be bounded:

$$\|\beta_k\| = \|\mathcal{NN}_{k+1} - \Upsilon_{k+1}\| \tag{2.37}$$

$$\leq \|\mathcal{NN}_k - \Psi_k\| + \eta\|\nabla J(\mathcal{NN}_k) - \nabla J(\Psi_k)\| \tag{2.38}$$

$$\leq (1 + \eta c_j)\|\mathcal{NN}_k - \Psi_k\| \tag{2.39}$$

where c_j is the Lipschitz constant of ∇J.

Substitute (2.32), (2.36), (2.39) into (2.28) yields:

$$\langle \nabla J(\Psi_k), \Psi_k - \Psi\rangle$$
$$\leq \frac{1}{\eta}(D(\Psi, \Psi_k) - D(\Psi, \Psi_{k+1}) + d\|\Psi_{k+1} - \Psi^*_{k+1}\| + 2d(1 + \eta c_j)\|\mathcal{NN}_k - \Psi_k\|) + \frac{1}{2}\eta L_J^2. \tag{2.40}$$

With (2.40), the summation of (2.23) over K iterations yields:

$$\begin{aligned}
&\frac{1}{K}\sum_{k=1}^{K} J(\Psi_k) - J(\Psi) \leq \frac{1}{\eta K}(D(\Psi, \Psi_1) - D(\Psi, \Psi_{K+1})) \\
&+ \frac{d}{\eta}\|\Psi_{k+1} - \Psi^*_{k+1}\| + \frac{2d}{\eta}(1 + \eta c_j)\|\mathcal{NN}_k - \Psi_k\| + \frac{\eta}{2\alpha}L_J^2. \qquad (2.41)
\end{aligned}$$

By following the similar process as (2.29)-(2.32), we have $D(\Psi, \Psi_1) - D(\Psi, \Psi_{K+1}) \leq d^2$. By Proposition 2.4, the projection error $\|\Psi_{k+1} - \Psi^*_{k+1}\| = O(\delta I + \lambda L_{nn})$. By Proposition 2.5, the lift error $\|\mathcal{NN}_k - \Psi_k\| = O(\lambda L_{nn})$. With these bounds, (2.41) leads to (2.21). □

2.5 Results

We implemented NNSynth in Python and evaluated its performance on a Macbook Pro 15 with 32 GB RAM and Intel Core i9 2.4-GHz CPU. To compare with existing tools, we run all experiments on a single CPU core without using GPUs to accelerate neural network training.

Table 2.1: Comparison between NNSynth and AMYTISS.

Benchmark	**2-d Robot**	**5-d Room Temp.**	**5-d Traffic**
Specification φ	Reach-avoid	Safety	Safety
Specification horizon H	16	8	7
Problem complexity $\lvert\widehat{X} \times \widehat{U}\rvert$	705600	3429216	1.25×10^8
Satisfaction Probability V_{avg}	96%	95%	80%
NNSynth (time) [s]	49.0	319.1	367.7
AMYTISS (time) [s]	108.4	34640.0	23100.0
Speedup	**2 x**	**108 x**	**62 x**

2.5.1 Benchmarks and Performance

We start by evaluating NNSynth on three benchmarks with an increasing number of complexity. We compare NNSynth with the state-of-the-art tool in synthesizing controllers for stochastic systems, AMYTISS [82]. Table 2.1 summarizes the comparison results. For each of the benchmarks, we list the specification φ used in this experiment along with its horizon H, the complexity of the problem measured by the number of abstract states times the number of discretized control actions $|\widehat{X} \times \widehat{U}|$, the average probability of satisfying the specification (averaged over the state space) V_{avg}, the execution time for each of the two tools, and the corresponding speedup. Indeed, the last row in Table 2.1 empirically proves that using neural networks to guide the controller synthesis provides significant improvement to the overall execution time. Below, we provide more details about each of the benchmarks.

Experiment #1: 2-d Robot. Consider a 2-dimensional robot model given by:

$$\begin{aligned} x_1^{(t+1)} &= x_1^{(t)} + u_1^{(t)}\cos(u_2^{(t)}) + \varsigma_1^{(t)} \\ x_2^{(t+1)} &= x_2^{(t)} + u_2^{(t)}\sin(u_2^{(t)}) + \varsigma_2^{(t)}, \end{aligned}$$

where the state space $X = [-10, 10] \times [-10, 10]$, control input space $U = [-1, 1] \times [-1, 1]$, and the noise $(\varsigma_1, \varsigma_2)$ follows a Gaussian distribution with covariance matrix $\Sigma = \text{diag}(0.75, 0.75)$. We are interested in the task of steering the robot into a goal set $[5, 7] \times [5, 7]$ in 16 time steps, while avoiding the obstacle set $[-2, 2] \times [-2, 2]$ (see Figure 2.2).

To construct the abstraction-based controller, we partition the state space with discretization parameters $(0.5, 0.5)$, and the input space with $(0.1, 0.1)$. This leads to a total number of $|\widehat{X}| = 1600$ abstract states and $|\widehat{U}| = 441$ control actions (by including the upper and lower limits of the input space as additional control actions) leading to a complexity of $|\widehat{X} \times \widehat{U}| = 705600$. NNSynth starts by training a neural network using imitation learning with a total of 121 expert trajectories. The neural network consists of two hidden layers

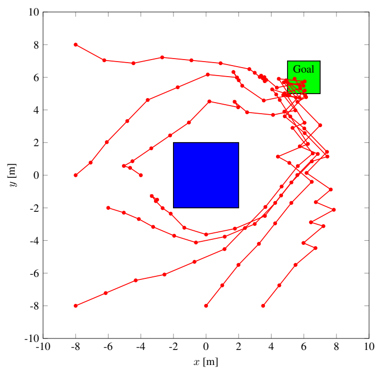


Figure 2.2: Closed-loop trajectories sampled from different initial states using the synthesized controller in Experiment #1.

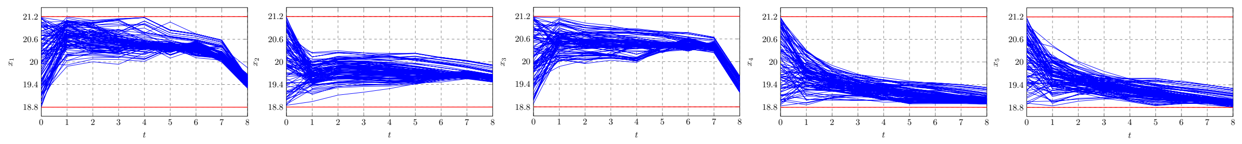


Figure 2.3: State trajectories sampled from different initial conditions using the synthesized controller in Experiment #2.

and ten neurons per hidden layer. We used Keras to train the neural network with the default adaptive learning rate optimization algorithm ADAM. By setting $\delta = 0.1$ (the same precision used to discretize the input space) and $I = 10$, NNSynth only needs to consider 100 local control actions (out of the $|\widehat{U}| = 441$ total control actions) to construct the finite-state abstraction $\widehat{\Sigma}^{\mathcal{NN}+\delta}$. The controller synthesis is then executed to find a controller $\widehat{\Psi}$ that maximizes the probability of satisfying the specification, and one was found in 49.0 seconds with an average satisfaction probability of 96%. The algorithm terminates in one iteration, and lifting the abstraction-based controller to a NN was not needed.

Using the same discretization parameters, AMYTISS was able to find a controller that satisfies the specs with 93% probability in 108.4 seconds. This shows a $2.2\times$ speedup of our tool (and an increase in the satisfaction probability) thanks to the fact that only 25% of the state-action pairs are considered during the synthesis. These 25% actions are chosen by the neural network that NNSynths used to guide the search. In Figure 2.2, we present 8 example trajectories under the control of $\widehat{\Psi}$, by sampling some initial states.

Experiment #2: 5-d Room Temperature Control. This example considers temperature regulation of 5 rooms each equipped with a heater and connected on a circle [82]. The state variables are temperatures of individual rooms, and the evolution of the 5 room temperatures is described as:

$$\begin{aligned}
T_i^{(t+1)} &= a_{ii}T_i^{(t)} + \gamma T_h u_i^{(t)} + \eta w_i^{(t)} + \beta T_{ei} + 0.01\varsigma_i^{(t)}, i \in \{1,3\} \\
T_i^{(t+1)} &= b_{ii}T_i^{(t)} + \eta w_i^{(t)} + \beta T_{ei} + 0.01\varsigma_i^{(t)},\ i \in \{2,4,5\}
\end{aligned}$$

where $a_{ii} = (1 - 2\eta - \beta - \gamma u_i^{(t)})$, $b_{ii} = (1 - 2\eta - \beta)$, and $w_i^{(t)} = T_{i-1}^{(t)} + T_{i+1}^{(t)}$ (with $T_0 = T_5$ and $T_6 = T_1$), and the parameters $\eta = 0.3$, $\beta = 0.022$, $\gamma = 0.05$, $T_{ei} = -1$, $T_h = 50$.

We consider a safety specification that requires the temperature of each room to maintain in the safe set $[18.8, 21.2]$ for at least 8 time steps. We partition the state space with grid size 0.4 in each dimension, and use the grid size $(0.05, 0.05)$ for the input space $U = [0, 1] \times [0, 1]$. Similar to the previous benchmark, NNSynth trains a neural network with two layers and ten neurons per hidden layer using 935 trajectories. With I set to 7, NNSynth used only 49 local control actions (out of 441 total control actions) to compute the abstraction and synthesize a controller. As shown in Table 2.1, NNSynth achieves a satisfaction probability of 95% and 108× speedup compared to AMYTISS. In Figure 2.3, we sample 100 initial states and present the evolution of the 5 state variables, which are all maintained within the safe set for at least 8 steps under the abstraction-based controller provided by NNSynth.

Experiment #3: 5-d Road Traffic Network. This example considers a road traffic network divided into 5 cells, and state variables x_i denote the number of vehicles per cell [82].

The 5-d road traffic network is modeled as:

$$
\begin{aligned}
x_1^{(t+1)} &= (1 - \frac{\tau v_1}{L_1})x_1^{(t)} + \frac{\tau v_5}{L_5} w_1^{(t)} + 6u_1^{(t)} + 0.7\varsigma_1^{(t)} \\
x_i^{(t+1)} &= (1 - \frac{\tau v_i}{L_i} - q)x_i^{(t)} + \frac{\tau v_{i-1}}{L_{i-1}} w_i^{(t)} + 0.7\varsigma_i^{(t)}, \ i \in \{2, 4\} \\
x_3^{(t+1)} &= (1 - \frac{\tau v_3}{L_3})x_3^{(t)} + \frac{\tau v_2}{L_2} w_3^{(t)} + 8u_2^{(t)} + 0.7\varsigma_3^{(t)} \\
x_5^{(t+1)} &= (1 - \frac{\tau v_5}{L_5})x_5^{(t)} + \frac{\tau v_4}{L_4} w_5^{(t)} + 0.7\varsigma_5^{(t)}
\end{aligned}
$$

where $w_i^{(t)} = x_{i-1}^{(t)}$ (with $x_0 = x_5$). Given the state space $X = [0, 10]^5$, the input space $U = [0, 1]^2$, a noise co-variance matrix $\Sigma = \mathrm{diag}(0.7, 0.7, 0.7, 0.7, 0.7)$, and a probability cut-off 1e$-$4, we are interested in designing a control strategy that keeps the number of vehicles per cell in a safety set $[0, 10]$ for at least 7 steps. To show the scalability of NNSynth, we partition the state space and the input space into $|\widehat{X}| = 12500$ and $|\widehat{U}| = 10000$ abstractions, respectively. This leads to a problem complexity in the order of 10^8 control-action pairs. As shown in Table 2.1, NNSynth was able to solve this problem in 367.7 seconds achieving more than 60$\times$ speedup compared with AMYTISS.

2.5.2 Further Insights

Beyond the performance evaluation, we conducted experiments to gain insights on the interaction between neural network training and abstraction-based controller synthesis. In particular, we aim to understand two questions: (i) how does the flexibility in the system augmentation (parameterized by I) help to discover the abstraction-based controller, and (ii) how does the abstraction-based controller help the neural network training?

Experiment #4: Effect of the parameter I on performance. To answer the first question, we vary the number of local actions that are considered at each abstract state. To that end, Table 2.2 shows the result of running NNSynth with different values of I. We report the probability of satisfying the specifications at the end along with the execution time.

In the 2-d robot case, the satisfaction probability grows from 55% to 96% by increasing the number of local state-control action pairs from 4 to 100. This shows that the neural network by itself is far away from the optimal control policy even if it is sufficiently trained. The reason behind this could be the neural network training is stuck at a local optimal. As favorable to NNSynth, abstraction-based controller synthesis can move away from the local optimal and further leads to better controllers, such as the one with satisfaction probability 96% in the 2-d robot example. Similar pattern is observed in the other benchmark.

Experiment #5: Effect of lifting the abstraction controller to a neural network on performance. Now, consider the second question. We train the neural network for 50 epochs in each iteration and compare the satisfaction probabilities for the synthesized controllers after each iteration in Table 2.3. After five iterations of training, synthesizing a controller, lifting to a NN, and retraining, the NN receives a total of 250 epochs of training. For comparison, we also record the base case where we train a neural network for only one iteration but with 250 epochs (the same total number of epochs as that accumulated over five iterations) but without lifting from the abstraction controller to the NN.

In the 2d Robot benchmark, the base case of 1 iteration with 250 epochs, the resulting controller achieved a 54% probability of satisfying the specifications. On the other side, with several iterations of neural network training, controller synthesis, and lifting the controller to a neural network, the resulting controllers improve over iterations. By the end of the 5th iteration, the neural network accumulates 250 epochs (same as the base case), but the resulting satisfaction probability increases to 90%. This is a clear evidence that lifting the synthesized controller to a neural network helps with the overall training of neural networks.

Table 2.2: Numerical results for Experiment #4.

Benchmark	**Number of local control-action pairs ($I \times I$)**	**Satisfaction Probability $V_{\mathbf{avg}}$**	**Execution time [s]**
2-d Robot	4	55%	21.4
	16	81%	23.5
	49	90%	31.2
	100	96%	48.1
5-d Room Temp.	4	65%	106.2
	16	94%	210.3
	49	95%	324.0
	100	95%	630.9

Table 2.3: Numerical results for Experiment #5.

Benchmark	**Iteration number**	**Total training epochs at the end of iterations**	**Satisfaction Probability $V_{\mathbf{avg}}$ at the end of iterations**
2-d Robot	(base case) 1	250	54%
	1	50	30%
	2	100	64%
	3	150	82%
	4	200	88%
	5	250	90%

Chapter 3

DoS-Resilient Multi-Robot Temporal Logic Motion Planning

In this chapter, we present an efficient multi-robot motion planning algorithm for missions captured by linear temporal logic (LTL) specifications, in the presence of bounded disturbances and denial-of-service (DoS) attacks against the communication between robots and base stations. Given an LTL formula ψ, our goal is to construct robot trajectories, and associated control strategies, to satisfy ψ and continuously establish communication paths between robots and base stations despite the DoS attacks and the disturbances on the robot states. Our approach combines and extends results from robust control and efficient motion planning via satisfiability modulo convex programming (SMC). We first compute a feedback controller that rejects the disturbance together with a perturbation of the DoS-free workspace that accounts for the worst-case disturbance scenario. On the perturbed workspace, we formulate the planning problem as a feasibility problem over Boolean and convex constraints, respectively capturing the DoS-resilient mission constraints and the constraints on the nominal, disturbance-free, robot dynamics. Numerical results show the effectiveness of our algorithm in providing DoS-resilient plans that are robust to disturbances and support the execution of complex missions.

3.1 Introduction

As multi-robot systems are increasingly being considered for a variety of mission-critical and safety-critical applications (e.g., monitoring, disaster relief, healthcare), accounting for the security implications of these technologies becomes key [61]. In fact, the specific nature of these multi-agent, networked autonomous systems, as well as their complexity, expose them to a set of unprecedented threats. Attacks may range from passive eavesdropping of the communication channel for data interception, to active communication jamming for disrupting legitimate transmissions, or the injection of malicious robots in the swarm [177, 59, 94]. Devising effective methods to account for these threats since the early stages of the design process, rather than undesirably or expensively retrofitting existing designs, is an open challenge.

A major difficulty for providing security guarantees about these systems stems from the need to reason about the tight integration of discrete abstractions (e.g., high-level tasks, intermittent links) with continuous trajectories and lower-level dynamics [61, 105]. This integration can soon become daunting for complex, high-dimensional systems, since a vast hybrid, discrete/continuous space must be explored while accounting for complex geometries, motion dynamics, safety, and temporal goals. The difficulties are further exacerbated by the uncertainties, as in the majority of real-world scenarios, due to internal noise sources, model errors, and unknown or adversarial environments. In this chapter, we address these challenges by focusing on the *resilient multi-robot motion planning problem* for complex missions captured by a high-level formal language, and in the presence of bounded disturbances and denial-of-service (DoS) attacks against the communication between robots and base stations.

Recent work has proposed defense mechanisms for multi-robot systems that can guarantee resilience to communication spoofing attacks [57, 114]. Security mechanisms against communication-jamming attacks have also been studied based on game-theoretic approaches [172, 88, 170] or multi-objective optimization [109]. Differently from these efforts, we consider a mission specified by a linear temporal logic (LTL) [106] formula ψ; we then aim to automatically generate dynamically-feasible robot trajectories, and associated control strategies, that satisfy ψ and guarantee continuous communication between robots and base stations despite the disturbance and the adversarial environment. To do so, we combine and extend results from robust control [112], which separate the concerns of disturbance rejection and trajectory planning, with a satisfiability modulo convex programming (SMC) approach [132, 133, 131], which efficiently reasons about the combination of discrete and convex constraints.

SMC was previously applied to solve reach-avoid and LTL motion planning problems, showing more than two orders of magnitude improvement in execution time with respect to state-of-the-art techniques based on the RRT (Rapidly-exploring Random Trees) and EST (Expansive Space Trees) methods on high-dimensional problems [132, 133, 131]. In this chapter, *we propose a novel SMC encoding that enables directly encapsulating DoS-resilience constraints within the planning problem*, by effectively capturing a notion of communication-based adjacency, in addition to physical adjacency, between workspace locations. Further, *we introduce a robust LTL motion planning formulation* that can efficiently account for disturbances in the robot system states. While LTL has shown to be capable of expressing a rich set of specifications (e.g., safety, progress, response, surveillance, and monitoring) and support algorithmic control synthesis for a variety of applications in robotics and autonomous systems [149, 73, 42, 165, 98, 122], traditional formulations of LTL motion planning do not effectively account for disturbances on the robot trajectories, and tend to become impractical, especially in the presence of adversarial environments. Numerical results show the effectiveness of our approach in providing DoS-resilient plans that are robust to disturbances and support the execution of complex missions [141].

3.2 Problem Formulation

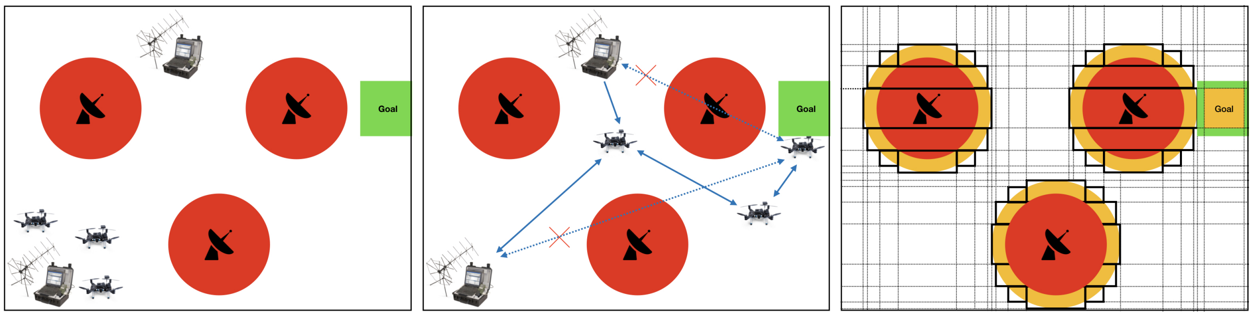


Figure 3.1: (Left) Pictorial representation of a workspace that contains a team of three robots, two base stations, and three jamming radars. The mission is to move at least one of robots to reach the goal location while maintaining communication between all the robots and at least one base station. (Middle) Any communication link that passes through a jamming area is considered under DoS attack. (Right) The workspace is perturbed (yellow) to account for disturbances and a coarse-grain discretization of the free space is computed.

In this section, we introduce models for the robots and the adversarial environment. We consider a team of robots that move in a workspace $\mathcal{W} \subset \mathbb{R}^d$, where d can be 2 or 3, corresponding to a 2-dimensional or 3-dimensional workspace, respectively. We use $\|a\|$ to denote the infinity norm of vector a. Given two sets $S_1 \subset \mathbb{R}^n$ and $S_2 \subset \mathbb{R}^n$, the Minkwoski (vector) sum is defined by $S_1 \oplus S_2 \triangleq \{s_1 + s_2 | s_1 \in S_1, s_2 \in S_2\}$, the Pontryagin (geometric) set difference is $S_1 \ominus S_2 \triangleq \{s | s \oplus S_2 \subseteq S_1\}$. For a constant $\alpha \in [0, 1[$ and a set $S \subset \mathbb{R}^n$, we denote by αS the set $\{\alpha s | s \in S\}$. A closed hyperball in $\mathbb{R}^n$ of radius $r \in \mathbb{R}_{\geq 0}$ is denoted by $\mathcal{B}(r) \triangleq \{x \in \mathbb{R}^n | \|x\| \leq r\}$. For two points $w_1, w_2 \in \mathcal{W}$, we denote by $\mathcal{L}(w_1, w_2)$ the set of points that lie on the line connecting w_1 and w_2, i.e., $\mathcal{L}(w_1, w_2) = \{w | w = s w_1 + (1 - s) w_2,\ 0 \leq s \leq 1\}$. We formulate the Denial-of-Service (DoS) resilient motion planning problem as follows.

3.2.1 Robot, Environment, and Threat Models

We assume that the workspace $\mathcal{W}$ contains a set of N_J adversarial communication-jamming radars. As shown in Fig. 3.1, each radar has an effective jamming radius causing a DoS for any communication passing through it. We denote by $J_k = \{w \in \mathcal{W} | w \in \{j_k\} \oplus \mathcal{B}_{r_k}\}$ the subset of the workspace affected by the kth jamming radar where $j_k \in \mathcal{W}$ is the position of the jamming radar and $r_k \in \mathbb{R}_{\geq 0}$ is its jamming radius. We suppose that the location j_k and radius r_k of each jamming radar are known, and leave the case of uncertain jamming radar position and radius for future work.

We then consider a team of N_R mobile robots operating in this adversarial environment. The mobile robots obey the following motion models:

$$x_{t+1}^i = f(x_t^i, u_t^i) + \theta_t^i, \qquad x_0^i = \overline{x}_0^i \tag{3.1}$$

where $x_t^i \in \mathcal{X} \subseteq \mathbb{R}^n$ is the state of the ith robot at time $t \in \mathbb{N}$, $u_t^i \in \mathcal{U} \subseteq \mathbb{R}^m$ is the ith robot input at time t, selected from the space of admissible controls $\mathcal{U}$, $\overline{x}_0^i$ is the ith robot initial state, and $\theta_t^i \in \Theta \subseteq \mathbb{R}^n$ is the bounded disturbance on the ith robot at time t.

Each robot in the team needs to establish communication, at all times, with one or more base stations in a set of N_B stationary base stations (e.g., to receive mission updates.) We denote by $B_i \in \mathcal{W}$ the location of the ith base station. The communication between base stations and robots can take place directly (single hop) or indirectly (multi-hop) through other robots. Throughout this chapter, we assume a line-of-sight communication model, in which two nodes (robots or base stations) can communicate whenever the straight line connecting them does not pass through a DoS region.

3.2.2 Temporal Logic Specification

In addition to maintaining communication with the base stations at all times, the team must perform a mission that is defined over a set of regions of interest. We assume that the regions of interest are polytopes and partition the workspace as $\mathcal{W} = \bigcup_1^r \mathcal{W}_i$, where $\{\mathcal{W}_1, \ldots, \mathcal{W}_r\}$ is a set of non-overlapping regions. For robot R_i, we can associate to each of the above regions a Boolean proposition in the set $\Pi^i = \{\pi_1^i, \ldots, \pi_r^i\}$, where π_j^i evaluates to one (true) if robot R_i is in region $\mathcal{W}_j$ and zero (false) otherwise. We then denote by $h_{\mathcal{W}\to\Pi^i} : \mathcal{W} \to \Pi^i$ the map from each point $w \in \mathcal{W}$ to the proposition $\pi_j^i \in \Pi^i$ that evaluates to one at w for robot R_i. Moreover, a subset of the state variables of each robot, describing its position (coordinates), is also used to describe $\mathcal{W}$. Therefore, we denote as $h_{\mathcal{X}\to\mathcal{W}} : \mathcal{X} \to \mathcal{W}$ the natural projection of the state x^i onto the workspace $\mathcal{W}$, and by $h_{\mathcal{X}\to\Pi^i}$ the map from the state space of robot R_i to the set of propositions Π^i, obtained after projecting the state onto the workspace, i.e., $h_{\mathcal{X}\to\Pi^i}(x^i) = h_{\mathcal{W}\to\Pi^i}(h_{\mathcal{X}\to\mathcal{W}}(x^i))$.

We express the specification for a multi-robot mission using linear temporal logic (LTL) [106]. LTL formulas can compactly describe temporal orderings of events along the robots' trajectories and express a rich set specifications (e.g., safety, progress, response, surveillance, and monitoring) to capture complex tasks [149, 73, 42, 165, 98, 122]. Let $\Pi = \bigcup_{i=1}^{R} \Pi^i$ be the set of propositions associated with the workspace regions for all robots, as defined above. We consider formulas over a set of atomic propositions Σ, where $\sigma(\pi) \in \Sigma$ is a Boolean or pseudo-Boolean predicate over Π. From atomic propositions in Σ, any LTL formula can be generated according to the following grammar:

$$\psi := \sigma \mid \neg\psi_0 \mid \psi_1 \wedge \psi_2 \mid \psi_1 \vee \psi_2 \mid \psi_1 \;\mathcal{U}\; \psi_2 \mid \psi_1 \;\mathcal{R}\; \psi_2,$$

where ψ_0, ψ_1, ψ_2 are LTL formulas. Based on the above grammar, we can define $false$ and *true* such that $false = \psi \wedge \neg\psi$ and $true = \neg false$. From the temporal operators *until* ($\mathcal{U}$), and *release* ($\mathcal{R}$), we can derive additional temporal operators, for example, *eventually* ($\Diamond$) and *always* ($\Box$), i.e., $\Diamond\psi = \mathit{true}\;\mathcal{U}\;\psi$, and $\Box\psi = \mathit{false}\;\mathcal{R}\;\psi$. We refer the reader to the literature (e.g., [20]) for the formal semantics of LTL. Defining the atomic propositions as Boolean or pseudo-Boolean predicates over Π allows us to express complex multi-robot behaviors like "either robot R_1 or R_2 must be in $\mathcal{W}_1$," via the proposition $\sigma_1 := \pi_1^1 \vee \pi_1^2$, or "at least one robot must be in $\mathcal{W}_2$" using the proposition $\sigma_2 := \sum_{i=1}^{N_R} \pi_2^i \geq 1$.

3.2.3 DoS-Resilient Motion Planning Problem

Despite the power of LTL in capturing complex missions, traditional formulations of LTL motion planning do not account for disturbances on the robot trajectories, which tends to be impractical, especially in the presence of adversarial environments, since disturbances can force some of the robots (or the communication links) to enter the DoS regions, leading to a mission failure. In this chapter, we generalize the classical formulations to account for the uncertainty stemming from disturbances as follows.

Definition 3.1 (Trajectory)**.** *A system trajectory is a tuple including the following infinite sequences:*

- $X = X_0 X_1 X_2 \dots$ *is a sequence of sets of system states where* $X_t = (X_t^0, \dots, X_t^{N_R})$ *includes all possible states of all the robots at time* t*,*
- $\mu = \mu_0 \mu_1 \mu_2 \dots$ *is a control policy, where* $\mu_t^i : \mathbb{R}^n \to \mathbb{R}^m$ *is the control law for the* i*th robot at time* t *and* $\mu_t = (\mu_t^0, \dots, \mu_t^{N_R})$ *is the set of control laws for all the robots at time* t*,*
- $\Lambda = \Lambda_0 \Lambda_1 \Lambda_2 \dots$ *is a sequence of sets of valuations over* Π*, where* $\Lambda_t = \{\lambda_t^i | \lambda_t^i = h_{\mathcal{X} \to \Pi^i}(x_t^i), x_t^i \in X_t^i, 1 \le i \le N_R\}$ *is the set of all possible valuations for all the possible states of all the robots at time* t*,*
- $\xi = \{\xi_0 \xi_1 \xi_2 \dots | \xi_t \in \Xi_t\}$ *is a set of sequences of valuations over* Σ *where* $\Xi_t = \{\sigma_i(\lambda_t) | \lambda_t \in \Lambda_t, \sigma_i \in \Sigma\}$ *represents the truth assignments of all the Boolean and pseudo-Boolean predicates associated with the state set* X_t *and propositions* Λ_t*.*

Robot j is considered in the ith robot's communication neighborhood at time t if there exists a line that connects i and j and does not pass through any jamming area J_k, with $1 \le k \le N_J$. However, due to disturbances, the robot states are no longer uniquely defined at each time. Instead, they can take any value within the sets X_t^i and X_t^j. Therefore, we define the set of DoS-free communication neighborhoods as follows.

Definition 3.2 (DoS-Free Multi-hop Communication Neighborhood)**.** *Given a system trajectory, the DoS-free, h-hop, communication neighborhood $C_{r,t}^i(h)$ of the ith robot at time t can be recursively defined as:*

$$\begin{aligned} C_{r,t}^i(1) = \{j | j \in \{1, \dots, N_R\}, &\mathcal{L}(h_{\mathcal{X}\to\mathcal{W}}(x_t^i), h_{\mathcal{X}\to\mathcal{W}}(x_t^j)) \notin J_k, \\ &\forall\, k \in \{1, \dots, N_J\}, \forall\, x_t^i \in X_t^i, \forall\, x_t^j \in X_t^j\}, \qquad (3.2) \\ C_{r,t}^i(h) = \{j | j \in C_{r,t}^k(1), &\forall\, k \in C_{r,t}^i(h-1)\}, h > 1. \qquad (3.3) \end{aligned}$$

Definition 3.3 (DoS-Free Base Station Communication Neighborhood)**.** *Given a system trajectory, the set of base stations $C_{b,t}^i(h)$ for which robot i can establish a DoS-free, h-hop communication at time t can be recursively defined as:*

$$C_{b,t}^i(1) = \{j | j \in \{1, \dots, N_B\}, \mathcal{L}(h_{\mathcal{X}\to\mathcal{W}}(x_t^i), B_j) \notin J_k \forall\, k \in \{1 \dots, N_J\}, \forall\, x_t^i \in X_t^i\} \qquad (3.4)$$

$$C_{b,t}^i(h) = \{j | j \in C_{b,t}^k(1), \forall\, k \in C_{r,t}^i(h-1)\}, h > 1 \qquad (3.5)$$

Definitions 3.2 and 3.3 require a communication link to be established between two robots (or a robot and the base stations) regardless of the disturbance. Moreover, because of the disturbance, there may not exist a single valuation over the atomic propositions in Σ at each time in a trajectory. We therefore require that all the possible valuations in the trajectory satisfy the LTL specification as follows.

Problem 3.1 (Centralized DoS-Resilient Motion Planning)**.** *Given a set of N_R robots whose individual dynamics are governed by (3.1), a set of N_B stationary base stations, a mission specification captured by the LTL formula ψ, synthesize a system trajectory that satisfies the following constraints:*

***Initial state constraint**:* $x_0^i = \overline{x}_0^i, \quad \forall\ i \in \{1, \ldots, N_R\}$,

***State constraints**:* $X_t^i \subseteq \mathcal{X}, \quad \forall\ t \in \mathbb{N}, \forall\ i \in \{1, \ldots, N_R\}$,

***Input constraints**:* $\mu_t^i(x_t^i) \in \mathcal{U}, \quad \forall\ x_t^i \in X_t^i, \forall\ t \in \mathbb{N}, \forall\ i \in \{1, \ldots, N_R\}$,

***Dynamics constraints**:* $f(x_t^i, \mu_t^i(x_t^i)) \oplus \Theta \subseteq X_{t+1}^i, \forall\ x_t^i \in X_t^i, \forall\ t \in \mathbb{N}, \forall\ i \in \{1, \ldots, N_R\}$,

***LTL constraints**:* $(\xi', 0) \models \psi \quad \forall\ \xi' \in \xi$,

***Collision avoidance constraints**:* $\forall\ t \in \mathbb{N}, \forall\ i, j \in \{1, \ldots, N_R\}, i \neq j$, $\|h_{\mathcal{X}\to\mathcal{W}}(x_t^i) - h_{\mathcal{X}\to\mathcal{W}}(x_t^j)\| \geq \epsilon$, *with* $\epsilon \in \mathbb{R}_{>0}$, $\forall\ x_t^i \in X_t^i, \forall\ x_t^j \in X_t^j$,

DoS resilience constraints: $\cup_{h=1}^{N_R} C_{b,t}^i(h) \neq \emptyset, \quad \forall\ t \in \mathbb{N}, \forall\ i \in \{1, \ldots, N_R\}$.

3.3 Satisfiability Modulo Convex Programming (SMC)-based Motion Planning

We resort to the Satisfiability Modulo Convex Programming (SMC) framework [132, 133, 131] to devise a motion planning algorithm that solves Problem 3.1. SMC-based motion planning is an iterative method that relies on encoding the planning problem, for a fixed horizon L, as a monotone SMC formula φ over Boolean and convex constraints, respectively capturing the mission constraints and the robot physical constraints.

As shown in Alg. 10, our motion planner consists of three steps. First, as an offline step, we generate a perturbation of the workspace by inflating the DoS jamming areas to account for the worst-case disturbance scenario. Details on how to compute the perturbed workspace and the associated guarantees are given in Sec. 3.4. Next, we translate both the LTL mission specification and the DoS-resilience constraints into a conjunction of Boolean constraints over the workspace propositions. Details on the generation of these constraints are provided in Sec. 3.5.

To solve these constraints, SMC uses an efficient Boolean satisfiability (SAT) solver to find a

Algorithm 10 SMC-Based Motion Planner

```
Input: $\mathcal{P} := \langle \mathcal{W}, J, B, \Pi, \Sigma, f, \Theta, \overline{x}_0, \mathcal{X}, \mathcal{U}, \epsilon, \psi \rangle$
Input: Disturbance rejection factor $\beta$
  Step 1: Compute the tube set and the workspace perturbation
  $(\Omega, \mu_\Omega) :=$ Compute-RCI$(f, \Theta, \mathcal{X}, \beta\mathcal{U})$
  $(J^*, \mathcal{W}^*_{\psi,-}, \mathcal{W}^*_{\psi,+}) :=$ Perturb$(J, \Omega, \mathcal{W})$
  $(\mathcal{W}^*, Adj_p, Adj_c) :=$Partition$(\mathcal{W}, J^*, \mathcal{W}^*_{\psi,-}, \mathcal{W}^*_{\psi,+})$

  Step 2: Use SMC to plan the nominal trajectory
  Initialize horizon:    $L := 1$;
  while Trajectory is not found do
    $|[\mathcal{P}, L]|_D :=$ Encode-Dis-Plan($\mathcal{W}^*$,$B$, $\Pi$, $\Sigma$, $Adj_p$, $Adj_c$, $\psi$, $L$)
    $|[\mathcal{P}, L]|_C :=$ Encode-Con-Plan($\mathcal{W}^*$, $f$, $\overline{x}_0$, $\mathcal{X}$, $(1-\beta)\mathcal{U}$, $\epsilon$, $L$)
    (Status$, z, v) :=$ SMC.Solve$(|[\mathcal{P}, L]|_D, |[\mathcal{P}, L]|_C)$;
    if Status == UNSAT then
      Increase horizon:    $L := L + 1$;
    end if
  end while
  Step 3: Trajectory Tracking
  At each time step, apply the input $u_t = v_t + \mu_\Omega(x_t - z_t)$
```

candidate sequence of workspace regions that satisfies the mission and DoS constraints while ignoring the robot dynamics, input, and state constraints. A convex solver is then used to check the feasibility of the candidate path. If both the Boolean and the convex constraints are satisfied, a valid trajectory is returned, consisting of the proposed plan and the corresponding *nominal* state and control input trajectories. Otherwise, the proposed high-level sequence is marked as infeasible and new candidate plans are generated until either a feasible one is found, or no trajectory is feasible for the current horizon length L. A prominent feature of SMC is the generation of compact infeasibility certificates, i.e., "succinct explanations" that can capture the root causes for the infeasibility of a plan and rule out the largest possible number of invalid plans for the SAT solver to accelerate the search. This iterative procedure was shown to be more than two orders of magnitude faster than state-of-the-art sampling based techniques for high-dimensional state spaces [131].

Finally, we compute a feedback control law that can track the *nominal* trajectory generated using the SMC approach and the perturbed workspace. This control law will be used to address disturbances during system operation. Details on the computation of the feedback law are provided in Sec. 3.6.

3.4 Robust Controlled Invariant Sets and Workspace Perturbation

Given the robot dynamics (3.1), a feedback controller that rejects the disturbance θ and forces the trajectories governed by (3.1) to evolve inside the state constraint set $\mathcal{X}$ can be characterized by the notion of robust controlled invariant set contained inside $\mathcal{X}$ [17]. A set $\Omega \subseteq \mathcal{X}$ is a *robust controlled invariant (RCI) set* for the system (3.1) if there exists a feedback controller $\mu_\Omega : \Omega \to \mathcal{U}$ such that, for every $x_t \in \Omega$, the following holds:

$$f(x_t, \mu_\Omega(x_t)) + \theta_t \in \Omega, \qquad \forall\, \theta_t \in \Theta, \forall\, t \in \mathbb{N}.$$

In other words, if the system state starts in Ω, then it will stay in Ω in spite of the disturbance. Moreover, when f is piecewise affine, we can effectively separate the goals of disturbance rejection and trajectory planning [121].

Given a design parameter $\beta \in [0, 1[$, a disturbance-free state trajectory $z_0, z_1, \ldots$, and an open-loop control trajectory $v_0, v_1, \ldots$ such that, for all t, $z_{t+1} = f(z_t, v_t)$ and $v_t \in (1-\beta)\mathcal{U}$, we can find a robust controlled invariant set Ω_β and a corresponding feedback law μ_{Ω_β} that ensure $\mu_{\Omega_\beta}(x_t) \in \beta\mathcal{U}$ and $x_t \in z_t \oplus \Omega_\beta$ for all t. In other words, the RCI set Ω can be regarded as a "tube," regulated by μ_{Ω_β}, around a nominal (disturbance-free) trajectory $z_0, z_1, \ldots$, determined by $v_0, v_1, \ldots$. In what follows, we will restrict our attention to piecewise affine robot dynamics for which algorithms that synthesize polytopic RCI sets are already available in the literature [119, 112, 121]. For simplicity, we also drop the subscript β from the RCI notation.

In our case, rejecting disturbances translates into designing a tube that lies entirely in the jamming-free region of the workspace. We call such a tube an *Ω-perturbation (inflation)* of the nominal trajectory. We then observe that computing an Ω-perturbed trajectory that lies in the jamming-free space can be rather translated into the problem of computing a nominal (ideal) trajectory that lies in a modified space in which the jamming area and the workspace regions are, instead, perturbed. To derive this perturbation of the space, we proceed as follows.

Given the LTL formula ψ, we denote by $\mathcal{W}_{\psi,+}$ the set of (jamming-free) workspace regions whose corresponding atomic propositions appear asserted (without negation) in ψ, and by $\mathcal{W}_{\psi,-}$ the set of workspace regions whose corresponding atomic propositions are negated in ψ. We assume that a region can be either asserted or negated in ψ, and therefore $\mathcal{W}_{\psi,+}$ and $\mathcal{W}_{\psi,-}$ are disjoint sets. We then "inflate" by Ω the jamming areas and the workspace regions $\mathcal{W}_{\psi}^{-}$ that need to be avoided, and "shrink" by Ω the workspace regions $\mathcal{W}_{\psi}^{+}$ which must be traversed. Formally, we obtain:

$$J^* = \{J_k \oplus \Omega \mid k \in \{1, \ldots, N_J\}\}$$

$$\mathcal{W}_{\psi,-}^* = \{\mathcal{W}' \oplus \Omega \mid \mathcal{W}' \in \mathcal{W}_{\psi,-}\} \qquad \mathcal{W}_{\psi,+}^* = \{\mathcal{W}' \ominus \Omega \mid \mathcal{W}' \in \mathcal{W}_{\psi,+}\}.$$

3.5 Synthesis of DoS-Free Nominal Trajectories

As pictorially shown in Fig. 3.1, we start by over-approximating the Ω-perturbed jamming areas J_k^* using a set of polyhedra, which originates a coarse, multi-resolution, discretization of the free space. Unlike grid-based methods, where the workspace is discretized using a grid (or mesh) of (small) uniform resolution, the coarse-grained abstraction used in this chapter avoids state explosion. This decomposition procedure is similar to the ones previously proposed for triangular [11] or polygonal [36] representations. We denote by $\mathcal{W}_1^*, \mathcal{W}_2^*, \ldots \mathcal{W}_{r^*}^*$

the set of regions obtained after discretization, r^* being the total number of regions.

Based on this partition, we compute two adjacency functions, denoted by Adj_p and Adj_c that correspond, respectively, to the physical adjacency and communication adjacency relations between the regions. In particular, two regions $\mathcal{W}_i^*$ and $\mathcal{W}_j^*$ are said to be physically adjacent, written $Adj_p(\mathcal{W}_i^*, \mathcal{W}_j^*) = 1$, if the polyhedra $\mathcal{W}_i^*$ and $\mathcal{W}_j^*$ share one facet; otherwise, we write $Adj_p(\mathcal{W}_i^*, \mathcal{W}_j^*) = 0$. Similarly, $\mathcal{W}_i^*$ and $\mathcal{W}_j^*$ are communication adjacent if we can connect any point of $\mathcal{W}_i^*$ with any point of $\mathcal{W}_j^*$ without passing through a jamming area, that is,

$$Adj_c(\mathcal{W}_i^*, \mathcal{W}_j^*) = \begin{cases} 1 & \text{if } \mathcal{L}(w_i, w_j) \not\in J_k^*, \quad \forall\, k \in \{1, \ldots, N_J\}, \forall\, (w_i, w_j) \in \mathcal{W}_i^* \times \mathcal{W}_j^* \\ 0 & \text{otherwise} \end{cases}$$

We use these notions of adjacency to encode the mission and DoS-resilience constraints as follows.

3.5.1 Encoding Mission and DoS-Resilience Constraints

For each robot, region, and time, we introduce a Boolean variable $\pi_{j,t}^i$ which evaluates to one if and only if robot i is in region $\mathcal{W}_j^*$ at time t. Similarly, for each base station and region, we introduce a Boolean variable κ_j^i which evaluates to one if and only if base station i is in region $\mathcal{W}_j^*$, since base stations are stationary and their locations do not change with time. We use these decision variables along with the physical adjacency function Adj_p to translate the high-level, discrete planning constraints into a conjunction of Boolean constraints using the Bounded Model Checking (BMC) encoding technique for LTL model checking. We refer the readers to the literature [20] for details on the Boolean encoding of LTL specifications. In the remainder of this section, we report the encoding of the communication constraints.

We introduce a set of Boolean variables of the form $r_t(i, j, h)$, each evaluating to one whenever robot i can establish an h-hop DoS-free communication with robot j, and zero otherwise. Similarly, a Boolean variable $b_t(i,j,h)$ evaluates to one whenever robot i can establish an h-hop DoS-free communication with base station j. We then capture the communication constraints as follows.

Adjacency Constraints. We encode single-hop communication adjacency as the conjunction of the following constraints:

$$\forall t \in \{0,\dots,L\}, \forall i,j \in \{1,\dots,N_R\}\colon r_t(i,j,1) \leftrightarrow \bigvee_{k=1}^{r^*} \left(\pi^i_{k,t} \wedge \left(\bigvee_{k' \in \mathcal{N}_c(i)} \pi^j_{k',t} \right) \right), \tag{3.6}$$

where $\mathcal{N}_c(i) = \{j \in \{1,\dots r^*\} | Adj_c(\mathcal{W}^*_i, \mathcal{W}^*_j) = 1\}$ is the set of indexes marking the regions that are communication adjacent (neighbors) to region $\mathcal{W}^*_i$. Similarly, for the base stations, we obtain

$$\begin{aligned} &\forall t \in \{0,\dots,L\}, \forall i \in \{1,\dots,N_R\}, \forall j \in \{1,\dots,N_B\} : \\ &\quad b_t(i,j,1) \leftrightarrow \bigvee_{k=1}^{r^*} \left(\kappa^i_k \wedge \left(\bigvee_{k' \in \mathcal{N}_c(i)} \pi^j_{k',t} \right) \right). \end{aligned} \tag{3.7}$$

Transitivity Constraints. To encode multi-hop communication, we generate the following constraints:

$$\begin{aligned} &\forall t \in \{0,\dots,L\}, \forall i,j \in \{1,\dots,N_R\}, \forall h \in \{1,\dots,N_R\} : \\ &\bigvee_{k=1}^{N_R} \bigvee_{\substack{h_1,h_2 \in \{1,\dots,N_R\} \\ h_1+h_2=h}} \left(r_t(i,k,h_1) \wedge r_t(k,j,h_2) \right) \leftrightarrow r_t(i,j,h) \end{aligned} \tag{3.8}$$

and conjoin them with the following ones:

$$\forall t \in \{0,\dots,L\}, \forall i \in \{1,\dots,N_R\}, \forall j \in \{1,\dots,N_B\}, \forall h \in \{1,\dots,N_R\}:$$
$$\bigvee_{k=1}^{N_R} \bigvee_{\substack{h_1,h_2 \in \{1,\dots,N_R\} \\ h_1+h_2=h}} (r_t(i,k,h_1) \wedge b_t(k,j,h_2)) \leftrightarrow b_t(i,j,h). \tag{3.9}$$

DoS-Resilience Constraints. Finally, the constraints below ensure that each robot is connected with at least one base station, by either a single or multi-hop communication link:

$$\forall i \in \{1,\dots,N_R\}: \qquad \bigwedge_{t=0}^{L} \bigvee_{j=1}^{N_B} \bigvee_{h=1}^{N_R} b_t(i,j,h). \tag{3.10}$$

3.5.2 Nominal Trajectory Planning

As discussed in Sec. 4.5, we use a SAT solver to find a high-level, candidate sequence of regions that satisfy the Boolean formula encoding the LTL specification and the DoS-resilience constraints. It is possible to represent this trajectory, which is infinite in general, with a finite sequence of the form $\rho = (\rho_0\rho_1 \dots \rho_{k-1})(\rho_k \dots \rho_L)^\omega$, consisting of a prefix $\rho_0\rho_1 \dots \rho_{k-1}$ and a loop sequence $\rho_k \dots \rho_L$ that repeats indefinitely, as denoted by the superscript ω [20].

Given the system piecewise affine dynamics f, the state and control constraint sets $\mathcal{X}$ and $(1-\beta)\mathcal{U}$, the robot initial state $\overline{x}_0$, the high-level candidate path ρ, the margin ϵ for collision avoidance, and the Ω-perturbed workspace regions associated with ρ, checking the feasibility of the candidate path, generating the nominal state trajectory $z_0^i, z_1^i, \dots$ for each robot, or providing succinct infeasibility certificates, whenever such state trajectories do not exist, can all be cast as convex programs [132, 131].

3.6 Tracking of the Nominal Trajectory

The final step is to compute the control law for tracking the nominal trajectory $z_0, z_1, \ldots$ by summing the nominal open-loop control input v_t and the feedback control law $\mu_\Omega(x_t - z_t)$ for all $t \in \mathbb{N}$. Algorithm 10 summarizes the proposed SMC-based robust motion planning procedure. Its correctness guarantees are stated below.

Theorem 3.2 (Correctness of Algorithm 10)**.** *Algorithm 10 is sound, that is, all trajectories resulting from its execution are solutions of Problem 3.1.*

Proof Sketch. Soundness of Alg. 10 directly follows from the separation between disturbance rejection and trajectory planning (see, e.g., [121, Theorem 5.4]), the soundness of the SMC-based motion planning algorithm (for the nominal trajectory) [131, Theorem 4.2], the construction of the perturbed sets $J^*, \mathcal{W}^*_{\psi,-}, \mathcal{W}^*_{\psi,+}$, and the soundness of the DoS-resilience encoding in (3.10), i.e., the fact that, if (3.10) holds, then there exists a communication path between each robot and at least one of the base stations at each time. $\square$

3.7 Results

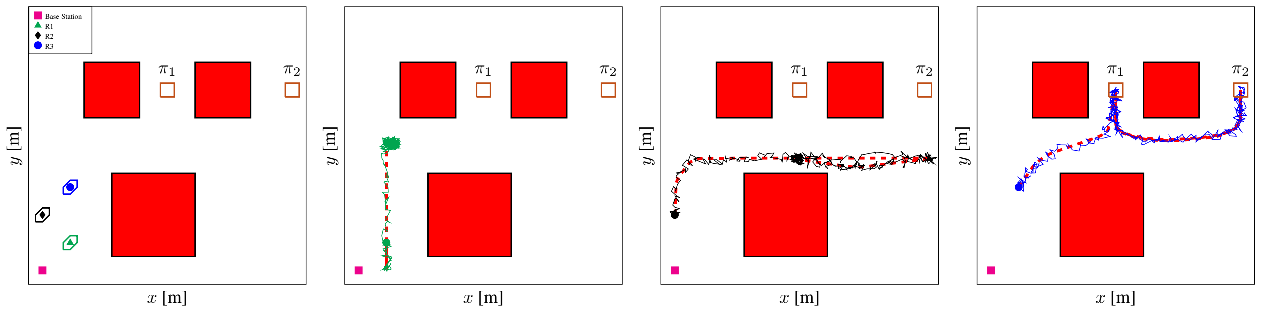


Figure 3.2: Workspace showing the initial position of the robots, the base stations, and the jamming areas (red boxes) along with the three trajectories subject to $(\Box\Diamond(\pi_1^3 = 1)) \wedge (\Box\Diamond(\pi_2^1 + \pi_2^2 + \pi_2^3 = 1))$. Actual trajectories (green for $R1$, black for $R2$, and blue for $R3$) are plotted on top of the nominal trajectories (dashed red).

We implemented Alg. 10 in Python on top of the SatEX solver [133], using Z3 [35] as a SAT solver and Cplex [64] as a convex optimization solver. All the experiments were executed on an Intel Core i7 2.3-GHz processor with 16 GB of memory.

To illustrate the capabilities of our algorithm in a multi-robot scenario under generic LTL specifications, we consider a team of 3 robots, $R1$, $R2$, $R3$, and one base station operating in the workspace represented in Fig. 3.2 (top left). We assume robot dynamics captured by chains of integrators, one chain for each coordinate of the workspace, and a sampling time of 0.5 s. The upper bound on the disturbance is 0.2 m on the robot position (coordinates) and zero on the higher-order states. Red boxes denote the DoS areas of the three jamming radars. Initial positions are shown in Fig. 3.2 (top left). The mission is specified by the LTL formula $\psi := (\Box\Diamond(\pi_1^3 = 1)) \wedge (\Box\Diamond(\pi_2^1 + \pi_2^2 + \pi_2^3 = 1))$ which requires that $R3$ visit region π_1 infinitely often, and that any of the robots visit location π_2 infinitely often.

Figure 3.2 shows the nominal trajectories $z_0^i z_1^i z_2^i \ldots$, $i \in \{1, 2, 3\}$, for the double integrator case (dashed red lines) along with the actual robots' trajectories $x_0^i x_1^i x_2^i \ldots$ (green for $R1$, black for $R2$, and blue for $R3$) for a realization of the disturbance from a random uniform distribution over the set of admissible disturbances. Figure 3.3 reports snapshots of the three robots at different times along with the nominal trajectories and the corresponding RCI sets. The planner strategically positions $R1$ to guarantee communication with the base station at all times. As $R3$ approaches the first goal, $R2$ is positioned to operate as an intermediate hub between $R3$ and $R1$, thus creating a 2-hop link between the base station and $R3$. Similarly, when $R3$ reaches the second goal, $R2$ is also moved to provide the necessary communication path for $R3$. The overall computation of the RCI set, the nominal trajectory, and the feedback law took around 3 s, 534 s, and 20 ms, respectively.

Table 3.1 reports the execution time of the three steps in Alg. 10 for a basic reach-avoid specification as the number of robots and the number of integrators (per robot) in the chain, hence the number of state variables, increase in the presence of one and two base stations.

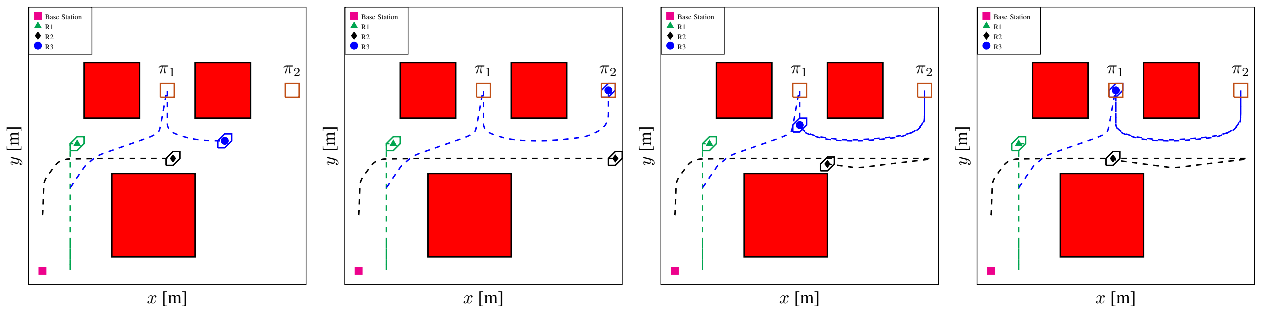


Figure 3.3: Snapshots of the nominal trajectories and the corresponding RCI sets, subject to $(\Box\Diamond(\pi_1^3 = 1)) \wedge (\Box\Diamond(\pi_2^1 + \pi_2^2 + \pi_2^3 = 1))$.

Table 3.1: Execution time for the workspace in Fig. 3.2.

# robots	# states	RCI [s]	One Base Station			Two Base Stations		
			# Bool variables	SMC [s]	μ_Ω [ms]	# Bool variables	SMC [s]	μ_Ω [ms]
2	4	2.878	36	92.99	10.2	108	175.64	11.5
	6	3.265	42	223.04	13.4	126	199.81	13.6
	8	3.780	42	88.98	18.5	126	1175.96	18.8
3	4	2.878	108	210.06	10.2	240	411.73	11.5
	6	3.265	126	347.89	13.4	280	474.17	13.6
	8	3.780	126	818.69	18.5	280	1328.92	18.8
4	4	2.878	240	565.31	10.2	450	647.16	11.5
	6	3.265	280	645.01	13.4	525	2685.97	13.6
	8	3.780	280	1597.51	18.5	525	2373.67	18.8

The table also reports the number of Boolean variables needed to encode the DoS-resilience constraints.

Part II

Neural Network Verification and Architecture Design

Chapter 4

Formal Verification of Neural Network Controlled Autonomous Systems

In this chapter, we consider the problem of formally verifying the safety of an autonomous robot equipped with a Neural Network (NN) controller that processes LiDAR images to produce control actions. Given a workspace that is characterized by a set of polytopic obstacles, our objective is to compute the set of safe initial states such that a robot trajectory starting from these initial states is guaranteed to avoid the obstacles. Our approach is to construct a finite state abstraction of the system and use standard reachability analysis over the finite state abstraction to compute the set of safe initial states. To mathematically model the imaging function, that maps the robot position to the LiDAR image, we introduce the notion of imaging-adapted partitions of the workspace in which the imaging function is guaranteed to be affine. Given this workspace partitioning, a discrete-time linear dynamics of the robot, and a pre-trained NN controller with Rectified Linear Unit (ReLU) nonlinearity, we utilize a Satisfiability Modulo Convex (SMC) encoding to enumerate all the possible assignments of different ReLUs. To accelerate this process, we develop a pre-processing algorithm that could rapidly prune the space of feasible ReLU assignments. Finally, we demonstrate the efficiency of the proposed algorithms using numerical simulations with the increasing complexity of the neural network controller.

4.1 Introduction

From simple logical constructs to complex deep neural network models, Artificial Intelligence (AI)-agents are increasingly controlling physical/mechanical systems. Self-driving cars, drones, and smart cities are just examples of such systems to name a few. However, regardless of the explosion in the use of AI within a multitude of cyber-physical systems (CPS) domains, the safety and reliability of these AI-enabled CPS is still an under-studied problem. It is then unsurprising that the failure of these AI-controlled CPS in several, safety-critical, situations leads to human fatalities [164].

Motivated by the urgency to study the safety, reliability, and potential problems that can rise and impact the society by the deployment of AI-enabled systems in the real world, several works in the literature focused on the problem of designing deep neural networks that are robust to the so-called adversarial examples [46, 41, 27, 138, 97, 100, 118]. Unfortunately, these techniques focus mainly on the robustness of the learning algorithm with respect to data outliers without providing guarantees in terms of safety and reliability of the decisions made by these neural networks. To circumvent this drawback, recent works focused on three main techniques namely (i) testing of neural networks, (ii) falsification (semi-formal verification) of neural networks, and (iii) formal verification of neural networks.

Representatives of the first class, namely testing of neural networks, are the works reported in [104, 151, 163, 145, 89, 159, 90, 137, 175, 146] in which the neural network is treated as a white box, and test cases are generated to maximize different coverage criteria. Such coverage criteria include neuron coverage, condition/decision coverage, and multi-granularity testing criteria. On the one hand, maximizing test coverage gives system designers confidence that the networks are reasonably free from defect. On the other hand, testing does not formally guarantee that a neural network satisfies a formal specification.

To take into consideration the effect of the neural network decisions on the entire system behavior, several researchers focused on the falsification (or semi-formal verification) of autonomous systems that include machine learning components [37, 153, 176]. In such falsification frameworks, the objective is to generate corner test cases that forces a violation of system-level specifications. To that end, advanced 3D models and image environments are used to bridge the gap between the virtual world and the real world. By parametrizing the input to these 3D models (e.g., position of objects, position of light sources, intensity of light sources) and sampling the parameter space in a fashion that maximizes the falsification of the safety property, falsification frameworks can simulate several test cases until a counterexample is found [37, 153, 176].

While testing and falsification frameworks are powerful tools to find corner cases in which the neural network or the neural network enabled system may fail, they lack the rigor promised by formal verification methods. Therefore, several researchers pointed to the urgent need of using formal methods to verify the behavior of neural networks and neural network enabled systems [79, 129, 128, 84, 85, 126]. As a result, recent works in the literature attempted the problem of applying formal verification techniques to neural network models.

Applying formal verification to neural network models comes with its unique challenges. First and foremost is the lack of widely-accepted, precise, mathematical specifications capturing the correct behavior of a neural network. Therefore, recent works focused entirely on verifying neural networks against simple input-output specifications [68, 39, 23, 117, 38, 108]. Such input-output techniques compute a guaranteed range for the output of a deep neural network given a set of inputs represented as a convex polyhedron. To that end, several algorithms that exploit the piecewise linear nature of the Rectified Linear Unit (ReLU) activation functions (one of the most famous nonlinear activation functions in deep neural networks) have been proposed. For example, by using binary variables to encode piecewise linear functions, the constraints of ReLU functions are encoded as a Mixed-Integer Linear Programming (MILP). Combining output specifications that are expressed in terms of Linear Programming (LP), the verification problem eventually turns to a MILP feasibility problem [38, 152].

Using off-the-shelf MILP solvers does not lead to scalable approaches to handle neural networks with hundreds and thousands of neurons [39]. To circumvent this problem, several MILP-like solvers targeted toward the neural network verification problem are proposed. For example, the work reported in [68] proposed a modified Simplex algorithm (originally used to solve linear programs) to take into account ReLU nonlinearities as well. Similarly, the work reported in [39] combines a Boolean satisfiability solving along with a linear over-approximation of piecewise linear functions to verify ReLU neural networks against convex specifications. Other techniques that exploit specific geometric structures of the specifications are also proposed [55, 169]. A thorough survey on different algorithms for verification of neural networks against input-output range specifications can be found in [168] and the references within.

Unfortunately, the input-output range properties are simplistic and fail to capture the safety and reliability of cyber-physical systems when controlled by a neural network. Recent works showed how to perform reachability-based verification of closed-loop systems in the presence of learning components [166, 65, 3]. Reachability analysis is performed by either separately estimating the output set of the neural network and the reachable set of continuous dynamics [166], or by translating the neural network controlled system into a hybrid system [65]. Once the neural network controlled system is translated into a hybrid system, off-the-shelf existing verification tools of hybrid systems, such as SpaceEx [53] for piecewise affine dynamics and Flow* [29] for nonlinear dynamics, can be used to verify safety properties of the system. Another related technique is the safety verification using barrier certificates [154].

In such approach, a barrier function is searched using several simulation traces to provide a certificate that unsafe states are not reachable from a given set of initial states.

Differently from the previous work—in the literature of formal verification of neural network controlled system—we consider, in this chapter, the case in which the robotic system is equipped with a LiDAR scanner that is used to sense the environment [140]. The LiDAR image is then processed by a neural network controller to compute the control inputs. Arguably, the ability of neural networks to process high-bandwidth sensory signals (e.g., cameras and LiDARs) is one of the main motivations behind the current explosion in the use of machine learning in robotics and CPS. Towards this goal, we develop a framework that can reason about the safety of the system while taking into account the robot continuous dynamics, the workspace configuration, the LiDAR imaging, and the neural network.

In particular, the contributions of this chapter can be summarized as follows:

1- A framework for formally proving safety properties of autonomous robots equipped with LiDAR scanners and controlled by neural network controllers.

2- A notion of imaging-adapted partitions along with a polynomial-time algorithm for processing the workspace into such partitions. This notion of imaging-adapted partitions plays a significant role in capturing the LiDAR imaging process.

3- A Satisfiability Modulo Convex (SMC)-based algorithm combined with an SMC-based pre-processing for computing finite abstractions of neural network controlled autonomous systems.

4.2 Problem Formulation

The symbols $\mathbb{N}$, $\mathbb{R}, \mathbb{R}^+$ and $\mathbb{B}$ denote the set of natural, real, positive real, and Boolean numbers, respectively. The symbols $\wedge, \neg$ and $\rightarrow$ denote the logical AND, logical NOT, and logical IMPLIES operators, respectively. Given two real-valued vectors $x_1 \in \mathbb{R}^{n_1}$ and $x_2 \in \mathbb{R}^{n_2}$, we denote by $(x_1, x_2) \in \mathbb{R}^{n_1+n_2}$ the column vector $[x_1^T, x_2^T]^T$. Similarly, for a vector $x \in \mathbb{R}^n$, we denote by $x_i \in \mathbb{R}$ the ith element of x. For two vectors $x_1, x_2 \in \mathbb{R}^n$, we denote by $\max(x_1, x_2)$ the element-wise maximum. For a set $S \subset \mathbb{R}^n$, we denote the boundary and the interior of this set by ∂S and $\text{int}(S)$, respectively. Given two sets S_1 and S_2, $f : S_1 \rightrightarrows S_2$ and $f : S_1 \rightarrow S_2$ denote a set-valued and ordinary map, respectively. Finally, given a vector $z = (x, y) \in \mathbb{R}^2$, we denote by $\text{atan2}(z) = \text{atan2}(y, x)$.

4.2.1 Dynamics and Workspace

We consider an autonomous robot moving in a 2-dimensional polytopic (compact and convex) workspace $\mathcal{W} \subset \mathbb{R}^2$. We assume that the robot must avoid the workspace boundaries ∂W along with a set of obstacles $\{\mathcal{O}_1, \ldots, \mathcal{O}_o\}$, with $\mathcal{O}_i \subset \mathcal{W}$ which is assumed to be polytopic. We denote by $\mathcal{O}$ the set of the obstacles and the workspace boundaries which needs to be avoided, i.e., $\mathcal{O} = \{\partial W, \mathcal{O}_1, \ldots, \mathcal{O}_o\}$. The dynamics of the robot is described by a discrete-time linear system of the form:

$$x^{(t+1)} = Ax^{(t)} + Bu^{(t)}, \tag{4.1}$$

where $x^{(t)} \in \mathcal{X} \subseteq \mathbb{R}^n$ is the state of robot at time $t \in \mathbb{N}$ and $u^{(t)} \subseteq \mathbb{R}^m$ is the robot input. The matrices A and B represent the robot dynamics and have appropriate dimensions. For a robot with nonlinear dynamics that is either differentially flat or feedback linearizable, the state space model (4.1) corresponds to its feedback linearized dynamics. We denote by

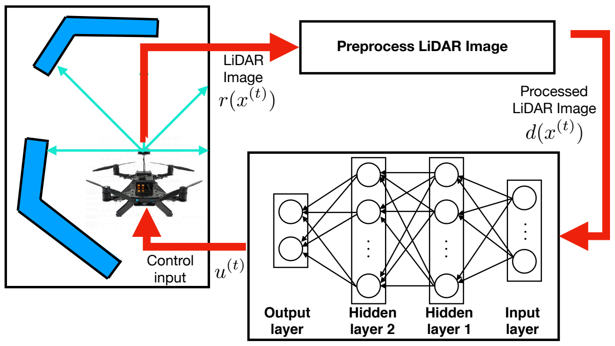


Figure 4.1: Pictorial representation of the problem setup under consideration.

$\zeta(x) \in \mathbb{R}^2$ the natural projection of x onto the workspace $\mathcal{W}$, i.e., $\zeta(x^{(t)})$ is the position of the robot at time t.

4.2.2 LiDAR Imaging

We consider the case when the autonomous robot uses a LiDAR scanner to sense its environment. The LiDAR scanner emits a set of N lasers evenly distributed in a 2π degree fan. We denote by $\theta_{\text{lidar}}^{(t)} \in \mathbb{R}$ the heading angle of the LiDAR at time t. Similarly, we denote by $\theta_i^{(t)} = \theta_{\text{lidar}}^{(t)} + (i-1)\frac{2\pi}{N}$, with $i \in \{1, \ldots, N\}$, the angle of the ith laser beam at time t where $\theta_1^{(t)} = \theta_{\text{lidar}}^{(t)}$ and by $\theta^{(t)} = (\theta_1^{(t)}, \ldots, \theta_N^{(t)})$ the vector of the angles of all the laser beams. While the heading angle of the LiDAR, $\theta_{\text{lidar}}^{(t)}$, changes as the robot pose changes over time, i.e., $\theta_{\text{lidar}}^{(t)} = f(x^{(t)})$ for some nonlinear function f, in this chapter we focus on the case when the heading angle of the LiDAR, $\theta_{\text{lidar}}^{(t)}$, is fixed over time and we will drop the superscript t from the notation. Such condition is satisfied in several real-world scenarios whenever the robot is moving while maintaining a fixed pose (e.g. a quadrotor whose yaw angle is maintained constant).

For the ith laser beam, the observation signal $r_i(x^{(t)}) \in \mathbb{R}$ is the distance measured between the robot position $\zeta(x^{(t)})$ and the nearest obstacle in the θ_i direction, i.e.:

$$r_i(x^{(t)}) = \min_{\mathcal{O}_i \in \mathcal{O}} \min_{z \in \mathcal{O}_i} \|z - \zeta(x^{(t)})\|_2 \qquad \text{s.t.} \quad \text{atan2}\left(z - \zeta(x^{(t)})\right) = \theta_i. \tag{4.2}$$

In this chapter, we will restrict our attention to the case when the LiDAR scanner is ideal (with no noise) although the bounded noise case can be incorporated in the proposed framework. The final LiDAR image $d(x^{(t)}) \in \mathbb{R}^{2N}$ is generated by processing the observations $r(x^{(t)})$ as follows:

$$d_i(x^{(t)}) = \left(r_i(x^{(t)}) \cos\theta_i,\ r_i(x^{(t)}) \sin\theta_i\right), \qquad d(x^{(t)}) \quad = \left(d_1(x^{(t)}), \ldots d_N(x^{(t)})\right). \tag{4.3}$$

4.2.3 Neural Network Controller

We consider a pre-trained neural network controller $f_{\text{NN}} : \mathbb{R}^{2N} \to \mathbb{R}^m$ that processes the LiDAR images to produce control actions with L internal and fully connected layers in addition to one output layer. Each layer contains a set of M_l neurons (where $l \in \{1, \ldots, L\}$) with Rectified Linear Unit (ReLU) activation functions. ReLU activation functions play an important role in the current advances in deep neural networks [77]. For such neural network architecture, the neural network controller $u^{(t)} = f_{\text{NN}}(d(x^{(t)}))$ can be written as:

$$\begin{aligned}
h^{1(t)} &= \max\left(0,\ W^0 d(x^{(t)}) + w^0\right), \\
h^{2(t)} &= \max\left(0,\ W^1 h^{1(t)} + w^1\right), \\
&\ \ \vdots \\
h^{L(t)} &= max\left(0,\ W^{L-1} h^{L-1(t)} + w^{L-1}\right), \\
u^{(t)} &= W^L h^{L(t)} + w^L,
\end{aligned} \tag{4.4}$$

where $W^l \in \mathbb{R}^{M_l \times M_{l-1}}$ and $w^l \in \mathbb{R}^{M_l}$ are the pre-trained weights and bias vectors of the neural network which are determined during the training phase.

4.2.4 Robot Trajectories and Safety Specifications

The trajectories of the robot whose dynamics are described by (4.1) when controlled by the neural network controller (4.2)-(4.4) starting from the initial condition $x_0 = x^{(0)}$ is denoted by $\eta_{x_0} : \mathbb{N} \to \mathbb{R}^n$ such that $\eta_{x_0}(0) = x_0$. A trajectory η_{x_0} is said to be safe whenever the robot position does not collide with any of the obstacles at all times.

Definition 4.1 (Safe Trajectory). *A robot trajectory η_{x_0} is called safe if $\zeta(\eta_{x_0}(t)) \in \mathcal{W},\ \zeta(\eta_{x_0}(t)) \notin \mathcal{O}_i,\ \forall \mathcal{O}_i \in \mathcal{O},\ \forall t \in \mathbb{N}$.*

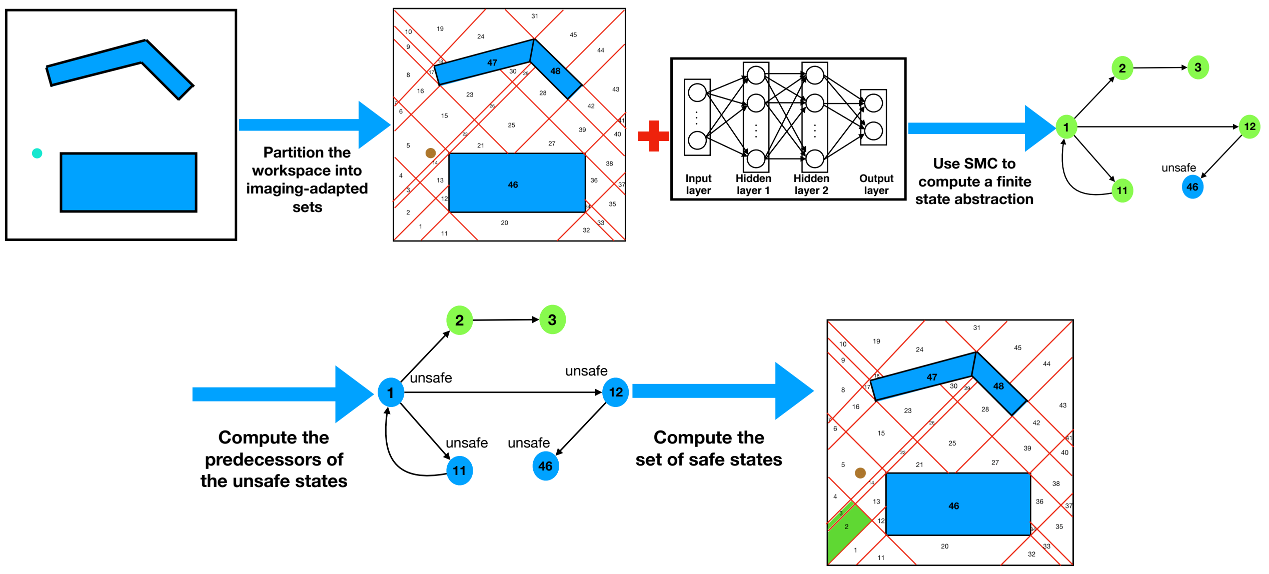
Using the previous definition, we now define the problem of verifying the system-level safety of the neural network controlled system as follows:

Problem 4.1. *Consider the autonomous robot whose dynamics are governed by* (4.1) *which is controlled by the neural network controller described by* (4.4) *which processes LiDAR images described by* (4.2)-(4.3)*. Compute the set of safe initial conditions $\mathcal{X}_{safe} \subseteq \mathcal{X}$ such that any trajectory η_{x_0} starting from $x_0 \in \mathcal{X}_{safe}$ is safe.*

4.3 Framework

Before we describe the proposed framework, we need to briefly recall the following definitions capturing the notion of a system and relations between different systems.

Definition 4.2. *An autonomous system $\mathcal{S}$ is a pair (X, δ) consisting of a set of states X and a set-valued map $\delta : X \rightrightarrows X$ representing the transition function. A system $\mathcal{S}$ is finite*



Partition the workspace into imaging-adapted sets
Input layer
Hidden layer 1
Hidden layer 2
Output layer
Use SMC to compute a finite state abstraction
unsafe
Compute the predecessors of the unsafe states
unsafe
unsafe
unsafe
unsafe
Compute the set of safe states

Figure 4.2: Pictorial representation of the proposed framework.

if X is finite. A system S is deterministic if δ is single-valued map and is non-deterministic if not deterministic.

Definition 4.3. *Consider a deterministic system $\mathcal{S}_a = (X_a, \delta_a)$ and a non-deterministic system $\mathcal{S}_b = (X_b, \delta_b)$. A relation $Q \subseteq X_a \times X_b$ is a simulation relation from $\mathcal{S}_a$ to $\mathcal{S}_b$, and we write $\mathcal{S}_a \preccurlyeq_Q \mathcal{S}_b$, if the following conditions are satisfied:*

1. *for every $x_a \in X_a$ there exists $x_b \in X_b$ with $(x_a, x_b) \in Q$,*
2. *for every $(x_a, x_b) \in Q$ we have that $x'_a = \delta_a(x_a)$ in $\mathcal{S}_a$ implies the existence of $x'_b \in \delta_b(x_b)$ in $\mathcal{S}_b$ satisfying $(x'_a, x'_b) \in Q$.*

Using the previous two definitions, we describe our approach as follows. As pictorially shown in Figure 4.2, given the autonomous robot system $\mathcal{S}_{\text{NN}} = (\mathcal{X}, \delta_{\text{NN}})$, where $\delta_{\text{NN}} : x \mapsto Ax + Bf_{\text{NN}}(d(x))$, our objective is to compute a finite state abstraction (possibly non-deterministic) $\mathcal{S}_{\mathcal{F}} = (\mathcal{F}, \delta_{\mathcal{F}})$ of $\mathcal{S}_{\text{NN}}$ such that there exists a simulation relation from $\mathcal{S}_{\text{NN}}$ to $\mathcal{S}_{\mathcal{F}}$, i.e., $\mathcal{S}_{\text{NN}} \preccurlyeq_Q \mathcal{S}_{\mathcal{F}}$. This finite state abstraction $\mathcal{S}_{\mathcal{F}}$ will be then used to check the safety specification.

The first difficulty in computing the finite state abstraction $\mathcal{S}_{\mathcal{F}}$ is the nonlinearity in the relation between the robot position $\zeta(x)$ and the LiDAR observations as captured by equation (4.2). However, we notice that we can partition the workspace based on the laser angles $\theta_1, \ldots, \theta_N$ along with the vertices of the polytopic obstacles such that the map d (defined in equation (4.3) which maps the robot position to the processed observations) is an affine map as shown in Section 4.4. Therefore, as summarized in Algorithm 11, the first step is to compute such partitioning $\mathcal{W}^\star$ of the workspace (WKSP-PARTITION, line 2 in Algorithm 11). While WKSP-PARTITION focuses on partitioning the workspace $\mathcal{W}$, one needs to partition the remainder of the state space $\mathcal{X}$ (STATE-SPACE-PARTITION, line 5 in Algorithm 11) to compute the finite set of abstract states $\mathcal{F}$ along with the simulation relation Q that maps between states in $\mathcal{X}$ and the corresponding abstract states in $\mathcal{F}$, and vice versa.

Unfortunately, the number of partitions grows exponentially in the number of lasers N and the number of vertices of the polytopic obstacles. To harness this exponential growth, we compute an aggregate-partitioning $\mathcal{W}'$ using only a few laser angles (called primary lasers and denoted by θ_p). The resulting aggregate-partitioning $\mathcal{W}'$ would contain a smaller number of partitions such that each partition in $\mathcal{W}'$ represents multiple partitions in $\mathcal{W}^\star$. Similarly, we can compute a corresponding aggregate set of states $\mathcal{F}'$ as:

$$s' = \{s \in \mathcal{F} \mid \exists x \in w', w' \in \mathcal{W}', (x, s) \in Q\}$$

where each aggregate state s' is a set representing multiple states in $\mathcal{F}$. Whenever possible, we will carry out our analysis using the aggregated-partitioning $\mathcal{W}'$ (and $\mathcal{F}'$) and use the fine-partitioning $\mathcal{W}^\star$ only if deemed necessary. Details of the workspace partitioning and computing the corresponding affine maps representing the LiDAR imaging function are given in Section 4.4.

Algorithm 11 Verify-NN($\mathcal{X}, \delta_{\mathrm{NN}}$)

```
1: Step 1: Partition the workspace
2: $(\mathcal{W}^\star, \mathcal{W}') = $ WKSP-PARTITION$(\mathcal{W}, \mathcal{O}, \theta_p, \theta_p)$

3: Step 2: Compute the finite state abstraction $\mathcal{S}_\mathcal{F}$
4: Step 2.1: Compute the states of $\mathcal{S}_\mathcal{F}$
5: $(\mathcal{F}, \mathcal{F}', Q) = $ STATE-SPACE-PARTITON$(\mathcal{W}^\star, \mathcal{W}')$
6: for each $s$ and $s'$ in $\mathcal{F}$ do
7:     $\delta_\mathcal{F}$.ADD-TRANSITION$(s, s')$
8: end for
9: Step 2.2: Pre-process the neural network
10: for each $s$ in $\mathcal{F}$ do
11:     $\mathcal{X}_s = \{x \in \mathcal{X} \mid (x, s) \in Q\}$
12:     $CE_s = $ PRE-PROCESS$(\mathcal{X}_s, \delta_{\mathrm{NN}})$
13: end for
14: Step 2.3: Compute the transition map $\delta_\mathcal{F}$
15: for each $s$ in $\mathcal{F}$ and $s'$ in $\mathcal{F}'$ where $s \notin s'$ do
16:     $\mathcal{X}_s = \{x \in \mathcal{X} \mid (x, s) \in Q\}$
17:     $\mathcal{X}_{s'} = \{x \in \mathcal{X} \mid (x, s^\star) \in Q, \ \forall s^\star \in s'\}$
18:     Status = CHECK-FEASIBILITY$(\mathcal{X}_s, \mathcal{X}_{s'}, \delta_{\mathrm{NN}}, CE_s)$
19:     if Status == INFEASIBLE then
20:         for each $s^\star$ in $s'$ do
21:             $\delta_\mathcal{F}$.REMOVE-TRANSITION$(s, s^\star)$
22:         end for
23:     else
24:         for each $s^\star$ in $s'$ do
25:             $\mathcal{X}_{s^\star} = \{x \in \mathcal{X} \mid (x, s^\star) \in Q\}$
26:             Status = CHECK-FEASIBILITY$(\mathcal{X}_s, \mathcal{X}_{s^\star}, \delta_{\mathrm{NN}}, CE_s)$
27:             if Status == INFEASIBLE then
28:                 $\delta_\mathcal{F}$.REMOVE-TRANSITION$(s, s^\star)$
29:             end if
30:         end for
31:     end if
32: end for
```

The state transition map $\delta_\mathcal{F}$ is computed as follows. First, we assume a transition exists between any two states s and s' in $\mathcal{F}$ (line 6- 7 in Algorithm 11). Next, we start eliminating unnecessary transitions. We observe that regions in the workspace that are adjacent or within some vicinity are more likely to force the need of transitions between their corresponding abstract states. Similarly, regions in the workspace that are far from each other are more likely to prohibit transitions between their corresponding abstract states. Therefore, in an attempt to reduce the number of computational steps in our algorithm, we check the

Algorithm 12 (Continue Algorithm 11) VERIFY-NN($\mathcal{X}, \delta_{\text{NN}}$)

```
1: Step 3: Compute the safe set
2: Step 3.1: Mark the abstract states corresponding to obstacles and workspace boundary as unsafe
       $\mathcal{F}^0_{\text{unsafe}} = \{s \in \mathcal{F} \mid \exists x \in \mathcal{X} : (x, s) \in Q,\ \zeta(x) \in \mathcal{O}_i, \mathcal{O}_i \in \mathcal{O}\}$
3: Step 3.2: Iteratively compute the predecessors of the abstract unsafe states
4: STATUS = FIXED-POINT-NOT-REACHED
5: while STATUS == FIXED-POINT-NOT-REACHED do
6:     $\mathcal{F}^k_{\text{unsafe}} = \mathcal{F}^{k-1}_{\text{unsafe}} \cup \text{PRE}(\mathcal{F}^{k-1}_{\text{unsafe}})$
7:     if $\mathcal{F}^k_{\text{unsafe}} == \mathcal{F}^{k-1}_{\text{unsafe}}$ then
8:         STATUS = FIXED-POINT-REACHED
9:     end if
10: end while
11: $\mathcal{F}_{\text{safe}} = \mathcal{F} \setminus \mathcal{F}_{\text{unsafe}}$
12: Step 3.3: Compute the set of safe states
13: $\mathcal{X}_{\text{safe}} = \{x \in \mathcal{X} \mid \exists s \in \mathcal{F}_{\text{safe}} : (x, s) \in Q\}$
14: Return $\mathcal{X}_{\text{safe}}$
```

transition feasibility between a state $s \in \mathcal{F}$ and an aggregate state $s' \in \mathcal{F}'$. If our algorithm (`CHECK-FEASIBILITY`) asserted that the neural network δ_{NN} prohibits the robot from transitioning between the regions corresponding to s and s' (denoted by $\mathcal{X}_s$ and $\mathcal{X}_{s'}$, respectively), then we conclude that no transition in $\delta_{\mathcal{F}}$ is feasible between the abstract state s and all the abstract states $s^\star$ in s' (lines 15-21 in Algorithm 11). This leads to a reduction in the number of state pairs that need to be checked for transition feasibility. Conversely, if our algorithm (`CHECK-FEASIBILITY`) asserted that the neural network δ_{NN} allows for a transition between the regions corresponding to s and s', then we proceed by checking the transition feasibility between the state s and all the states $s^\star$ contained in the aggregate state $s^\star$ (lines 24-28 in Algorithm 11).

Checking the transition feasibility (`CHECK-FEASIBILITY`) between two abstract states entails reasoning about the robot dynamics, the neural network, along with the affine map representing the LiDAR imaging computed from the previous workspace partitioning. While the robot dynamics is assumed linear, the imaging function is affine, the technical difficulty lies in reasoning about the behavior of the neural network controller. Thanks to the ReLU activation functions in the neural network, we can encode the problem of checking the transition feasibility between two regions as formula φ, called monotone Satisfiability Modulo Convex (SMC) formula [134, 135], over Boolean and convex constraints representing, respectively, the ReLU phases and the dynamics, the neural network weights, and the imaging constraints. In addition to using the SMC solver to check the transition feasibility (`CHECK-FEASIBILITY`) between abstract states, it will be used also to perform some pre-processing of the neural network function δ_{NN} (lines 10-12 in Algorithm 11) which is going to speed up the process of checking the the transition feasibility. Details of the SMC encoding and the strategy to check transition feasibility (`CHECK-FEASIBILITY`) are given in Section 4.5.

Once the finite state abstraction $\mathcal{S}_{\mathcal{F}}$ and the simulation relation Q is computed, the next step is to partition the finite states $\mathcal{F}$ into a set of unsafe states $\mathcal{F}_{\text{unsafe}}$ and a set of safe states $\mathcal{F}_{\text{safe}}$ using the following fixed-point computation:

$$\mathcal{F}^k_{\text{unsafe}} = \begin{cases} \{s \in \mathcal{F} \mid \exists x \in \mathcal{X} : (x,s) \in Q,\ \zeta(x) \in \mathcal{O}_i, \mathcal{O}_i \in \mathcal{O}\} & k = 0 \\ \mathcal{F}^{k-1}_{\text{unsafe}} \bigcup_{s \in \mathcal{F}^{k-1}_{\text{unsafe}}} \textsc{Pre}(s) & k > 0 \end{cases}$$

$$\mathcal{F}_{\text{unsafe}} = \lim_{k \to \infty} \mathcal{F}^k_{\text{unsafe}}, \qquad \mathcal{F}_{\text{safe}} = \mathcal{F} \setminus \mathcal{F}_{\text{unsafe}}.$$

where the $\mathcal{F}^0_{\text{unsafe}}$ represents the abstract states corresponding to the obstacles and workspace boundaries, $\mathcal{F}^k_{\text{unsafe}}$ with $k > 0$ represents all the states that can reach $\mathcal{F}^0_{\text{unsafe}}$ in k-steps, and $\textsc{Pre}(s)$ is defined as:

$$\textsc{Pre}(s) = \{s' \in \mathcal{F} \mid s \in \delta_{\mathcal{F}}(s')\}.$$

The remaining abstract states are then marked as the set of safe states $\mathcal{F}_{\text{safe}}$. Finally, we can compute the set of safe states $\mathcal{X}_{\text{safe}}$ as:

$$\mathcal{X}_{\text{safe}} = \{x \in \mathcal{X} \mid \exists s \in \mathcal{F}_{\text{safe}} : (x,s) \in Q\}.$$

These computations are summarized in lines 2-13 in Algorithm 11.

4.4 Imaging-Adapted Workspace Partitioning

We start by introducing the notation of the important geometric objects. We denote by $\textsc{Ray}(w, \theta)$ the ray originated from a point $w \in \mathcal{W}$ in the direction θ, i.e.:

$$\textsc{Ray}(w, \theta) = \{w' \in \mathcal{W} \mid \text{ atan2}(w' - w) = \theta\}.$$

Similarly, we denote by $\textsc{Line}(w_1, w_2)$ the line segment between the points w_1 and w_2, i.e.:

$$\textsc{Line}(w_1, w_2) = \{w' \in \mathcal{W} \mid w' = \nu w_1 + (1 - \nu) w_2,\ 0 \leq \nu \leq 1\}.$$

For a convex polytope $P \subseteq \mathcal{W}$, we denote by $\textsc{Vert}(P)$, its set of vertices and by $\textsc{Edge}(P)$ its set of line segments representing the edges of the polytope.

4.4.1 Imaging-Adapted Partitions

The basic idea behind our algorithm is to partition the workspace into a set of polytopic sets (or regions) such that for each region $\mathcal{R}$ the LiDAR rays intersects with the same obstacle/workspace edge regardless of the robot positions $\zeta(x) \in \mathcal{R}$. To formally characterize this property, let $\mathcal{O}^\star = \bigcup_{\mathcal{O}_i \in \mathcal{O}} \mathcal{O}_i$ be the set of all points in the workspace in which an obstacle or workspace boundary exists. Consider a workspace partition $\mathcal{R} \subseteq \mathcal{W}$ and a robot position $\zeta(x)$ that lies inside this partition, i.e., $\zeta(x) \in \mathcal{R}$. The intersection between the kth LiDAR laser beam $\textsc{Ray}(\zeta(x), \theta_k)$ and $\mathcal{O}^\star$ is a unique point characterized as:

$$z_{k,\zeta(x)} = \underset{z \in \mathcal{W}}{\operatorname{argmin}} \|z - \zeta(x)\|_2 \quad \text{s.t.} \quad z \in \textsc{Ray}(\zeta(x), \theta_k) \cap \mathcal{O}^\star. \tag{4.5}$$

By sweeping $\zeta(x)$ across the whole region $\mathcal{R}$, we can characterize the set of all possible intersection points as:

$$\mathcal{L}_k(\mathcal{R}) = \bigcup_{\zeta(x)\in\mathcal{R}} z_{k,\zeta(x)}. \tag{4.6}$$

Using the set $\mathcal{L}_k(\mathcal{R})$ described above, we define the notion of imaging-adapted partitions as follows.

Definition 4.4. *A set $\mathcal{R} \subset \mathcal{W}$ is said to be an imaging-adapted partition if the following property holds:*

$$\mathcal{L}_k(\mathcal{R}) \textit{ is a line segment } \quad \forall k \in \{1, \dots, N\}. \tag{4.7}$$

Figure 4.3 shows concrete examples of imaging-adapted partitions. Imaging-adapted partitions enjoy the following property:

Lemma 4.2. *Consider an imaging-adapted partition $\mathcal{R}$ with corresponding sets $\mathcal{L}_1(\mathcal{R}), \dots, \mathcal{L}_N(\mathcal{R})$. The LiDAR imaging function $d : \mathcal{R} \to \mathbb{R}^{2N}$ is an affine function of the form:*

$$d_k(\zeta(x)) = P_{k,\mathcal{R}}\zeta(x) + Q_{k,\mathcal{R}}, \qquad d = (d_1, \dots, d_N) \tag{4.8}$$

for some constant matrices $P_{k,\mathcal{R}}$ and vectors $Q_{k,\mathcal{R}}$ that depend on the region $\mathcal{R}$ and the LiDAR angle θ_k.

Proof. Consider an arbitrary LiDAR laser with an angle θ_k and arbitrary robot position $\zeta(x) \in \mathcal{R}$. The LiDAR image d_k can be written as:

$$d_k = z_{k,\zeta(x)}(\mathcal{R}) - \zeta(x) \tag{4.9}$$

where $z_{k,\zeta(x)}(\mathcal{R})$ is defined in (4.5). It follows from the fact that $\mathcal{R}$ is an imaging-adapted partition that the set $\mathcal{L}_k(\mathcal{R})$ is a line segment. Let $a_k, b_k \in \mathbb{R}^2$ be the vertices of this line segment, i.e., $(a_k, b_k) = \textsc{Vert}(\mathcal{L}_k(\mathcal{R}))$ and recall that $z_{k,\zeta(x)}(\mathcal{R})$ satisfies $z_{k,\zeta(x)}(\mathcal{R}) \in \mathcal{L}_k(\mathcal{R})$ and hence $z_{k,\zeta(x)}(\mathcal{R})$ lies on the line segment $\textsc{Line}(a_k, b_k)$. Therefore there exists a ν_k such that:

$$z_{k,\zeta(x)}(\mathcal{R}) = (1-\nu_k)a_k + \nu b_k \tag{4.10}$$

where $0 \le \nu_k \le 1$. It follows from the definition of $z_{k,\zeta(x)}(\mathcal{R})$ in (4.5) that $z_{k,\zeta(x)}(\mathcal{R})$ also lies on $\textsc{Ray}(\zeta(x), \theta_k)$ and hence:

$$\tan(\theta_k) = \frac{z_2 - x_2}{z_1 - x_1}, \tag{4.11}$$

where (z_1, z_2) are the two elements of $z_{k,\zeta(x)}(\mathcal{R}) \in \mathcal{R} \subset \mathbb{R}^2$ while (x_1, x_2) are the corresponding two elements of $\zeta(x) \in \mathcal{R} \subset \mathbb{R}^2$. Substituting (4.10) in (4.11) yields:

$$\tan(\theta_k) = \frac{(1-\nu_k)a_2 + \nu_k b_2 - x_2}{(1-\nu_k)a_1 + \nu_k a_2 - x_1} \tag{4.12}$$

where $(a_1, a_2) = a_k$ and $(b_1, b_2) = b_k$ are the two elements of a_k and b_k, respectively. By solving (4.12) for ν_k, we conclude that:

$$\nu_k = A_{\nu_k}\zeta(x) + b_{\nu_k}, \tag{4.13}$$

$$A_{\nu_k} = \begin{cases} \begin{bmatrix} \frac{1}{b_2 - a_2} & 0 \end{bmatrix} & \theta_k = \pi/2 \text{ or } 3\pi/2 \\ \\ \begin{bmatrix} \frac{\tan(\theta_k)}{a_2 - b_2 + (b_1 - a_1)\tan(\theta_k) - a_2} & \frac{1}{a_2 - b_2 + (b_1 - a_1)\tan(\theta_k)} \end{bmatrix} & \text{otherwise,} \end{cases}$$

$$b_{\nu_k} = \begin{cases} 0 & \theta_k = \pi/2 \text{ or } 3\pi/2 \\ \frac{a_2 - a_2\tan(\theta_k)}{a_2 - b_2 + (b_1 - a_1)\tan(\theta_k)} & \text{otherwise,} \end{cases}$$

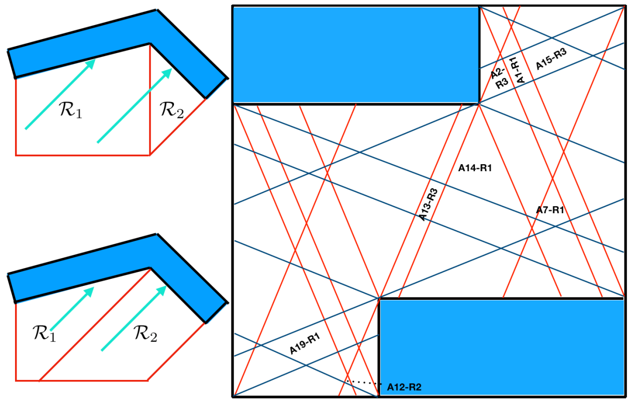


Figure 4.3: (left-up) A partitioning of the workspace that is *not* imaging-adapted. Within region $\mathcal{R}_1$, the LiDAR ray (cyan arrow) intersects with different obstacle edges depending on the robot position. (left-down) A partitioning of the workspace that is imaging-adapted. For both regions $\mathcal{R}_1$ and $\mathcal{R}_2$, the LiDAR ray (cyan arrow) intersects the same obstacle edge regardless of the robot position. (right) Imaging-adapted partitioning of the workspace used in Section 4.6.

.

where A_{ν_k} and b_{ν_k} are constants that depends on the values of the constants a_k, b_k, and θ_k. From (4.9),(4.10), and (4.13), we conclude that:

$$d_k(\zeta(x)) = P_{k,\mathcal{R}}\zeta(x) + Q_{k,\mathcal{R}} \tag{4.14}$$

with $P_{k,\mathcal{R}} = (b_k - a_k)(A - I)$ (where I is the 2×2 identity matrix) and $Q_{k,\mathcal{R}} = a_k + b_{\nu_k}(b_k - a_k)$ are constants that depends on a_k, b_k and θ_k form which we conclude that $d_k(\zeta(x))$ is affine. Note that we added the subscript $\mathcal{R}$ to $P_{k,\mathcal{R}}$ and $Q_{k,\mathcal{R}}$ to emphasize the face that these constant matrices depends on the region $\mathcal{R}$. Since we picked k arbitrary, we finally conclude that $d(\zeta(x))$ is also an affine function. $\square$

4.4.2 Partitioning the Workspace

Motivated by Lemma 4.2, our objective is to design an algorithm that can partition the workspace $\mathcal{W}$ into a set of imaging-adapted partitions. As summarized in Algorithm 13, our algorithm starts by computing a set of line segments $\mathcal{G}$ that will be used to partition the workspace (lines 1-6 in Algorithm 13). This set of line segments $\mathcal{G}$ are computed as follows. First, we define the set $\mathcal{V}$ as the one that contains all the vertices of the workspace and the obstacles, i.e., $\mathcal{V} = \bigcup_{\mathcal{O}_i \in \mathcal{O}} \textsc{Vert}(\mathcal{O}_i)$. Next, we consider rays originating from all the vertices in $\mathcal{V}$ and pointing in the opposite directions of the angles $\theta_1, \ldots, \theta_N$. By intersecting these rays with the obstacles and picking the closest intersection points, we acquire the line segments $\mathcal{G}$ that will be used to partition the workspace. In other words, $\mathcal{G}$ is computed as:

$$\mathcal{G}_k = \{\textsc{Line}(v, z) \mid v \in \mathcal{V},\ z = \underset{z \in \textsc{Ray}(v, \theta_k + \pi) \cap \mathcal{O}^\star}{\operatorname{argmin}} \|z - v\|_2\}$$
$$\mathcal{G} = \bigcup_{k=1}^{N} \mathcal{G}_k \tag{4.15}$$

Thanks to the fact that the vertices v are fixed, finding the intersection between $\textsc{Ray}(v, \theta_k + \pi)$ and $\mathcal{O}^\star$ is a standard ray-polytope intersection problem which can be solved efficiently [13].

The next step is to compute the intersection points $\mathcal{P}$ between the line segments $\mathcal{G}$ and the edges of the obstacles $\mathcal{E} = \bigcup_{\mathcal{O}_i \in \mathcal{O}} \textsc{Edge}(\mathcal{O}_i)$. A naive approach will be to consider all combinations of line segments in $\mathcal{G} \cup \mathcal{E}$ and test them for intersection. Such approach is combinatorial and would lead to an execution time that is exponential in the number of laser angles and vertices of obstacles. Thanks to the advances in the literature of computational geometry, such intersection points can be computed efficiently using the plane-sweep algorithm [13]. The plane-sweep algorithm simulates the process of sweeping a line downwards over the plane. The order of the line segments $\mathcal{G} \cup \mathcal{E}$ from left to right as they intersect the sweep line is stored in a data structure called the sweep-line status. Only segments that are adjacent in the horizontal ordering need to be tested for intersection. Though the sweeping process can be visualized as continuous, the plane-sweep algorithm sweeps only the endpoints of segments in $\mathcal{G} \cup \mathcal{E}$, which are given beforehand, and the intersection points, which are computed on the fly. To keep track of the endpoints of segments in $\mathcal{G} \cup \mathcal{E}$ and the intersection points, we use a balanced binary search tree as data structure to support insertion, deletion, and searching in $O(log\ n)$ time, where n is number of elements in the data structure.

The final step is to use the line segments $\mathcal{G} \cup \mathcal{E}$ and their intersection points $\mathcal{P}$, discovered by the plane-sweep algorithm, to compute the workspace partitions. To that end, consider the undirected planar graph whose vertices are the intersection points $\mathcal{P}$ and whose edges are $\mathcal{G} \cup \mathcal{E}$, denoted by $\textsc{Graph}(\mathcal{P}, \mathcal{G} \cup \mathcal{E})$. The workspace partitions are equivalent to finding subgraphs of $\textsc{Graph}(\mathcal{P}, \mathcal{G} \cup \mathcal{E})$ such that each subgraph contains only one simple cycle [1]. To find these simple cycles, we use a modified Depth-First-Search algorithm in which it starts from a vertex in the planar graph and then traverses the graph by considering the rightmost turns along the vertices of the graph. Finally, the workspace partitions are computed as the convex hulls of all the vertices in the computed simple cycles. It follows directly from the fact that each region is constructed from the vertices of a simple cycle that there exists no line segment in $\mathcal{G} \cup \mathcal{E}$ that intersects with the interior of any region, i.e., for any workspace partition $\mathcal{R}$, the following holds:

$$\text{int}(\mathcal{R}) \cap e = \emptyset \qquad \forall e \in \mathcal{G} \cup \mathcal{E} \tag{4.16}$$

This process is summarized in lines 9-18 in Algorithm 13. An important property of the regions determined by Algorithm 13 is stated by the following proposition.

Proposition 4.3. *Consider a workspace partition $\mathcal{R}$ that is computed by Algorithm 13 and*

[1]A cycle in an undirected graph is called simple when no repetitions of vertices and edges, other than the starting and ending vertex.

satisfies (4.16)*. The following property holds for any LiDAR ray with angle θ_k:*

$$\exists e \in \mathcal{E} \qquad \textit{such that} \qquad \mathcal{L}_k(\mathcal{R}) \subseteq e,$$

where $\mathcal{L}_k(\mathcal{R})$ is defined in (4.6)*.*

Proof. We assume, for the sake of contradiction, that there exist two obstacle edges $\textsc{Line}(v_1, v_2), \textsc{Line}(w_1, w_2) \in \mathcal{E}$ with $(v_1, v_2) \neq (w_1, w_2)$ along with rays originating from points $p_1, p_2 \in \mathcal{R}$ such that the intersection points:

$$z_1 = \arg \min_{z_1 \in \textsc{Ray}(p_1, \theta_k) \cap \mathcal{O}^\star} \|z_1 - p_1\|, \qquad z_2 = \arg \min_{z_2 \in \textsc{Ray}(p_2, \theta_k) \cap \mathcal{O}^\star} \|z_2 - p_2\|.$$

satisfy $z_1 \in \textsc{Line}(v_1, v_2)$ and $z_2 \in \textsc{Line}(w_1, w_2)$.

Now consider the set P_1 defined as follows:

$$P_1 = \{p \in \mathcal{R} \mid p \in \textsc{Ray}(z, \theta_k + \pi), \quad \forall z \in \textsc{Line}(v_1, v_2)\}$$

It follows from the definition of z_1 that $p_1 \in P_1$. It also follows from the definition of P_1 that $P_1 \subseteq \mathcal{R}$. Moreover, it follows from the definition of the set $\mathcal{E}$ along with the fact that $\textsc{Line}(v_1, v_2) \in \mathcal{E}$ that v_1 and v_2 satisfy $v_1, v_2 \in \mathcal{V}$. It follows from the definition of the set $\mathcal{G}$ in (4.15) that it contains line segments from the rays originated at elements of the set $\mathcal{V}$. Hence, there exists v_1', v_1'', v_2', v_2'' such that the line segments $\textsc{Line}(v_1', v_1'')$ and $\textsc{Line}(v_2', v_2'')$ satisfy:

$$\textsc{Line}(v_1', v_1'') \subset \textsc{Ray}(v_1, \theta_k + \pi) \subset P_1 \subseteq \mathcal{R}, \qquad \textsc{Line}(v_1', v_1'') \in \mathcal{G} \tag{4.17}$$

$$\textsc{Line}(v_2', v_2'') \subset \textsc{Ray}(v_2, \theta_k + \pi) \subset P_1 \subseteq \mathcal{R}, \qquad \textsc{Line}(v_2', v_2'') \in \mathcal{G} \tag{4.18}$$

However, it follows from (4.16) that line segments that are elements of $\mathcal{G}$ do not intersect the interior of $\mathcal{R}$. Hence:

$$\left.\begin{aligned} \textsc{Line}(v_1', v_1'') \subset \mathcal{R} \\ \textsc{Line}(v_1', v_1'') \cap \operatorname{int}(\mathcal{R}) = \emptyset \end{aligned}\right\} \Rightarrow \textsc{Line}(v_1', v_1'') \subset \partial\mathcal{R} \tag{4.19}$$

$$\left.\begin{aligned} \textsc{Line}(v_2', v_2'') \subset \mathcal{R} \\ \textsc{Line}(v_2', v_2'') \cap \operatorname{int}(\mathcal{R}) = \emptyset \end{aligned}\right\} \Rightarrow \textsc{Line}(v_2', v_2'') \subset \partial\mathcal{R} \tag{4.20}$$

Similarly, by considering w_1, w_2, z_2, we conclude that there exists line segments $\textsc{Line}(w_1', w_1'') \subset \textsc{Ray}(w_1, \theta_k + \pi)$ and $\textsc{Line}(w_2', w_2'') \subset \textsc{Ray}(v_2, \theta_k + \pi)$ are elements of $\mathcal{G}$ and satisfy:

$$\textsc{Line}(w_1', w_1'') \subset \partial\mathcal{R}, \quad \textsc{Line}(w_2', w_2'') \subset \partial\mathcal{R} \tag{4.21}$$

It follows from Euclidean geometry that any polygon in $\mathbb{R}^2$ can have at maximum two edges that are "parallel". It also follows from (4.19)-(4.21) that $\textsc{Line}(v_1', v_1''), \textsc{Line}(v_2', v_2''), \textsc{Line}(w_1', w_1'')$, and $\textsc{Line}(w_2', w_2'')$ are edges of $\mathcal{R}$. However, it follows from the definitions of the four line segments that they are subsets of rays that share the same angle, and hence they are all parallel. Hence we conclude that $\textsc{Line}(v_1', v_1'') = \textsc{Line}(w_1', w_1'')$ and $\textsc{Line}(v_2', v_2'') = \textsc{Line}(w_2', w_2'')$ from which it is direct to conclude that $(v_1, v_2) = (w_1, w_2)$, a contradiction.

$\square$

We conclude this section by stating our first main result, quantifying the correctness and complexity of Algorithm 13.

Theorem 4.4. *Given a workspace with polytopic obstacles and a set of laser angles $\theta_1, \ldots, \theta_N$, then Algorithm 13 computes the partitioning $\mathcal{R}_1, \ldots, \mathcal{R}_r$ such that:*

1. $\mathcal{W} = \bigcup_{i=1}^{r} \mathcal{R}_i$, *2.* $\mathcal{R}_i$ *is an imaging-adapted partition* $\quad \forall i = 1, \ldots, r$,

3. $d : \mathcal{R}_i \to \mathbb{R}^{2N}$ *is affine* $\quad \forall i = 1, \ldots, r$.

Moreover, the time complexity of Algorithm 13 is $O(M \log\ M + I \log\ M)$*, where* $M = |\mathcal{G} \cup \mathcal{E}|$ *is cardinality of* $\mathcal{G} \cup \mathcal{E}$*, and* I *is number of intersection points between segments in* $\mathcal{G} \cup \mathcal{E}$*.*

Algorithm 13 WKSP-PARTITION $(\mathcal{W}, \mathcal{O}, \theta, \theta_p)$

```
1: Step 1: Generate partition segments
2: $\mathcal{O}^\star = \bigcup_{\mathcal{O}_i \in \mathcal{O}} \mathcal{O}_i, \quad \mathcal{V} = \bigcup_{\mathcal{O}_i \in \mathcal{O}} \textsc{Vert}(\mathcal{O}_i), \quad \mathcal{E} = \bigcup_{\mathcal{O}_i \in \mathcal{O}} \textsc{Edge}(\mathcal{O}_i)$
3: for $k \in \{1, \ldots, N\}$ do
4:     Use a ray-polygon intersection algorithm to compute:
           $$\mathcal{G}_k = \{\textsc{Line}(v, z) \mid v \in \mathcal{V},\ z = \operatorname*{argmin}_{z \in \textsc{Ray}(v, \theta_k + \pi) \cap \mathcal{O}^\star} \|z - v\|_2\}$$
5: end for
6: $\mathcal{G} = \bigcup_{k \in \theta} \mathcal{G}_k, \qquad \mathcal{G}' = \bigcup_{k \in \theta_p} \mathcal{G}_k$

7: Step 2: Compute intersection points
8: $\mathcal{P} = \text{PLANE-SWEEP}(\mathcal{G} \cup \mathcal{E}), \qquad \mathcal{P}' = \text{PLANE-SWEEP}(\mathcal{G}' \cup \mathcal{E})$

9: Step 3: Construct the partitions
10: $\textsc{Cycles} = \textsc{Find-Vertices-Of-Simple-Cycle}(\textsc{Graph}(\mathcal{P}, \mathcal{G} \cup \mathcal{E}))$
11: $\textsc{Cycles}' = \textsc{Find-Vertices-Of-Simple-Cycle}\ (\textsc{Graph}(\mathcal{P}', \mathcal{G}' \cup \mathcal{E}))$.
12: for $c \in \textsc{Cycles}$ do
13:     $\mathcal{R} = \textsc{Convex-Hull}(c)$
14:     $\mathcal{W}^\star$.ADD($\mathcal{R}$)
15: end for
16: for $c \in \textsc{Cycles}'$ do
17:     $\mathcal{R}' = \textsc{Convex-Hull}(c)$
18:     $\mathcal{W}'$.ADD($\mathcal{R}'$)
19: end for
20: Return $\mathcal{W}^\star, \mathcal{W}'$
```

4.5 Computing the Finite State Abstraction

Once the workspace is partitioned into imaging-adapted partitions $\mathcal{W}^\star = \{\mathcal{R}_1, \ldots, \mathcal{R}_r\}$ and the corresponding imaging function is identified, the next step is to compute the finite state transition abstraction $\mathcal{S}_\mathcal{F} = (\mathcal{F}, \delta_\mathcal{F})$ of the closed loop system along with the simulation relation Q. The first step is to define the state space $\mathcal{F}$ and its relation to $\mathcal{X}$. To that end, we start by computing a partitioning of the state space $\mathcal{X}$ that respects $\mathcal{W}^\star$. For the sake of simplicity, we consider $\mathcal{X} \subset \mathbb{R}^n$ that is n-orthotope, i.e., there exists constants $\underline{x}_i, \overline{x}_i \in \mathbb{R}, i = 1, \ldots, n$ such that:

$$\mathcal{X} = \{x \in \mathbb{R}^n \mid \underline{x}_i \leq x_i < \overline{x}_i, \quad i = 1, \ldots, n\}$$

Now, given a discretization parameter $\epsilon \in \mathbb{R}^+$, we define the state space $\mathcal{F}$ as:

$$\mathcal{F} = \{(k_1, k_3, \ldots, k_n) \in \mathbb{N}^{n-1} \mid 1 \leq k_1 \leq r, 1 \leq k_i \leq \frac{\overline{x}_i - \underline{x}_i}{\epsilon}, i = 3, \ldots, n\} \tag{4.22}$$

where r is the number of regions in the partitioning $\mathcal{W}^\star$. In other words, the parameter ϵ is used to partition the state space into hyper-cubes of size ϵ in each dimension $i = 3, \ldots, n$. A state $s \in \mathcal{F}$ represents the index of a region in $\mathcal{W}^\star$ followed by the indices identifying a hypercube in the remaining $n - 2$ dimensions. Note that for the simplicity of notation, we assume that $\overline{x}_i - \underline{x}_i$ is divisible by ϵ for all $i = 1, \ldots, n$. We now define the relation $Q \subseteq \mathcal{X} \times \mathcal{F}$ as:

$$\begin{aligned} Q = \{(x, s) \in \mathcal{X} \times \mathcal{F} \mid s = (k_1, k_3, \ldots, k_n), x = (\zeta(x), x_3, \ldots, x_n), \\ \zeta(x) \in \mathcal{R}_{k_1}, \underline{x}_i + \epsilon(k_i - 1) \leq x_i < \underline{x}_i + \epsilon k_i, i = 3, \ldots, n\}. \end{aligned} \tag{4.23}$$

Finally, we define the state transition function $\delta_\mathcal{F}$ of $\mathcal{S}_\mathcal{F}$ as follows:

$$\begin{aligned} &(k_1', k_3', \ldots k_n') \in \delta_\mathcal{F}((k_1, k_3, \ldots k_n)) \text{ if} \\ &\quad \exists x = (\zeta(x), x_3, \ldots, x_n) \in \mathcal{R}_{k_1}, \underline{x}_i + \epsilon(k_i - 1) \leq x_i < \underline{x}_i + \epsilon k_i, \\ &\quad x' = (\zeta(x'), x_3', \ldots, x_n') \in \mathcal{R}_{k_1'}, \underline{x}_i + \epsilon(k_i' - 1) \leq x_i' < \underline{x}_i + \epsilon k_i', \\ &\quad \text{s.t.} \quad x' = Ax + Bf_{NN}(d(x)). \end{aligned} \tag{4.24}$$

It follows from the definition of $\delta_{\mathcal{F}}$ in (4.24) that checking the transition feasibility between two states s and s' is equivalent to searching for a robot initial and goal states along with a LiDAR image that will force the neural network controller to generate an input that moves the robot between the two states while respecting the robots dynamics. In the reminder of this section, we focus on solving this feasibility problem.

4.5.1 SMC Encoding of NN

We translate the problem of checking the transition feasibility in $\delta_{\mathcal{F}}$ into a feasibility problem over a monotone SMC formula [134, 135] as follows. We introduce the Boolean indicator variables b_j^l with $l = 1, \ldots, L$ and $j = 1, \ldots, M_l$ (recall that L represents the number of layers in the neural network, while M_l represents the number of neurons in the lth layer). These Boolean variables represent the phase of each ReLU, i.e., an asserted b_j^l indicates that the output of the jth ReLU in the lth layer is $h_j^l = (W^{l-1}h^{l-1}+w^{l-1})_j$ while a negated b_j^l indicates that $h_j^l = 0$. Using these Boolean indicator variables, we encode the problem of checking the transition feasibility between two states $s = (k_1, k_3, \ldots, k_n)$ and $s' = (k'_1, k'_3, \ldots, k'_n)$ as:

$$\exists\, x, x' \in \mathbb{R}^n, u \in \mathbb{R}^m, d \in \mathbb{R}^{2N}, (b^l, h^l, t^l) \in \mathbb{B}^{M_l} \times \mathbb{R}^{M_l} \times \mathbb{R}^{M_l}, l \in \{1, \ldots, L\}$$

subject to:

$$\zeta(x) \in \mathcal{R}_{k_1} \;\wedge\; \underline{x}_i + \epsilon(k_i - 1) \leq x_i < \underline{x}_i + \epsilon k_i,\; i = 3, \ldots, n \tag{4.25}$$

$$\wedge \zeta(x') \in \mathcal{R}_{k'_1} \wedge \underline{x}_i + \epsilon(k'_i - 1) \leq x'_i < \underline{x}_i + \epsilon k'_i,\; i = 3, \ldots, n \tag{4.26}$$

$$\wedge\, x' = Ax + Bu \tag{4.27}$$

$$\wedge\, d_k = P_{k, \mathcal{R}_{k_1}} \zeta(x) + Q_{k, \mathcal{R}_{k_1}}, \qquad k = 1, \ldots, N \tag{4.28}$$

$$\wedge \left(t^1 = W^0 d + w^0 \right) \;\wedge\; \left(\bigwedge_{l=2}^{L} t^l = W^{l-1} h^{l-1} + w^l \right) \tag{4.29}$$

$$\wedge \left(u = W^L h^L + w^L \right) \tag{4.30}$$

$$\wedge \bigwedge_{l=1}^{L} \bigwedge_{j=1}^{M_j} b^l_j \rightarrow \left[\left(h^l_j = t^l_j \right) \wedge \left(t^l_j \geq 0 \right) \right] \tag{4.31}$$

$$\wedge \bigwedge_{l=1}^{L} \bigwedge_{j=1}^{M_j} \neg b^l_j \rightarrow \left[\left(h^l_j = 0 \right) \wedge \left(t^l_j < 0 \right) \right] \tag{4.32}$$

where (4.25)-(4.26) encode the state space partitions corresponding to the states s and s'; (4.27) encodes the dynamics of the robot; (4.28) encodes the imaging function that maps the robot position into LiDAR image; (4.29)-(4.32) encodes the neural network controller that maps the LiDAR image into a control input.

Compared to Mixed-Integer Linear Programs (MILP), monotone SMC formulas avoid using encoding heuristics like big-M encoding which leads to numerical instabilities. The SMC decision procedures follow an iterative approach combining efficient Boolean Satisfiability (SAT) solving with numerical convex programming. When applied to the encoding above, at each iteration the SAT solver generates a candidate assignment for the ReLU indicator variables b^l_j. The correctness of these assignments are then checked by solving the corresponding set of convex constraints. If the convex program turned to be infeasible, indicating a wrong choice of the ReLU indicator variables, the SMC solver will identify the set of "Irreducible Infeasible Set" (IIS) in the convex program to provide the most succinct explanation of the conflict. This IIS will be then fed back to the SAT solver to prune its search space and provide the next assignment for the ReLU indicator variables. SMC solvers were shown to better handle problems (compared with MILP solvers) for problems with relatively large number of Boolean variables [135].

4.5.2 Pruning Search Space by Pre-processing

While a neural network with M ReLUs would give rise to 2^M combinations of possible assignments to the corresponding Boolean indicator variables, we observe that only several of those combinations are feasible for each workspace region. In other words, the LiDAR imaging function along with the workspace region enforces some constraints on the inputs to the neural network which in turn enforces constraints on the subsequent layers. By performing pre-processing on each of the workspace regions, we can discover those constraints and augment it to the SMC encoding (4.25)-(4.32) to prune combinations of assignments of the ReLU indicator variables.

To find such constraints, we consider an SMC problem with the fewer constraints (4.25), (4.28)-(4.32). By iteratively solving the reduced SMC problem and recording all the IIS conflicts produced by the SMC solver, we can compute a set of counter-examples that are unique for each region. By iteratively invoking the SMC solver while adding previous counter-examples as constraints until the problem is no longer satisfiable, we compute the set $\mathcal{R}$-Conflicts which represents all the counter-examples for region $\mathcal{R}$. Finally, we add the following constraint:

$$\bigvee_{c \in \mathcal{R}\text{-}\textsc{Conflicts}} \neg c \tag{4.33}$$

to the original SMC encoding (4.25)-(4.32) to prune the set of possible assignments to the ReLU indicator variables. In Section 4.6, we show that pre-processing would result in several orders of magnitude reduction in the execution time.

4.5.3 Correctness of NN Verification Algorithm

We end our discussion with the following results which assert the correctness of the whole framework described in this chapter. We first start by establishing the correctness of computing the finite state abstraction $\mathcal{S}_{\mathcal{F}}$ along with the simulation relation Q as follows:

Proposition 4.5. *Consider the finite state abstraction* $\mathcal{S}_{\mathcal{F}} = (\mathcal{F}, \delta_{\mathcal{F}})$ *where* $\mathcal{F}$ *is defined by* (4.22) *and* $\delta_{\mathcal{F}}$ *is defined by* (4.24) *and computed by means of solving the SMC formulas* (4.25)-(4.33)*. Consider also the system* $\mathcal{S}_{NN} = (\mathcal{X}, \delta_{NN})$ *where* $\delta_{NN} : x \mapsto Ax + Bf_{NN}(d(x))$*. For the relation* Q *defined in* (4.23)*, the following holds:* $\mathcal{S}_{NN} \preccurlyeq_Q \mathcal{S}_{\mathcal{F}}$.

Recall that Algorithm 11 applies standard reachability analysis on $\mathcal{S}_{\mathcal{F}}$ to compute the set of unsafe states. It follows directly from the correctness of the simulation relation Q established above that our algorithm computes an over-approximation of the set of unsafe states, and accordingly an under-approximation of the set of safe states. This fact is captured by the following result that summarizes the correctness of the proposed framework:

Theorem 4.6. *Consider the safe set* $\mathcal{X}_{safe}$ *computed by Algorithm 11. Then any trajectory* η_x *with* $\eta_x(0) \in \mathcal{X}_{safe}$ *is a safe trajectory.*

While Theorem 4.6 establishes the correctness of the proposed framework in Algorithm 11, two points needs to be investigated namely (i) complexity of Algorithm 11 and (ii) maximality of the set $\mathcal{X}_{\text{safe}}$. Although Algorithm 13 computes the imaging-adapted partitions efficiently (as shown in Theorem 4.4), analyzing a neural network with ReLU activation functions is shown to be NP-hard. Exacerbating the problem, Algorithm 11 entails analyzing the neural network a number of times that is exponential in the number of partition regions. In addition, floating point arithmetic used by the SMC solver may introduce errors that are not analyzed in this chapter. In Section 4.6, we evaluate the efficiency of using the SMC decision procedures to harness this computational complexity. As for the maximality of the computed $\mathcal{X}_{\text{safe}}$ set, we note that Algorithm 11 is not guaranteed to search for the maximal $\mathcal{X}_{\text{safe}}$.

4.6 Results

We implemented the proposed verification framework as described by Algorithm 11 on top of the SMC solver named *SATEX* [124]. All experiments were executed on an Intel Core i7 2.5-GHz processor with 16 GB of memory.

4.6.1 Scalability of the Workspace Partitioning Algorithm

As the first step of our verification framework, imaging-adapted workspace partitioning is tested for numerical stability with increasing number of laser angles and obstacles. Table 4.1 summarizes the scalability results in terms of the number of computed regions and the execution time grows as the number of LiDAR lasers and obstacle vertices increase. Thanks to adopting well-studied computational geometry algorithms, our partitioning process takes less than 1.5 minutes for the scenario where a LiDAR scanner is equipped with 298 lasers (real-world LiDAR scanners are capable of providing readings from 270 laser angles).

Table 4.1: Scalability results for the WKSP-PARTITION Algorithm

Number of Vertices	**Number of Lasers**	**Number of Regions**	**Time [s]**
8	8	111	0.0152
	38	1851	0.3479
	118	17237	5.5300
10	8	136	0.0245
	38	2254	0.4710
	118	20343	6.9380
12	8	137	0.0275
	38	2418	0.5362
	120	23347	8.0836
	218	76337	37.0572
	298	142487	86.6341

4.6.2 Computational Reduction Due to Pre-processing

The second step is to pre-process the neural network. In particular, we would like to answer the following question: given a partitioned workspace, how many ReLU assignments are feasible in each region, and if any, what is the execution time to find them out. Recall that a ReLU assignment is feasible if there exist a robot position and the corresponding LiDAR image that will lead to that particular ReLU assignment.

Thanks to the IIS counterexample strategy, we can find all feasible ReLU assignments in pre-processing. Our first observation is that the number of feasible assignments is indeed much smaller compared to the set of all possible assignments. As shown in Table 4.2, for a neural network with a total of 32 neurons, only 11 ReLU assignments are feasible (within the region under consideration). Comparing this number to $2^{32} = 4.3E9$ possibilities of ReLU assignments, we conclude that pre-processing is very effective in reducing the search space by several orders of magnitude.

Furthermore, we conducted an experiment to study the scalability of the proposed pre-processing for an increasing number of ReLUs. To that end, we fixed one choice of workspace regions while changing the neural network architecture. The execution time, the number of

Table 4.2: Execution time of the SMC-based pre-processing as a function of the neural network architecture.

Number of Hidden Layers	**Total Number of Neurons**	**Number of Feasible ReLU Assignments**	**Number of Counter-examples**	**Time [s]**
1	32	11	60	2.7819
	72	31	183	11.4227
	92	58	265	18.4807
	102	68	364	43.2459
	152	101	540	78.3015
	172	146	778	104.4720
	202	191	897	227.2357
	302	383	1761	656.3668
	402	730	2614	1276.4405
	452	816	4325	1856.0418
	502	1013	3766	2052.0574
	552	1165	4273	4567.1767
	602	1273	5742	6314.4890
	652	1402	5707	7166.3059
	702	1722	6521	8813.1829
2	22	3	94	1.3180
	42	19	481	10.9823
	62	35	1692	53.2246
	82	33	2685	108.2584
	102	58	5629	292.7412
	122	71	9995	739.4883
	142	72	18209	2098.0220
	162	98	34431	6622.1830
	182	152	44773	12532.8552
3	32	5	319	5.7227
	47	7	5506	148.8727
	62	45	72051	12619.5353
4	22	9	205	10.4667
	42	5	1328	90.1148

generated counterexamples, along with the number of feasible ReLU assignments are given in Table 4.2. For the case of neural networks with one hidden layer, our implementation of the counterexample strategy is able to find feasible ReLU assignments for a couple of hundreds of neurons in less than 4 minutes. In general, the number of counterexamples, and hence feasible ReLU assignments, and execution time grows with the number of neurons. However, the number of neurons is not the only deciding factor. Our experiments show that the depth of the network plays a significant role in affecting the scalability of the proposed algorithms. For example, comparing the neural network with one hidden layer and a hundred neurons per layer versus the network with two layers and fifty neurons per layer we notice that both networks share the same number of neurons. Nevertheless, the deeper network resulted in one order of magnitude increase regarding the number of generated counterexamples and one order of magnitude increase in the corresponding execution time. Interestingly, both of the architectures share a similar number of feasible ReLU assignments. In other words,

Table 4.3: Execution time of the SMC-based pre-processing as a function of the workspace region. Region indices are shown in Figure 4.3.

Region Index	**Number of Feasible ReLU Assignments**	**Number of Counter-examples**	**Time [s]**
A2-R3	33	2685	108.2584
A14-R1	55	4925	215.8251
A13-R3	7	1686	69.4158
A1-R1	25	2355	99.2122
A7-R1	26	3495	139.3486
A12-R2	3	1348	54.4548
A15-R3	25	3095	121.7869
A19-R1	38	4340	186.6428

similar features of the neural network can be captured by fewer counterexamples whenever the neural network has fewer layers. This observation can be accounted for the fact that counterexamples that correspond to ReLUs in early layers are more powerful than those involves ReLUs in the later layers of the network.

In the second part of this experiment, we study the dependence of the number of feasible ReLU assignments on the choice of the workspace region. To that end, we fix the architecture of the neural network to one with 2 hidden layers and 40 neurons per layer. Table 4.3 reports the execution time, the number of counterexamples, and the number of feasible ReLU assignments across different regions of the workspace. In general, we observe that the number of feasible ReLU assignments increases with the size of the region.

4.6.3 Transition Feasibility

Following our verification streamline, the next step is to compute the transition function of the finite state abstraction $\delta_{\mathcal{F}}$, i.e., check transition feasibility between regions. Table 4.4 shows performance comparison between our proposed strategy that uses counterexamples obtained from pre-processing and the SMC encoding without preprocessing. We observe that the SMC encoding empowered by counterexamples, generated through the pre-processing

Table 4.4: Performance of the SMC-based encoding for computing $\delta_{\mathcal{F}}$ as a function of the neural network (timeout = 1 hour).

Number of Hidden Layers	**Total Number of Neurons**	**Time [s] (Exploit Counter-examples)**	**Time [s] (Without Counter-examples)**
1	82	0.5056	50.1263
	102	7.1525	timeout
	112	12.524	timeout
	122	18.0689	timeout
	132	20.4095	timeout
2	22	0.1056	15.8841
	42	4.8518	timeout
	62	3.1510	timeout
	82	2.6112	timeout
	102	11.0984	timeout
	122	3.8860	timeout
	142	0.7608	timeout
	162	2.7917	timeout
	182	193.6693	timeout
3	32	0.3884	388.549
	47	0.9034	timeout
	62	59.393	timeout

phase, scales more favorably compared to the ones that do not take counterexamples into account leading to 2-3 orders of magnitude reduction in the execution time. Moreover, and thanks to the pre-processing counter-examples, we observe that checking transition feasibility becomes less sensitive to changes in the neural network architecture as shown in Table 4.4.

Chapter 5

Two-Level Lattice Neural Network Architectures for Control of Nonlinear Systems

In this chapter, we consider the problem of automatically designing a Rectified Linear Unit (ReLU) Neural Network (NN) architecture (number of layers and number of neurons per layer) with the guarantee that it is sufficiently parametrized to control a nonlinear system. Whereas current state-of-the-art techniques are based on hand-picked architectures or heuristic based search to find such NN architectures, our approach exploits the given model of the system to design an architecture; as a result, we provide a guarantee that the resulting NN architecture is sufficient to implement a controller that satisfies an achievable specification. Our approach exploits two basic ideas. First, assuming that the system can be controlled by an unknown Lipschitz-continuous state-feedback controller with some Lipschitz constant upper-bounded by K_{cont}, we bound the number of affine functions needed to construct a Continuous Piecewise Affine (CPWA) function that can approximate the unknown Lipschitz-continuous controller. Second, we utilize the authors' recent results on a novel NN architecture named as the Two-Level Lattice (TLL) NN architecture, which was shown to be capable of implementing any CPWA function just from the knowledge of the number of affine functions that compromises this CPWA function. We evaluate our method on designing a NN architecture to control an inverted pendulum shows the efficiency of the proposed approach.

5.1 Introduction

Multilayer Neural Networks (NN) have shown tremendous success in realizing feedback controllers that can achieve several complex control tasks [22]. Nevertheless, the current state-of-the-art practices for designing these deep NN-based controllers are based on heuristics and hand-picked hyper-parameters (e.g., number of layers, number of neurons per layer, training parameters, training algorithm) without an underlying theory that guides their design. For example, several researchers have studied the problem of Automatic Machine Learning (AutoML) and in particular the problem of hyperparameter (number of layers, number of neurons per layer, and learning algorithm parameters) optimization and tuning in deep NN (see for example [103, 14, 101, 7, 110] and the references within). In this line of work, an iterative and exhaustive search through a manually specified subset of the hyperparameter space is performed. The best hyperparameters are then selected according to some performance metric without any guarantee on the correctness of the chosen architecture.

We focus on the fundamental question of how to systematically choose the NN architecture (number of layers and number of neurons per layer) such that we guarantee the correctness of the chosen NN architecture in terms of its ability to control a nonlinear dynamical system. In particular, we seek to use knowledge of the underlying control problem to guide the design of NN architectures [50].

Our approach exploits several insights. First, state-of-the-art NN utilizes Rectified Linear Units (ReLU), which in turn restricts the NN controller to implement only Continuous Piecewise Affine (CPWA) functions. As is widely known, CPWA function is compromised of several affine functions (named local linear functions), which are defined over a set of polytypic regions (called local linear regions). In other words, a ReLU NN—by virtue of its CPWA character—partitions its input space into a set of polytypic regions (named activation regions), and applies a linear controller at each of these regions. Therefore, a NN architecture dictates the number of such activation regions in the corresponding CPWA function that is represented by the trainable parameters in the NN. That is, to design a NN architecture, one needs to perform two steps: (i) compute (or upper bound) the number of activation regions required to implement a controller that satisfy the specifications and (ii) transform this number of activation regions into a NN architecture that is guaranteed to give rise to this number of activation regions.

To approach the first step, namely counting the number of the required activation regions, we assume the existence of an unknown robust Lipschitz-continuous, state-feedback controller with some Lipschitz constant upper-bounded by K_{cont} that is capable of controlling the system while meeting the specifications. Without the knowledge of such controller, other than the upper bound on its Lipschitz constant K_{cont}, we can upper-bound the number of activation regions needed to approximate this controller by a CPWA function while still meeting the same specifications.

Next, we build on recent results obtained by the authors on a novel NN architecture named Two-Level Lattice (TLL) NN architecture [48]. Unlike other NN architecture for which the number of activation regions is unknown *a priori*, the TLL-NN architecture enjoys the property that it is parametrized directly by the number of its activation regions. That is, once the number of activation regions is computed using the existence of such an unknown robust Lipschitz-continuous controller, a TLL-NN architecture can be directly generated from this knowledge. Such NN is then guaranteed to be sufficiently parametrized to implement a CPWA function that approximates the unknown Lipschitz-continuous controller, providing a systematic approach to design such architecture for NN controllers.

5.2 Abstract Disturbance Simulation

We will denote by $\mathbb{N}$, $\mathbb{R}$ and $\mathbb{R}^+$ the set of natural numbers, the set of real numbers and the set of non-negative real numbers, respectively. For a function $f : A \to B$, let $\text{dom}(f)$ return the domain of f, and let $\text{range}(f)$ return the range of f. For a set $V \in \mathbb{R}^n$, let $\text{int}(V)$ return the interior of V. For $x \in \mathbb{R}^n$, we will denote by $\|x\|$ the infinity norm of x; for $x \in \mathbb{R}^n$ and $\epsilon \geq 0$ we will denote by $B(x; \epsilon)$ the ball of radius ϵ centered at x as specified by $\|\cdot\|$. For $f : \mathbb{R}^n \to \mathbb{R}^m$, $\|f\|_\infty$ will denote the essential supremum norm of f. Finally, given two sets A and B denote by B^A the set of all functions with domain A and range B.

5.2.1 Dynamical Model

In this chapter, we will consider a continuous-time nonlinear dynamical system specified by the ordinary differential equation (ODE):

$$\dot{x}(t) = f(x(t), u(t)) \tag{5.1}$$

where the state vector $x(t) \in \mathbb{R}^n$, and the control vector $u(t) \in \mathbb{R}^m$. Formally, we have the following definition:

Definition 5.1 (Control System). *A control system is a tuple* $\Sigma = (X, U, \mathcal{U}, f)$ *where*

- $X \subseteq \mathbb{R}^n$ *is the compact state space;*

- *$U \subseteq \mathbb{R}^m$ is the compact set of admissible (instantaneous) controls;*
- *$\mathcal{U} \subseteq U^{\mathbb{R}^+}$ is the space of admissible open-loop control functions – i.e. $v \in \mathcal{U}$ is a function $v : \mathbb{R}^+ \to U$; and*
- *$f : \mathbb{R}^n \times U \to \mathbb{R}^n$ is a vector field specifying the time evolution of states according to (5.1).*

A control system is said to be (globally) Lipschitz if there exists constants K_x and K_u such that for all $x, x' \in \mathbb{R}^n$ and $u, u' \in \mathbb{R}^m$:

$$\|f(x,u) - f(x',u')\| \leq K_x\|x - x'\| + K_u\|u - u'\|. \tag{5.2}$$

In the sequel, we will primarily be concerned with solutions to (5.1) that result from instantaneous state-feedback controllers, $\Psi : X \to U$. Thus, we use $\zeta_{x_0\Psi}$ to denote the *closed-loop* solution of (5.1) starting from initial condition x_0 (at time $t = 0$) and using *state-feedback controller* Ψ. We refer to such a $\zeta_{x_0\Psi}$ as a (closed-loop) *trajectory* of its associated control system.

Definition 5.2 (Closed-loop Trajectory)**.** *Let Σ be a Lipschitz control system, and let $\Psi : \mathbb{R}^n \to U$ be a globally Lipschitz continuous function. A closed-loop trajectory of Σ under controller Ψ and starting from $x_0 \in X$ is the function $\zeta_{x_0\Psi} : \mathbb{R}^+ \to X$ that uniquely solves the integral equation:*

$$\zeta_{x_0\Psi}(t) = x_0 + \int_0^t f(\zeta_{x_0\Psi}(\sigma), \Psi(\zeta_{x_0\Psi}(\sigma)))d\sigma. \tag{5.3}$$

It is well known that such solutions exist and are unique under these assumptions [69]. We will only consider feedback controllers for which X is positively invariant under feedback, i.e. range$(\zeta_{x_0\Psi}) \subseteq X$.

For any given feedback controller, Ψ, the open-loop control functions created by its trajectories may not be elements of $\mathcal{U}$. Thus, we make the following additional definition:

Definition 5.3 (Feedback Controllable). *A Lipschitz control system Σ is feedback controllable by a Lipschitz controller $\Psi : \mathbb{R}^n \to U$ if the following is satisfied: $\Psi \circ \zeta_{x\Psi} \in \mathcal{U}$, $\forall x \in X$. A Lipschitz control system is called feedback controllable if it is feedback controllable for each globally Lipschitz feedback controller.*

In this chapter, we will henceforth consider only feedback controllable Lipschitz control systems. We conclude this subsection by defining the (sampled) transition system embedding of a feedback-controlled system that is inspired by the work in [174].

Definition 5.4 (τ-sampled Transition System Embedding). *Let $\Sigma = (X, U, \mathcal{U}, f)$ be a feedback controllable Lipschitz control system, and let $\Psi : \mathbb{R}^n \to U$ be a Lipschitz continuous feedback controller. For any $\tau > 0$, the τ-sampled transition system embedding of Σ under Ψ is the tuple $S_\tau(\Sigma_\Psi) = (X_\tau, \mathcal{U}_\tau, \xrightarrow[\Sigma_\Psi]{})$ where:*

- *$X_\tau = X$ is the state space;*
- *$\mathcal{U}_\tau = \{(\Psi \circ \zeta_{x_0\Psi})|_{t\in[0,\tau]} : x_0 \in X\}$ is the set of open loop control inputs generated by Ψ-feedback, each restricted to the domain $[0, \tau]$; and*
- *$\xrightarrow[\Sigma_\Psi]{} \subseteq X_\tau \times \mathcal{U}_\tau \times X_\tau$ such that $x \xrightarrow[\Sigma_\Psi]{u} x'$ iff both $u = (\Psi \circ \zeta_{x\Psi})|_{t\in[0,\tau]}$ and $x' = \zeta_{x\Psi}(\tau)$.*

5.2.2 Abstract Disturbance Simulation

In this subsection, we propose a new simulation relation, which we call *abstract disturbance simulation*, as a formal notion of specification satisfaction for metric transition systems. Abstract disturbance simulation enforces a notion of specification that is robust to perturbation of the state, and this will facilitate solving the main problem in this chapter.

Abstract disturbance simulation is inspired by robust bisimulation [80] and especially disturbance bisimulation [91], but it abstracts those notions away from their definitions in terms of control system embeddings and explicit modeling of disturbance inputs. In this way, it is conceptually similar to the technique used in [174] and [107] to define a quantized abstraction, where deliberate non-determinism is introduced in order to account for input errors. As a prerequisite, we introduce the following definition.

Definition 5.5 (Perturbed Metric Transition System). *Let $S = (X, U, \underset{S}{\longrightarrow})$ be a metric transition system where $X \subseteq X_M$ for some metric space (X_M, d). Then the δ-perturbed metric transition system of S, $\mathfrak{S}^\delta$, is a tuple $\mathfrak{S}^\delta = (X, U, \underset{\mathfrak{S}^\delta}{\longrightarrow})$ where the (altered) transition relation, $\underset{\mathfrak{S}^\delta}{\longrightarrow}$, is defined as follows:*

$$x \underset{\mathfrak{S}^\delta}{\overset{u}{\longrightarrow}} x' \qquad \textit{iff} \qquad \exists x'' \in X \ \textit{s.t.}\ d(x'', x') \leq \delta \ \textit{and}\ x \underset{S}{\overset{u}{\longrightarrow}} x''. \tag{5.4}$$

Note that $\mathfrak{S}^\delta$ has identical states and input labels to S, and it also subsumes all of the transitions therein, i.e. $\underset{S}{\longrightarrow} \subset \underset{\mathfrak{S}^\delta}{\longrightarrow}$. However, the transition relation for $\mathfrak{S}^\delta$ explicitly contains new nondeterminism relative to the transition relation of S. This nondeterminism can be thought of as perturbing the targets state of each transition in S; each such perturbation becomes the target of a (nondeterministic) transition with the same input label as the original transition.

With this definition in hand, we can finally define an abstract disturbance simulation between two metric transition systems.

Definition 5.6 (Abstract Disturbance Simulation). *Let $S = (X_S, U, \underset{S}{\longrightarrow})$ and $T = (X_T, U_T, \underset{T}{\longrightarrow})$ be metric transition systems whose state spaces X_S and X_T are subsets of the same metric space (X_M, d). Then T abstract-disturbance simulates S under disturbance δ, written $S \preceq_{\mathcal{AD}_\delta} T$ if there is a relation $R \subseteq X_S \times X_T$ such that*

1. *for every $(x, y) \in R$, $d(x, y) \leq \epsilon$;*

2. *for every $x \in X_S$ there exists a pair $(x, y) \in R$; and*

3. *for every $(x, y) \in R$ and $x \xrightarrow[\mathfrak{S}^\delta]{u} x'$ there exists a $y \xrightarrow[T]{v} y'$ such that $(x', y') \in R$.*

Remark 5.1. *$\preceq_{\mathcal{AD}_0}$ corresponds with the usual notion of simulation for metric transition systems. Thus, $S \preceq_{\mathcal{AD}_\delta} T \Leftrightarrow \mathfrak{S}^\delta \preceq_{\mathcal{AD}_0} T$.*

5.2.3 ReLU Neural Network Architectures

In this chapter, our primary focus will be on controlling the nonlinear system defined in (5.1) with a state-feedback neural network controller $\mathcal{NN} : X \to U$, where $\mathcal{NN}$ denotes a Rectified Linear Unit Neural Network (ReLU NN). Such a (K-layer) ReLU NN is specified by composing K *layer* functions (or just *layers*). A layer with $\mathfrak{i}$ inputs and $\mathfrak{o}$ outputs is specified by a $(\mathfrak{o} \times \mathfrak{i})$ real-valued matrix of *weights*, W, and a $(\mathfrak{o} \times 1)$ real-valued matrix of *biases*, b, as follows: $L_\theta : \mathbb{R}^{\mathfrak{i}} \to \mathbb{R}^{\mathfrak{o}}, z \mapsto \max\{Wz + b, 0\}$. where the max function is taken element-wise, and $\theta \triangleq (W, b)$ for brevity. Thus, a K-layer ReLU NN function as above is specified by K layer functions $\{L_{\theta^{(i)}} : i = 1, \ldots, K\}$ whose input and output dimensions are *composable*: that is they satisfy $\mathfrak{i}_i = \mathfrak{o}_{i-1} : i = 2, \ldots, K$. Specifically:

$$\mathcal{NN}(x) = (L_{\theta^{(K)}} \circ L_{\theta^{(K-1)}} \circ \cdots \circ L_{\theta^{(1)}})(x). \tag{5.5}$$

When we wish to make the dependence on parameters explicit, we will index a ReLU function $\mathcal{NN}$ by a *list of matrices* $\Theta \triangleq (\theta^{(1)}, \ldots, \theta^{(K)})$ [1].

Specifying the number of layers and the *dimensions* of the associated matrices $\theta^{(i)} = (\ W^{(i)}, b^{(i)}\ )$

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

specifies the *architecture* of the ReLU NN. Therefore, we will use:

$$\mathrm{Arch}(\Theta) \triangleq ((n, \mathfrak{o}_1), (\mathfrak{i}_2, \mathfrak{o}_2), \ldots, (\mathfrak{i}_{K-1}, \mathfrak{o}_{K-1}), (\mathfrak{i}_K, m)) \tag{5.6}$$

to denote the architecture of the ReLU NN $\mathscr{N\!N}_\Theta$.

Since we are interested in designing ReLU architectures, we will also need the following result from [48, Theorem 7], which states that a Continuous, Piecewise Affine (CPWA) function, f, can be implemented exactly using a Two-Level-Lattice (TLL) NN architecture that is parameterized exclusively by the number of local linear functions in f.

Definition 5.7 (Local Linear Function). *Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be CPWA. Then a local linear function of f is a linear function $\ell : \mathbb{R}^n \to \mathbb{R}^m$ if there exists an open set $\mathfrak{O}$ such that $\ell(x) = f(x)$ for all $x \in \mathfrak{O}$.*

Definition 5.8 (Linear Region). *Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be CPWA. Then a linear region of f is a the largest set $\mathfrak{R} \subseteq \mathbb{R}^n$ such that f has only one local linear function on the interior of $\mathfrak{R}$.*

Theorem 5.2 (Two-Level-Lattice (TLL) NN Architecture [9, Theorem 7]). *Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be a CPWA function, and let $\bar{N}$ be an upper bound on the number of local linear functions in f. Then there is a Two-Level-Lattice (TLL) NN architecture $\mathrm{Arch}(\Theta^{TLL}_{\bar{N}})$ parameterized by $\bar{N}$ and values of $\Theta^{TLL}_{\bar{N}}$ such that: $f(x) = \mathscr{N\!N}_{\Theta^{TLL}_{\bar{N}}}(x)$. In particular, the number of linear regions of f is such an upper bound on the number of local linear functions.*

Finally, note that a ReLU NN function, $\mathscr{N\!N}$, is known to be a continuous, piecewise affine (CPWA) function consisting of finitely many linear segments. Thus, $\mathscr{N\!N}$ is itself necessarily globally Lipschitz continuous.

5.3 Problem Formulation

We can now state the main problem we will consider in this chapter. In brief, we wish to identify the architecture for a ReLU network to be used as an instantaneous feedback controller for the control system Σ: this architecture must have parameter weights that allow it to control Σ up to a specification that can be met by some other, non-NN controller.

Despite our choice to consider fundamentally continuous-time models, we formulate our main problem in terms of their (τ-sampled) transition system embeddings. This choice reflects recent success in verifying specifications for such transition system embeddings by means of techniques adapted from computer science; see e.g. [148], where a variety of specifications are considered in this context, among them LTL formula satisfaction. Thus, our main problem is stated in terms of the simulation relations in the previous section.

Problem 5.3. *Let $\delta > 0$ and $K_{cont} > 0$ be given. Let Σ be a feedback controllable Lipschitz control system, and let $S_{spec} = (X_{spec}, U_{spec}, \xrightarrow[S_{spec}]{})$ be a transition system encoding for a specification on Σ. Finally, let $\tau = \tau(f, K_x, K_u, K_{cont}, \delta)$ be determined by the parameters specified.*

Now, suppose that there exists a Lipschitz continuous controller $\Psi : \mathbb{R}^n \to U$ with Lipschitz constant $K_\Psi \leq K_{cont}$ such that:

$$S_\tau(\Sigma_\Psi) \preceq_{\mathcal{AD}_\delta} S_{spec}. \tag{5.7}$$

Then the problem is to identify a ReLU architecture, $Arch(\Theta)$, with the property that there exists values for Θ such that:

$$S_\tau(\Sigma_{\mathcal{NN}_\Theta}) \preceq_{\mathcal{AD}_0} S_{spec}. \tag{5.8}$$

One of the primary assumptions is that there exists a controller Ψ which satisfies the specification, S_{spec}. We use this assumption largely to help ensure that the problem is well posed. For example, this assumption ensures that we aren't trying to assert the existence of NN controller for a system and specification that can't be achieved by *any* continuous controller – such examples are known to exist for nonlinear systems. In this way the existence of a controller Ψ subsumes any possible conditions of this kind that one might wish to impose: stabilizability or controllability for example.

Moreover, there is a strong conceptual reason to consider abstract disturbance simulation in specification satisfaction for such a Ψ. Our approach to solve this problem will be to design a NN architecture that can approximate any such Ψ sufficiently closely. However, $\mathcal{NN}_\Theta$ clearly belongs to a smaller class of functions than Ψ, so an arbitrary controller Ψ cannot, in general, be represented *exactly* by means of $\mathcal{NN}_\Theta$. This presents an obvious difficulty because instantaneous errors between Ψ and $\mathcal{NN}_\Theta$ may accumulate by means of the system dynamics, i.e. via (5.3).

5.4 ReLU Architectures for Nonlinear Systems

Before we state the main theorem of the chapter, we introduce the following notation in the form of two definitions.

Definition 5.9 (Vector Field Bound, $\mathcal{K}$). *Let $\mathcal{K} \triangleq \max_{x\in X, u\in U} \|f(x,u)\|$, which is well defined because $X \times U$ is compact and f is continuous.*

Definition 5.10 (Extent of X). *The extent of the compact set X is defined as:*

$$ext(X) \triangleq \max_{k=1,\dots,n} \left| \max_{x\in X} \pi_k(x) - \min_{x\in X} \pi_k(x) \right|, \tag{5.9}$$

where $\pi_k(x)$ is the projection of x onto its k^{th} component.

The main result of the chapter is the following theorem, which directly solves 5.3.

Theorem 5.4 (ReLU Architecture). *Let $\delta > 0$ and $K_{cont} > 0$ be given, and let Σ and S_{spec} be as in the statement of 5.3. Finally, choose a $\mu > 0$ such that:*

$$K_u \cdot \mu \cdot \frac{\mu}{6 \cdot K_{cont} \cdot \mathcal{K}} \cdot e^{K_x \frac{\mu}{6 \cdot K_{cont} \cdot \mathcal{K}}} < \delta, \tag{5.10}$$

and set:

$$\tau \leq \frac{\mu}{6 \cdot K_{cont} \cdot \mathcal{K}} \quad \text{and} \quad \eta \leq \frac{\mu}{6 \cdot K_{cont}}, \tag{5.11}$$

(which depend only on f, K_x, K_u, K_{cont} and δ).

If there exists a Lipschitz continuous controller $\Psi : \mathbb{R}^n \to U$ with Lipschitz constant $K_\Psi \leq K_{cont}$ such that:

$$S_\tau(\Sigma_\Psi) \preceq_{\mathcal{AD}_\delta} S_{spec}. \tag{5.12}$$

Then a TLL NN architecture $Arch(\Theta_N^{TLL})$ of size:

$$N \geq m \cdot \left(n! \cdot \sum_{k=1}^{n} \frac{2^{2k-1}}{(n-k)!} \right) \cdot \left(\frac{ext(X)}{\eta} \right)^n \tag{5.13}$$

has the property that there exist values for Θ_N^{TLL} such that:

$$S_\tau(\Sigma_{\mathcal{NN}_{\Theta_N^{TLL}}}) \preceq_{\mathcal{AD}_0} S_{spec}. \tag{5.14}$$

Proof Sketch:

The proof of 5.4 consists of establishing the following two implications:

- Approximate controllers satisfy the specification: There is an approximation accuracy,

μ, and sampling period, τ, with the following property: if the unknown controller Ψ satisfies the specification (under δ disturbance and sampling period τ), then any controller (NN or otherwise) which approximates Ψ to accuracy μ will also satisfy the specification (but under no disturbance). This implication is shown in 5.10 of 5.5.

- Any controller can be approximated by a CPWA with the same fixed number of linear regions: If unknown controller Ψ has a Lipschitz constant $K_\Psi \leq K_{\text{cont}}$, then Ψ can be approximated by a CPWA with a number of regions that depends only on K_{cont} and the desired approximation accuracy. This implication is shown in 5.13 of 5.6.

We will show these results for *any* controller Ψ that satisfies the assumptions of 5.4. Thus, these results together show the following implication: if there *exists* a controller Ψ that satisfies the assumptions of 5.4, then there is a CPWA controller that satisfies the specification. And moreover, this CPWA controller has a at most a number of linear regions that depends only on the parameters of the problem *and not the particular controller* Ψ.

The conclusion of the theorem will then follow directly from 5.2 [48, Theorem 7]: together, they specify that any CPWA with the same number of linear regions (or fewer) can be implemented exactly by a common TLL NN architecture.

5.5 Approximate Controllers Satisfy the Specification

The goal of this section is to choose constants $\mu > 0$ and $\tau > 0$ such that *any* controller Υ with $\|\Upsilon - \Psi\|_\infty \leq \mu/3$ satisfies the specification $S_\tau(\Sigma_\Upsilon) \preceq_{\mathcal{AD}_0} S_{\text{spec}}$. The approach will be as follows. First, we confine ourselves to a region in the state space on which the controller Ψ doesn't vary much: the size of this region is determined entirely by the approximation accuracy, μ, and the bound on the Lipschitz constant, K_{cont}. Then we confine the *trajectories* of Σ_Ψ to this region by bounding the duration of those trajectories, i.e. τ.

Finally, we feed these results into a Grönwall-type bound to choose μ. In particular, we choose μ small enough such that the error incurred by using Υ instead of Ψ is within the disturbance robustness, δ. From this we will conclude that Υ satisfies the specification as claimed whenever $\|\Upsilon - \Psi\| \leq \mu/3$. A more detailed road map of these steps is as follows.

- Let μ be an approximation error. Then:

  1. Choose $\eta = \eta(\mu)$ such that a Lipschitz function with constant K_{cont} doesn't vary by more than $\mu/3$ between any two points that are 2η apart.
  2. Choose $\tau = \tau(\mu)$ such that $\|x - \xi_{xv}(\tau)\| \leq \eta$ for any continuous open-loop control v (use the fact that $\|f\|$ is bounded).
  3. Use a) and b) to conclude that $\|\Upsilon(\zeta_{x\Upsilon}(t)) - \Psi(\zeta_{x\Psi}(t))\| \leq \|\Upsilon - \Psi\|_\infty + 2\mu/3$ for $t \in [0, \tau]$
  4. Assume $\|\Upsilon - \Psi\|_\infty \leq \mu/3$. Choose $\mu = \mu(\delta)$ such that a Grönwall-type bound satisfies:

$$\|\zeta_{x\Upsilon}(\tau(\mu)) - \zeta_{x\Psi}(\tau(\mu))\| \leq K_u \cdot \mu \cdot \tau(\mu) \cdot e^{K_x \tau(\mu)} < \delta. \tag{5.15}$$

  Conclude that if $\|\Upsilon - \Psi\|_\infty \leq \mu/3$, then: $S_{\tau'}(\Sigma_\Upsilon) \preceq_{\mathcal{AD}_0} \mathfrak{S}_{\tau'}(\Sigma_{\Psi'}) \preceq_{\mathcal{AD}_0} S_{\text{spec}}$.

Now we proceed with the proof. We first formalize Steps *i)*, *ii)* and *iii)* in the next three propositions.

Proposition 5.5. *Let $\mu > 0$ be given, and let Ψ be as above. Then there exists an $\eta = \eta(\mu)$ such that:*

$$\|x - x'\| \leq 2\eta \implies \|\Psi(x) - \Psi(x')\| \leq \mu/3. \tag{5.16}$$

Proposition 5.6. *Let $\mu > 0$ be given, and let $\eta = \eta(\mu)$ be as in the previous proposition. Finally, let Σ be as specified in the statement of 5.4. Then there exists a $\tau = \tau(\mu)$ such that for any Lipschitz feedback controller Υ:*

$$\|x - \zeta_{x\Upsilon}(t)\| \leq \eta = \eta(\mu) \quad \forall t \in [0, \tau]. \tag{5.17}$$

Proof. Since f is continuous on the compact set $X \times U$, it is bounded, and let $\mathcal{K}$ be this bound as stated in 5.9. Then by (5.3) we have

$$\begin{aligned}
\|x - \zeta_{x\Upsilon}(t)\| &= \left\| \int_0^t f(\zeta_{x\Upsilon}(\sigma), \Upsilon(\zeta_{x\Upsilon}(\sigma)))d\sigma \right\| && (5.18)\\
&\leq \int_0^t \|f(\zeta_{x\Upsilon}(\sigma), \Upsilon(\zeta_{x\Upsilon}(\sigma)))\| \, d\sigma && (5.19)\\
&\leq \int_0^t \mathcal{K} d\sigma = \mathcal{K}t. && (5.20)
\end{aligned}$$

Hence, choose $\tau = \tau(\mu) \leq \frac{\eta(\mu)}{\mathcal{K}}$ and the conclusion follows. □

Proposition 5.7. *Let $\mu > 0$ be given. Let Σ and Ψ be as in the statement of 5.4; let $\eta = \eta(\mu)$ be as in 5.5; let $\tau = \tau(\mu)$ be as in 5.6; and let $\Upsilon : \mathbb{R}^n \to U$ be a Lipschitz continuous function. Then:*

$$\forall t \in [0, \tau] \quad \|\Upsilon(\zeta_{x\Upsilon}(t)) - \Psi(\zeta_{x\Psi}(t))\| \leq \|\Upsilon - \Psi\|_\infty + 2\mu/3 \tag{5.21}$$

Proof. By the triangle inequality, we have:

$$\begin{aligned}
&\|\Upsilon(\zeta_{x\Upsilon}(t)) - \Psi(\zeta_{x\Psi}(t))\|\\
&\quad = \|\Upsilon(\zeta_{x\Upsilon}(t)) - \Psi(\zeta_{x\Upsilon}(t)) + \Psi(\zeta_{x\Upsilon}(t)) - \Psi(\zeta_{x\Psi}(t))\|\\
&\quad = \|\Upsilon(\zeta_{x\Upsilon}(t)) - \Psi'(\zeta_{x\Upsilon}(t)) + \Psi(\zeta_{x\Upsilon}(t)) - \Psi(x) + \Psi(x) - \Psi(\zeta_{x\Psi}(t))\|\\
&\quad \leq \|\Upsilon(\zeta_{x\Upsilon}(t)) - \Psi(\zeta_{x\Upsilon}(t))\| + \|\Psi(\zeta_{x\Upsilon}(t)) - \Psi(x)\| + \|\Psi(x) - \Psi(\zeta_{x\Psi}(t))\|. \qquad (5.22)
\end{aligned}$$

The first term in (5.22) is bounded by $\|\Upsilon - \Psi\|_\infty$. Now consider the second term. By 5.6, $\|\zeta_{x\Upsilon}(t) - x\| \leq \eta$; thus, by 5.5 we conclude that $\|\Psi(\zeta_{x\Upsilon}(t)) - \Psi(x)\| \leq \mu/3$. The final term is likewise bounded by $\mu/3$ for the same reasons. Thus, the conclusion follows. $\square$

To prove Step *iv)* we first need the following two results.

Proposition 5.8 (Grönwall Bound)**.** *Let Σ and Ψ be as in the statement of 5.4, and let Υ be as in the statement of 5.7. If:*

$$\|\Upsilon(\zeta_{x\Upsilon}(t)) - \Psi(\zeta_{x\Psi}(t))\| \leq \kappa \quad \forall t \in [0, \tau] \tag{5.23}$$

then: $\|\zeta_{x\Upsilon}(t) - \zeta_{x\Psi}(t)\| \leq K_u \cdot \kappa \cdot t \cdot e^{K_x t} \quad \forall t \in [0, \tau]$.

Proof. By definition and the properties of the integral, we have:

$$\begin{aligned}
&\|\zeta_{x\Upsilon}(t) - \zeta_{x\Psi}(t)\| \\
&= \|\int_0^t f(\zeta_{x\Upsilon}(\sigma), \Upsilon(\zeta_{x\Upsilon}(\sigma))) - f(\zeta_{x\Psi}(t), \Psi(\zeta_{x\Psi}(t))) d\sigma\| \\
&\leq \int_0^t \|f(\zeta_{x\Upsilon}(\sigma), \Upsilon(\zeta_{x\Upsilon}(\sigma))) - f(\zeta_{x\Psi}(t), \Psi'(\zeta_{x\Psi}(t)))\| d\sigma \\
&\leq \int_0^t K_x \|\zeta_{x\Upsilon}(\sigma) - \zeta_{x\Psi}(\sigma)\| + K_u \|\Upsilon(\zeta_{x\Upsilon}(\sigma)) - \Psi(\zeta_{x\Psi}(\sigma))\| d\sigma \\
&\leq \int_0^t K_x \|\zeta_{x\Upsilon}(\sigma) - \zeta_{x\Psi}(\sigma)\| + K_u \cdot \kappa \; d\sigma.
\end{aligned} \tag{5.24}$$

The claimed bound now follows directly from the Grönwall Inequality [69]. $\square$

Lemma 5.9. *Let Σ, Ψ and Υ be as before. Also, suppose that $\mu' > 0$ is such that:*

$$K_u \cdot \mu \cdot \frac{\mu}{6 \cdot K_{cont} \cdot \mathcal{K}} \cdot e^{K_x \frac{\mu}{6 \cdot K_{cont} \cdot \mathcal{K}}} < \delta. \tag{5.25}$$

If $\|\Upsilon - \Psi\|_\infty \leq \mu/3$, then: $\|\zeta_{x\Upsilon}(\tau(\mu)) - \zeta_{x\Psi}(\tau(\mu))\| \leq \delta$.

Proof. This is a direct consequence of applying 5.7 to 5.8. □

The final result in this section is the following Lemma.

Lemma 5.10. *Let Σ, Ψ and Υ be as before, and suppose that $\mu > 0$ is such that:*

$$K_u \cdot \mu \cdot \frac{\mu}{6 \cdot K_{cont} \cdot \mathcal{K}} \cdot e^{K_x \frac{\mu}{6 \cdot K_{cont} \cdot \mathcal{K}}} < \delta. \tag{5.26}$$

If $\|\Upsilon - \Psi\|_\infty \leq \mu/3$, then for $\tau \leq \frac{\mu}{6 \cdot K_{cont} \cdot \mathcal{K}}$ we have:

$$S_\tau(\Sigma_\Upsilon) \preceq_{\mathcal{AD}_0} \mathfrak{S}_\tau(\Sigma_\Psi). \tag{5.27}$$

And hence: $S_\tau(\Sigma_\Upsilon) \preceq_{\mathcal{AD}_0} S_{spec}$.

Proof. By definition, $S_\tau(\Sigma_\Upsilon)$ and $\mathfrak{S}_\tau(\Sigma_\Psi)$ have the same state spaces, X. Thus we propose the following as an abstract disturbance simulation under 0 disturbance (i.e. a conventional simulation for metric transition systems): $R = \{(x, x) | x \in X\}$.

Clearly, R satisfies the property that for all $(x, y) \in R$, $d(x, y) \leq 0$, and for every $x \in X$, there exists an $y \in X$ such that $(x, y) \in R$. Thus, it only remains to show the third property of 5.6 under 0 disturbance.

In particular, let $(x, x) \in R$, and suppose that $x \xrightarrow[\Sigma_\Upsilon]{\Upsilon \circ \zeta_{x\Upsilon}} x' = \zeta_{x\Upsilon}(\tau)$. If $x \xrightarrow[\Sigma_\Psi]{\Psi \circ \zeta_{x\Psi}} x'$, then we have shown that R is an abstract disturbance simulation under 0 disturbance. But by definition, $x \xrightarrow[\Sigma_\Psi]{\Psi \circ \zeta_{x\Psi}} x'' = \zeta_{x\Psi}(\tau)$, and by 5.9, $\|x' - x''\| \leq \delta$. Thus, by definition of $\mathfrak{S}_\tau(\Sigma_\Psi)$, $x \xrightarrow[\Sigma_\Psi]{\Psi \circ \zeta_{x\Psi}} x'$ as required. □

5.6 CPWA Approximation of a Controller

The results in 5.5 showed that any controller, Υ, whether it is CPWA or not, will satisfy the specification if it is close to Ψ in the sense that $\|\Upsilon - \Psi\|_\infty \leq \mu/3$ (where μ is as specified therein). Thus, the main objective of this section will be to show that an arbitrary Ψ can be approximated to this accuracy by a CPWA controller, Υ_{CPWA}, subject to the following caveat. It is well known that CPWA functions are good function approximators in general, but we have to keep in mind our eventual use of 5.2: thus, we need to approximate *any such* Ψ by a CPWA with the same, bounded number of linear regions. Thus, our objective in this section is to find not just a controller Υ_{CPWA} that approximates Ψ to the specified accuracy, but one that achieves this objective using not more than some common, fixed number of linear regions that depends only on the problem parameters (and not the function Ψ itself which is assumed to be unknown except for a bound on its Lipschitz constant).

With this objective in mind, our strategy will be to partition the state space X into a grid of sup-norm balls such that no Ψ can vary by much between them: indeed we will use balls of size η, as specified in 5.5. Thus, we propose the following starting point: inscribe a slightly smaller ball within each η ball of the partition, and choose the value of Υ_{CPWA} on each such ball to be a constant value equal to $\Psi(x)$ for some x therein. Because we have chosen the size of the partition to be small, such an Υ_{CPWA} will still be a good approximation of Ψ for these points in its domain. Using this approach, then, we only have to concern ourselves with how "extend" a function so defined to the entire space X as a CPWA. Moreover, note that this procedure is actually independent of the particular Ψ chosen, despite appearances: we are basing our construction on a grid size η that depends only on the problem parameters, and the construction will work no matter what particular value $\Psi(x)$ has within each grid square.

The first step in this procedure will be to show how to extend such a function over the largest-dimensional "gaps" between the smaller inscribed balls; the blue region depicted in 5.1 is an example of this large-dimensional gap for $X \subset \mathbb{R}^2$ (the notation in the figure will be explained later). This result must control the error of the extension so as to preserve our desired approximation bound, as well provide a count of the number of linear regions necessary to do so; this is 5.12. This result can then be extended to all of the other gaps between inscribed balls to yield a CPWA function with domain X, approximation error $\mu/3$, and a known number of regions; this is 5.13.

In order to prove our first lemma of this section, we need a couple of definitions to help with the terminology.

Definition 5.11 (Face). *Let $C = [0,1]^n$ be a unit hypercube of dimension n. A set $F \subseteq C$ is a k-dimensional face of C if there exists a set $J \subseteq \{1, \ldots, n\}$ such that $|J| = n - k$ and*

$$\forall x \in F \ . \ \bigwedge_{j \in J} \pi_j(x) \in \{0,1\}. \tag{5.28}$$

Let $\mathscr{F}_k(C)$ denote the set of k-dimensional faces of C, and let $\mathscr{F}(C)$ denote the set of all faces of C (of any dimension).

Remark 5.11. *A k-dimensional face of the hypercube $C = [0,1]^n$ is isomorphic to the hypercube $[0,1]^k$.*

Definition 5.12 (Corner). *Let $C = [0,1]^n$. A corner of C is a 0-dimensional face of C.*

Lemma 5.12. *Let $C = [0,1]^n$, and suppose that $\Gamma_c : \mathscr{F}_0(C) \to \mathbb{R}$ is a function defined on the corners of C. Then there is a CPWA function $\Gamma : C \to \mathbb{R}$ such that:*

- *$\forall x \in \mathscr{F}_0(C) . \Gamma(x) = \Gamma_c(x)$, i.e. Γ extends Γ_c to C;*
- *Γ has at most $2^{n-1} \cdot n!$ linear regions; and*
- *for all $x \in C$, $\min_{x \in \mathscr{F}_0(C)} \Gamma_c(x) \leq \Gamma(x) \leq \max_{x \in \mathscr{F}_0(C)} \Gamma_c(x)$.*

Proof. First, we assume without loss of generality that the given function Γ_c takes distinct values on each element of its domain.

This is a proof by induction on dimension. In particular, we will use the following induction hypothesis:

- There is a function $\Gamma_k : \cup_{i=1}^{k} \mathscr{F}_i(C) \to \mathbb{R}$ such that for all $\tilde{F} \in \mathscr{F}_k(C)$, $\Gamma_k|_{\tilde{F}}$ has the following properties:
  - it is CPWA
  - it has at most $2^{k-1} \cdot k!$ linear regions; and
  - for all $x \in \tilde{F}$: $\min_{x \in \mathscr{F}_0(\tilde{F})} \Gamma_c(x) \leq \Gamma_k(x) \leq \max_{x \in \mathscr{F}_0(\tilde{F})} \Gamma_c(x)$.

We start by showing that if the induction hypothesis above holds for k, then it also holds for $k+1$.

To show the induction step, first note that for any face $F \in \mathscr{F}_{k+1}(C)$, all of *its* faces are already in the domain of Γ_k. That is $\cup_{i=1}^{k} \mathscr{F}_i(F) \subseteq \mathrm{dom}(\Gamma_k)$. Thus, we can define Γ_{k+1} by extending Γ_k to $\mathrm{int}(F)$ for each $F \in \mathscr{F}_{k+1}(C)$. Since these interiors are mutually disjoint, we can do this by explicit construction on each individually, in such a way that the desired properties hold.

In particular, let $F \in \mathscr{F}_{k+1}(C)$, and let ν be the midpoint of F, i.e. the k-cube isomorphism of ν is $[\frac{1}{2}, \ldots, \frac{1}{2}]$. ν is clearly in the interior of F, so define:

$$\Gamma_{k+1}(\nu) = \frac{1}{|\mathscr{F}_0(F)|} \sum_{x \in \mathscr{F}_0(F)} \Gamma_k(x) \tag{5.29}$$

and note that the corners of F are also corners of C Thus, $\Gamma_{k+1}(\nu)$ is the average of all of the corners of the $k+1$-face that contains it. Now, extend Γ_{k+1} to the rest of $\mathrm{int}(F)$ as follows:

let $b \in \cup_{i=1}^{k} \mathscr{F}_i(F)$ and define:

$$\Gamma_{k+1}(\lambda \cdot \nu + (1-\lambda) \cdot b) = \lambda \cdot \Gamma_{k+1}(\nu) + (1-\lambda) \cdot \Gamma_k(b) \quad \forall \lambda \in [0,1]. \tag{5.30}$$

This definition clearly covers $\text{int}(F)$, and it also satisfies the requirement that:

$$\forall x \in F \ \min_{x \in \mathscr{F}_0(F)} \Gamma_c(x) \leq \Gamma_{k+1}(x) \leq \max_{x \in \mathscr{F}_0(F)} \Gamma_c(x) \tag{5.31}$$

because the induction hypothesis ensures that each b is on a face of F, and the corners of a face of F are a subset of the corners of F. Thus, it remains to show the bound on the number of linear regions. But from the construction, $\Gamma_{k+1}|_F$ has one linear region for linear region of Γ_k on a k-face of F. Since the $k+1$-face F has $2 \cdot (k+1)$ k-faces, we conclude by the induction hypothesis that $\Gamma_{k+1}|_F$ has at most:

$$2 \cdot (k+1) \cdot 2^{k-1} \cdot k! = 2^k \cdot (k+1)! \tag{5.32}$$

linear regions. This completes the proof of the induction step.

It remains only to show a base case. For this, we select $k = 1$, i.e. the line-segment faces of C. Each 1-face of C has only two corners and no other faces other than itself. Thus, for each $F \in \mathscr{F}_0(C)$ we can simply define $\Gamma_1|_F$ to linearly interpolate between those two corners. $\Gamma_1|_F$ is thus CPWA, and it satisfies the required bounds on its values. Moreover, $\Gamma_1|_F$ has exactly $2^{1-1} \cdot 1! = 1$ linear region. Thus, the function Γ_1 so defined satisfies the induction hypothesis stated above. $\square$

Definition 5.13 (η-partition). *Let $\eta > 0$ be given. Then an η-partition of X is a regular, non-overlapping grid of $\eta/2$ balls in the* sup *norm that partitions X. Let X_{cent} denote the set of centers of these balls, and let $X_{part} = \{B(x_c; \eta/2) | x_c \in X_{cent}\}$ denote the partition.*

Definition 5.14 (Neighboring Grid Center/Square). *Let X_{part} be an η-partition of X, and* *let $B(x_c;\eta/2) \in X_{part}$. Then a neighboring grid center (resp. square) to x_c is an $x'_c \in X_c$ (respectively $B(x'_c;\eta/2) \in X_{part}$) such that $B(x'_c;\eta/2)$ shares a face (of any dimension) with $B(x_c;\eta/2)$. The set of neighbors of a center, x_c, will be denoted by $\mathcal{N}(x_c)$.*

Lemma 5.13. *Let $\eta = \eta(\mu)$ be chosen as in 5.5, and let Ψ be as before. Then there is a CPWA function $\Upsilon_{\scriptscriptstyle CPWA} : \mathbb{R}^n \to U$ such that:*

- $\|\Upsilon_{\scriptscriptstyle CPWA} - \Psi\|_\infty \leq \frac{\mu}{3}$*; and*
- $\Upsilon_{\scriptscriptstyle CPWA}$ *has at most*

$$m \cdot \left(n! \cdot \sum_{k=1}^{n} \frac{2^{2k-1}}{(n-k)!} \right) \cdot \left(\frac{ext(X)}{\eta} \right)^n \tag{5.33}$$

linear regions.

Proof. Our proof will assume that $U \subseteq \mathbb{R}$, since the extension to $m > 1$ is straightforward from the $m = 1$ case. The basic proof will be to create an η-partition of X, and define Υ_{CPWA} to be constant on $\rho \cdot \eta/2 < \eta/2$ radius balls centered at each of the grid centers in the partition; we will then use 5.12 to "extend" this function to the rest of X as a CPWA function. In particular, for each $x_c \in X_C$, we start by defining:

$$\Upsilon_{\text{CPWA}}(x) = \Psi(x_c) \quad \forall x \in B(x_c; \rho \cdot \eta/2). \tag{5.34}$$

Then we will extend this function to the rest of X, and prove the claims for that extension.

To simplify the proof, we will henceforth focus on a particular x_c, and show how to extend Υ_{CPWA} from $B(x_c; \rho\cdot\eta/2)$ to the "gaps" between it and each of the neighboring balls, $B(x'_c; \rho \cdot \eta/2)$ for $x'_c \in \mathcal{N}(x_c)$. To further simplify the proof, we define here two additional pieces of notation. First, for each $x_c \in X_c$ and each $k \in \{1, \ldots, n\}$ define a function $\omega_k^{(x_c)}$ as follows:

$$\begin{aligned}
&\omega_k^{(x_c)} : \{-1, 0, +1\} \to 2^{\mathbb{R}} \\
&\omega_k^{(x_c)} : \quad 0 \mapsto [\pi_k(x_c) - \rho\tfrac{\eta}{2},\ \pi_k(x_c) + \rho\tfrac{\eta}{2}] \\
&\omega_k^{(x_c)} : +1 \mapsto [\pi_k(x_c) + \rho\tfrac{\eta}{2}, \pi_k(x_c) + \tfrac{\eta}{2} + (1-\rho)\tfrac{\eta}{2}] \\
&\omega_k^{(x_c)} : -1 \mapsto [\pi_k(x_c) - \tfrac{\eta}{2} - (1-\rho)\tfrac{\eta}{2}, \pi_k(x_c) - \rho\tfrac{\eta}{2}].
\end{aligned}$$

Then, define the function:

$$\begin{aligned}
&\mathcal{R}^{(x_c)} : \{-1, 0, 1\}^n \to 2^{\mathbb{R}^n} \\
&\mathcal{R}^{(x_c)} : \iota \mapsto \omega_1^{(x_c)}(\pi_1(\iota)) \times \omega_2^{(x_c)}(\pi_2(\iota)) \times \cdots \times \omega_n^{(x_c)}(\pi_n(\iota)),
\end{aligned}$$

and let $\mathbf{0} \triangleq (0, \ldots, 0) \in \{-1, 0, 1\}^n$. Also, define $\dim(\iota)$ as the number of non-zero elements in ι.

Now let $x_c \in X_c$ be fixed. Using the above notation, the ball $B(x_c; \rho \cdot \eta/2)$ is given by:

$$B(x_c; \rho \cdot \eta/2) = \mathcal{R}^{(x_c)}(\mathbf{0}), \tag{5.35}$$

Similarly each of the "gaps" between $\mathcal{R}^{(x_c)}(\mathbf{0})$ and its neighbors, $\mathcal{R}^{(x_c')}(\mathbf{0})$ for $x_c' \in \mathcal{N}(x_c)$, are the hypercubes: $\mathcal{R}^{(x_c)}(\iota)$ for $\iota \in \{-1, 0, 1\}^n \backslash \{\mathbf{0}\}$, and hence:

$$\bigcup_{x_c \in X_c, \iota \in \{-1,0,1\}^n \backslash \mathbf{0}} \mathcal{R}^{(x_c)}(\iota) = X \backslash \bigcup_{x_c' \in X_c} B(x_c'; \rho \cdot \eta/2). \tag{5.36}$$

This notation is illustrated in two dimensions in 5.1.

The first step is to show that Υ_{CPWA} can be extended from the constant-valued region, $\mathcal{R}^{(x_c)}(\mathbf{0})$, to each of its neighbors, $\mathcal{R}^{(x_c)}(\iota)$, in a consistent way as a CPWA. To do this, first note that $\mathcal{R}^{(x_c)}(\mathbf{0})$ has 2^n neighboring regions with indices $\iota' \in \{-1, +1\}^n$, and each of

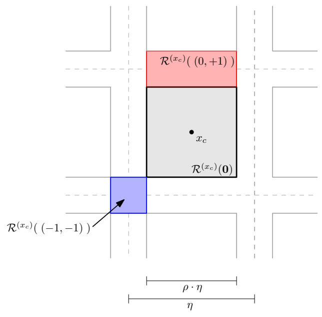

$\mathcal{R}^{(x_c)}(\,(0,+1)\,)$
x_c
$\mathcal{R}^{(x_c)}(\mathbf{0})$
$\mathcal{R}^{(x_c)}(\,(-1,-1)\,)$
$\rho\cdot\eta$
η

Figure 5.1: Illustration of $\mathcal{R}^{(x_c)}$ notation for $X \subset \mathbb{R}^2$. For x_c as labeled, the regions $\mathcal{R}^{(x_c)}((-1,-1))$, $\mathcal{R}^{(x_c)}((0,+1))$ and $\mathcal{R}^{(x_c)}(\mathbf{0})$ are shown in blue, red and light gray, respectively.

these regions intersects a different $\mathcal{R}^{(x_c')}(\mathbf{0})$ for $x_c' \in \mathcal{N}(x_c)$ at each corner, but is otherwise disjoint from them. Thus, 5.12 can be used to define a CPWA on each such $\mathcal{R}^{(x_c)}(\iota')$ in a way that is consistent with the definition of Υ_{CPWA} on the $\mathcal{R}^{(x_c')}(\mathbf{0})$. These definitions are also consistent with each other, since these regions are disjoint. Moreover, this procedure yields the same extension when started from $x_c' \in \mathcal{N}(x_c)$ instead of x_c (by the symmetric way that 5.12 is proved). Thus, it remains only to define Υ_{CPWA} on regions with indices of the form $\iota'' \in \{-1,0,+1\}^n \backslash \{-1,+1\}^n \cup \{\mathbf{0}\}$. However, each such $\mathcal{R}^{(x_c)}(\iota'')$ intersects $2^{n-\dim(\iota'')}$ regions with indices of the form $\iota' \in \{-1,+1\}^n$, and each of those intersections is a $\dim(\iota'')$ face of the corresponding $\mathcal{R}^{(x_c)}(\iota')$. But on each such $\dim(\iota'')$ face, Υ_{CPWA} is defined and agrees with $\Gamma_{\dim(\iota'')}$ from the construction in 5.12. Finally, since $\Gamma_{\dim(\iota'')}$ (and hence Υ_{CPWA}) is identical up to isomorphism on each of these $\dim(\iota'')$ faces, Υ_{CPWA} can be extended on to $\mathcal{R}^{(x_c)}(\iota'')$ by isomorphism between the $\dim(\iota'')$ nonzero indices, and Υ_{CPWA} as defined on one of the $\dim(\iota'')$ faces of $\mathcal{R}^{(x_c)}(\iota')$. Finally, the symmetry of this procedure and 5.12 ensures that this assignment will be consistent when starting from some $x_c' \in \mathcal{N}(x_c)$ instead of x_c.

Next, we show that for this Υ_{CPWA}, $\|\Upsilon_{\text{CPWA}} - \Psi\| \le \mu/3$. This largely follows from the interpolation property proven in 5.12. In particular, on some $\mathcal{R}^{(x_c)}(\iota)$, Υ_{CPWA} takes exactly the same values as some $\Gamma_{\dim(\iota)}$ constructed according to 5.12, where the interpolation happens between $\dim(\iota)$ points in $V \triangleq \{\Psi(x_c')|x_c' \in \mathcal{N}(x_c) \cup \{x_c\}\}$. Thus,

$$\forall x \in \mathcal{R}^{(x_c)}(\iota) \quad \min_{y \in V} \Psi(y) \leq \Upsilon_{\text{CPWA}}(x) \leq \max_{y \in V} \Psi(y). \tag{5.37}$$

Let $x \in \mathcal{R}^{(x_c)}(\iota)$ be fixed temporarily, and suppose that $\Upsilon_{\text{CPWA}}(x) - \Psi(x) \geq 0$ and $\max_{y \in V} \Psi(y) - \Psi(x) \geq 0$. Then:

$$|\Upsilon_{\text{CPWA}}(x) - \Psi(x)| = \Upsilon_{\text{CPWA}}(x) - \Psi(x) \leq \max_{y \in V} \Psi(y) - \Psi(x) = |\max_{y \in V} \Psi(y) - \Psi(x)| \leq \frac{\mu}{3} \tag{5.38}$$

where the last inequality follows from our choice of η from 5.5, since $|y - x| \leq 2\eta$ for all $y \in V$. The other cases can be considered as necessary, and they lead to the same conclusion. Hence, we conclude $\|\Upsilon_{\text{CPWA}} - \Psi\|_\infty \leq \mu/3$, since our choice of center x_c and ι was arbitrary.

Now we just need to (over)-count the number of linear regions needed in the extension Υ_{CPWA}. This too will follow from the construction in 5.12. Note that on each $\mathcal{R}^{(x_c)}(\iota)$, Υ_{CPWA} has the same number of linear regions as some $\Gamma_{\dim(\iota)}$ that was constructed by 5.12, which by the same lemma has $2^{\dim(\iota)-1} \cdot \dim(\iota)!$ regions. Thus, we count at most:

$$\sum_{k=1}^{n} \binom{n}{k} \cdot 2^k \cdot 2^{k-1} \cdot k! = n! \cdot \sum_{k=1}^{n} \frac{2^{2k-1}}{(n-k)!} \tag{5.39}$$

linear regions. Finally, since we need this many regions for the neighboring regions of a single grid square, we obtain an upper bound for the total number of regions by multiplying (5.39) by the number of grid squares in the partition, $(\frac{\text{ext}(X)}{\eta})^n$ (then by the m, in the multi-dimensional output case). $\square$

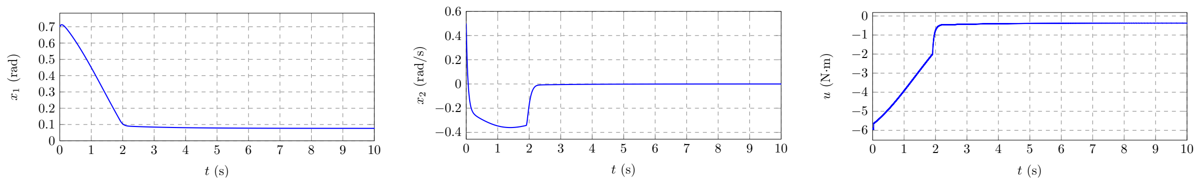


Figure 5.2: States and inputs of the inverted pendulum with initial condition $[0.7, 0.5]^T$.

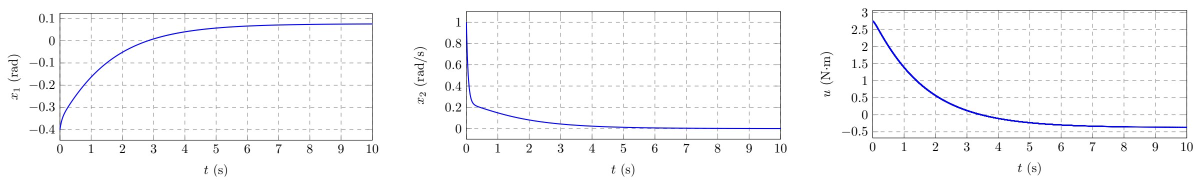


Figure 5.3: States and inputs of the inverted pendulum with initial condition $[-0.4, 1.0]^T$

5.7 Numerical Results

We illustrate the results in this chapter on an inverted pendulum described by the following model:

$$f(x_1, x_2, u) = \begin{bmatrix} x_2 & \frac{g}{l}\sin(x_1) - \frac{h}{ml^2}x_2 + \frac{1}{ml}\cos(x_1)u \end{bmatrix}^T$$

where x_1 is the angular position, x_2 is the angular velocity, and control input u is the torque applied on the point mass. The parameters are the rod mass, m; the rod length, l; the (dimensionless) coefficient of rotational friction, h; and the acceleration due to gravity, g. For the purposes of our experiments, we considered a subset of the state/control space specified by: $x_1 \in [-1, 1]$, $x_2 \in [-1, 1]$ and $u \in [-6, 6]$. Furthermore, we considered model parameters: $m = 0.5$ kg; $l = 0.5$ m; $h = 2$; and $g = 9.8$ N/kg. Then for different choices of the design parameters μ, we obtain the following sizes N for the corresponding TLL-NN architecture along with the corresponding τ, η and the δ that is required for the specification satisfaction:

In the sequel, we will show the control performance of a TLL-NN architecture with 400 local linear region. While there are a number of techniques that can be used to train the resulting

μ	δ	τ	η	N
0.35	0.8694	0.0098	0.583	235
0.3	0.5287	0.0083	0.5	320
0.25	0.3039	0.0069	0.417	460
0.2	0.1610	0.0056	0.334	720
0.15	0.0749	0.0042	0.25	1280
0.1	0.0275	0.0028	0.167	2880

Table 5.1: Dependence of NN parameter on partition parameters

NN, for the sake of simplicity, we utilize Imitation learning where the NN is trained in a supervised fashion from data collected from an expert controller. In particular, we designed an expert controller that stabilizes the inverted pendulum; we chose to use Pessoa [93] to design our expert using the parameter values specified above. In particular, we tasked Pessoa to design a zero-order-hold controller that stabilizes the inverted pendulum in a subset $X_{\text{spec}} = [-1, 1] \times [-0.5, 0.5]$: that is the controller should transfer the state of the system to this specified set and keep it there for all time thereafter. From this expert controller, we collected 8400 data points of state-action pairs; this data was obtained by uniformly sampling the state space. We then used Keras [31] to train the TLL NN using this data. Finally, we simulated the motion of the inverted pendulum using this TLL NN controller. Shown in 5.2 and 5.3 are the state and control trajectories for this controller starting from initial state $[0.7, 0.5]$ and $[-0.4, 1]$, respectively. In both, the TLL NN controller met the same specification that was used to design the expert.

Bibliography

[1] A. Abate, D. Ahmed, A. Edwards, M. Giacobbe, and A. Peruffo. Fossil: A software tool for the formal synthesis of lyapunov functions and barrier certificates using neural networks. In *Proceedings of the 24th ACM International Conference on Hybrid Systems: Computation and Control*, 2021.

[2] J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning*, pages 22–31, 2017.

[3] M. E. Akintunde, A. Lomuscio, L. Maganti, and E. Pirovano. Reachability analysis for neural agent-environment systems. In *Proceedings of the Sixteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2018)*. AAAI, 2018.

[4] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and UfukTopcu. Safe reinforcement learning via shielding. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 2669–2678, 2018.

[5] G. Anderson, A. Verma, I. Dillig, and S. Chaudhuri. Neurosymbolic reinforcement learning with formally verified exploration. In *Advances in Neural Information Processing Systems*, pages 6172–6183, 2020.

[6] S. Bak, H.-D. Tran, K. Hobbs, and T. T. Johnson. Improved geometric path enumeration for verifying relu neural networks. In *Proceedings of the 32nd International Conference on Computer Aided Verification*, 2020.

[7] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. *arXiv:1611.02167*, 2016.

[8] A. Balakrishnan and J. V. Deshmukh. Structured reward shaping using signal temporal logic specifications. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3481–3486, 2019.

[9] O. Bastani, S. Li, and A. Xu. Safe reinforcement learning via statistical model predictive shielding. In *Robotics: Science and Systems*, 2021.

[10] O. Bastani, Y. Pu, and A. Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Advances in Neural Information Processing Systems*, pages 2499–2509, 2018.

[11] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas. Symbolic planning and control of robot motion [grand challenges of robotics]. *IEEE Robotics & Automation Magazine*, 14(1):61–70, 2007.

[12] C. Belta, B. Yordanov, and E. A. Gol. *Formal methods for discrete-time dynamical systems*, volume 15. Springer, 2017.

[13] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars. *Computational geometry: algorithms and applications*. Springer-Verlag TELOS, 2008.

[14] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[15] F. Berkenkamp, A. Krause, and A. P. Schoellig. Bayesian optimization with safety constraints: safe and automatic parameter tuning in robotics. In *Machine Learning*. Springer, 2021.

[16] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause. Safe model-based reinforcement learning with stability guarantees. In *Advances in neural information processing systems*, 2017.

[17] D. Bertsekas. Infinite time reachability of state-space regions by using feedback control. *IEEE Transactions on Automatic Control*, 17(5):604–613, 1972.

[18] A. Bhatia, L. E. Kavraki, and M. Y. Vardi. Sampling-based motion planning with temporal goals. In *2010 IEEE International Conference on Robotics and Automation*, pages 2689–2696. IEEE, 2010.

[19] A. Bhatia, M. R. Maly, L. E. Kavraki, and M. Y. Vardi. Motion planning with complex goals. *IEEE Robotics & Automation Magazine*, 18(3):55–64, 2011.

[20] A. Biere, K. Heljanko, T. Junttila, T. latvala, and V. Schuppan. Linear encoding of bounded LTL model checking. *Logical Methods in Computer Science*, 2(5:5):1–64, 2006.

[21] A. Biere, K. Heljanko, T. Junttila, T. Latvala, and V. Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, 2(5:5):1–64, 2006.

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

[23] R. Bunel, I. Turkaslan, P. H. S. Torr, P. Kohli, and M. P. Kumar. A unified view of piecewise linear neural network verification. *arXiv preprint*, 2018.

[24] Z. Cao, M. Kwon, and D. Sadigh. Transfer reinforcement learning across homotopy classes. In *IEEE Robotics and Automation Letters*, 2021.

[25] S. Carr, N. Jansen, and U. Topcu. Verifiable rnn-based policies for pomdps under temporal logic constraints. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence*, page 4121–4127, 2020.

[26] N. Cauchi and A. Abate. StocHy-automated verification and synthesis of stochastic processes. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 258–259, 2019.

[27] M. Charikar, J. Steinhardt, and G. Valiant. Learning from untrusted data. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 47–60. ACM, 2017.

[28] S. Chen, M. Fazlyab, M. Morari, G. J. Pappas, and V. M. Preciado. Learning lyapunov functions for hybrid systems. In *Proceedings of the 24th ACM International Conference on Hybrid Systems: Computation and Control*, 2021.

[29] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification (CAV)*, pages 258–263. Springer, 2013.

[30] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick. End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3387–3395, 2019.

[31] F. Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[32] R. Choudhury, G. Swamy, D. Hadfield-Menell, and A. D. Dragan. On the utility of model learning in hri. In *Proceedings of the 14th ACM/IEEE International Conference on Human-Robot Interaction*, HRI '19, page 317–325. IEEE Press, 2019.

[33] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. In *Advances in neural information processing systems*, pages 8092–8101, 2018.

[34] Y. Chow, O. Nachum, A. Faust, E. Duenez-Guzman, and M. Ghavamzadeh. Lyapunov-based safe policy optimization for continuous control. In *RL4RealLife Workshop in the 36th International Conference on Machine Learning*, 2019.

[35] L. De Moura and N. Björner. Z3: An efficient SMT solver. In *Proc. Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, 2008.

[36] X. C. Ding, M. Kloetzer, Y. Chen, and C. Belta. Automatic deployment of robotic teams. *IEEE Robotics & Automation Magazine*, 18(3):75–86, 2011.

[37] T. Dreossi, A. Donzé, and S. A. Seshia. Compositional falsification of cyber-physical systems with machine learning components. In *NASA Formal Methods Symposium*, pages 357–372. Springer, 2017.

[38] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*. Springer, 2018.

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

[40] S. Esmaeil Zadeh Soudjani and A. Abate. Adaptive and sequential gridding procedures for the abstraction and verification of stochastic processes. *SIAM Journal on Applied Dynamical Systems*, 12(2):921–956, 2013.

[41] T. Everitt, G. Lea, and M. Hutter. AGI safety literature review. *arXiv preprint*, 2018.

[42] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas. Temporal logic motion planning for dynamic robots. *Automatica*, 45(2):343–352, 2009.

[43] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas. Hybrid controllers for path planning: A temporal logic approach. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 4885–4890. IEEE, 2005.

[44] G. E. Fainekos, S. G. Loizou, and G. J. Pappas. Translating temporal logic to controller specifications. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 899–904. IEEE, 2006.

[45] M. Fazlyab, A. Robey, H. Hassani, M. Morari, and G. Pappas. Efficient and accurate estimation of lipschitz constants for deep neural networks. In *Advances in Neural Information Processing Systems*, pages 11423–11434, 2019.

[46] A. Ferdowsi, U. Challita, W. Saad, and N. B. Mandayam. Robust deep reinforcement learning for security and safety in autonomous vehicle systems. *arXiv preprint*, 2018.

[47] J. Ferlez, H. Khedr, and Y. Shoukry. Fast batllnn: fast box analysis of two-level lattice neural networks. In *25th ACM International Conference on Hybrid Systems: Computation and Control*, pages 1–11, 2022.

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

[49] J. Ferlez and Y. Shoukry. Bounding the complexity of formally verifying neural networks: A geometric approach. In *60th IEEE Conference on Decision and Control*, pages 5104–5109, 2021.

[50] J. Ferlez, X. Sun, and Y. Shoukry. Two-level lattice neural network architectures for control of nonlinear systems. In *59th IEEE Conference on Decision and Control*, 2020.

[51] C. Finn, S. Levine, and P. Abbeel. Guided cost learning: deep inverse optimal control via policy optimization. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

[52] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. Gillula, and C. J. Tomlin. A general safety framework for learning-based control in uncertain robotic systems. *IEEE Transactions on Automatic Control*, 64(7):2737–2752, 2018.

[53] G. Frehse, C. L. 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 *International Conference on Computer Aided Verification (CAV)*, pages 379–395. Springer, 2011.

[54] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling. Ffrob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research*, 37(1):104–136, 2018.

[55] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. Ai 2: Safety and robustness certification of neural networks with abstract interpretation. In *Security and Privacy (SP), 2018 IEEE Symposium on*, 2018.

[56] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of the 15th IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification XV*, pages 3–18, 1996.

[57] S. Gil, S. Kumar, M. Mazumder, D. Katabi, and D. Rus. Guaranteeing spoof-resilient multi-robot networks. *Autonomous Robots*, 41(6):1383–1400, 2017.

[58] M. Guo, K. H. Johansson, and D. V. Dimarogonas. Motion and action planning under ltl specifications using navigation functions and action description language. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 240–245. IEEE, 2013.

[59] L. Gupta, R. Jain, and G. Vaszkun. Survey of important issues in uav communication networks. *IEEE Communications Surveys & Tutorials*, 18(2):1123–1152, 2016.

[60] M. Hasanbeig, Y. Kantaros, A. Abate, D. Kroening, G. J. Pappas, , and I. Lee. Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees. In *58th IEEE Conference on Decision and Control*, pages 5338–5343, 2019.

[61] F. Higgins, A. Tomlinson, and K. M. Martin. Threats to the swarm: Security considerations for swarm robotics. *International Journal on Advances in Security*, 2(2&3), 2009.

[62] K. Hsu, R. Majumdar, K. Mallik, and A.-K. Schmuck. Multi-layered abstraction-based controller synthesis for continuous-time systems. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control*, page 120–129, 2018.

[63] K.-C. Hsu, V. Rubies-Royo, C. J. Tomlin, and J. F. Fisac. Safety and liveness guarantees through reach-avoid reinforcement learning. In *Robotics: Science and Systems*, 2021.

[64] IBM. Ibm ilog cplex optimizer. `www.ibm.com/software/integration/optimization/cplex-optimizer/`, 2012.

[65] 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*, pages 169–178, 2019.

[66] P. Jagtap and M. Zamani. QUEST: a tool for state-space quantization-free synthesis of symbolic controllers. In *International conference on quantitative evaluation of systems*, pages 309–313. Springer, 2017.

[67] Y. Jiang, S. Bharadwaj, B. Wu, R. Shah, U. Topcu, and P. Stone. Temporal-logic-based reward shaping for continuing learning tasks. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, pages 7995–8003, 2020.

[68] 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*, pages 97–117. Springer, 2017.

[69] H. K. Khalil. *Nonlinear Systems*. Pearson, Third edition, 2001.

[70] H. Khedr, J. Ferlez, and Y. Shoukry. Peregrinn: Penalized-relaxation greedy neural network verifier. In *International Conference on Computer Aided Verification*, pages 287–300. Springer, 2021.

[71] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *Proceedings of the 31st International Conference on Machine Learning*, 2014.

[72] G. Klančar, A. Zdešar, S. Blažič, and I. Škrjanc. Wheeled mobile robotics. *Elsevier*, 2017.

[73] M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Trans. Automatic Control*, 53(1):287–297, 2008.

[74] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Where's waldo? sensor-based temporal logic motion planning. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3116–3121. IEEE, 2007.

[75] H. Kress-Gazit, M. Lahijanian, and V. Raman. Synthesis for robots: Guarantees and feedback for robot behavior. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:211–236, 2018.

[76] H. Kress-Gazit, T. Wongpiromsarn, and U. Topcu. Correct, reactive, high-level robot control. *IEEE Robotics & Automation Magazine*, 18(3):65–74, 2011.

[77] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[78] O. Kupferman and M. Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, pages 19:291–314, 2001.

[79] Z. Kurd and T. Kelly. Establishing safety criteria for artificial neural networks. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pages 163–169. Springer, 2003.

[80] V. Kurtz, P. M. Wensing, and H. Lin. Robust Approximate Simulation for Hierarchical Control of Linear Systems under Disturbances. *American Control Conference*, 2020.

[81] T. Latvala. Efficient model checking of safety properties. In *Model Checking Software. 10th International SPIN Workshop*, pages 74–88. Springer, 2003.

[82] A. Lavaei, M. Khaled, S. Soudjani, and M. Zamani. AMYTISS: parallelized automated controller synthesis for large-scale stochastic systems. In *International Conference on Computer Aided Verification*, pages 461–474. Springer, 2020.

[83] A. Lavaei, S. Soudjani, A. Abate, and M. Zamani. Automated verification and synthesis of stochastic hybrid systems: A survey. In *Automatica*, 2021.

[84] J. Leike, M. Martic, V. Krakovna, P. A. Ortega, T. Everitt, A. Lefrancq, L. Orseau, and S. Legg. AI safety gridworlds. *arXiv preprint*, 2017.

[85] F. Leofante, N. Narodytska, L. Pulina, and A. Tacchella. Automated verification of neural networks: Advances, challenges and perspectives. *arXiv preprint*, 2018.

[86] A. Liu, G. Shi, S.-J. Chung, A. Anandkumar, and Y. Yue. Robust regression for safe exploration in control. In *Proceedings of Machine Learning Research*, 2020.

[87] C. Liu, T. Arnon, C. Lazarus, C. Barrett, and M. J. Kochenderfer. Algorithms for verifying deep neural networks. *arXiv preprint arXiv:1903.06758*, 2019.

[88] X. Lu, D. Xu, L. Xiao, L. Wang, and W. Zhuang. Anti-jamming communication game for UAV-aided VANETs. In *IEEE Global Communications Conf.*, pages 1–6, Dec 2017.

[89] L. Ma, F. Juefei-Xu, J. Sun, C. Chen, T. Su, F. Zhang, M. Xue, B. Li, L. Li, Y. Liu, J. Zhao, and Y. Wang. Deepgauge: Comprehensive and multi-granularity testing criteria for gauging the robustness of deep learning systems. *arXiv preprint*, 2018.

[90] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao, and Y. Wang. Deepmutation: Mutation testing of deep learning systems. *arXiv preprint*, 2018.

[91] K. Mallik, A.-K. Schmuck, S. Soudjani, and R. Majumdar. Compositional Synthesis of Finite-State Abstractions. *IEEE Transactions on Automatic Control*, 64(6):2629–2636, 2019.

[92] M. Mazo, A. Davitian, and P. Tabuada. Pessoa: a tool for embedded controller synthesis. In *International conference on computer aided verification*, pages 566–569. Springer, 2010.

[93] M. Mazo, A. Davitian, and P. Tabuada. PESSOA: A tool for embedded controller synthesis. In *Proceedings of the 22nd International Conference on Computer Aided Verification*, CAV'10, pages 566–569. Springer-Verlag, 2010.

[94] M. J. Mears. Cooperative electronic attack using unmanned air vehicles. In *Proc. American Control Conference*, pages 3339–3347. IEEE, 2005.

[95] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio. On the number of linear regions of deep neural networks. In *Advances in Neural Information Processing Systems*, 2014.

[96] S. Mouelhi, A. Girard, and G. Gössler. CoSyMA: a tool for controller synthesis using multi-scale abstractions. In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pages 83–88, 2013.

[97] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 27–38. ACM, 2017.

[98] P. Nuzzo, H. Xu, N. Ozay, J. Finn, A. Sangiovanni-Vincentelli, R. Murray, A. Donze, and S. Seshia. A contract-based methodology for aircraft electric power system design. *IEEE Access*, 2:1–25, 2014.

[99] M. Palan, G. Shevchuk, N. Charles Landolfi, and D. Sadigh. Learning reward functions by integrating human demonstrations and preferences. In *Robotics: Science and Systems*, 2019.

[100] A. Paudice, L. Muñoz-González, and E. C. Lupu. Label sanitization against label flipping poisoning attacks. *arXiv preprint*, 2018.

[101] S. Paul, V. Kurin, and S. Whiteson. Fast efficient hyperparameter tuning for policy gradients. *arXiv:1902.06583*, 2019.

[102] P. Pauli, A. Koch, J. Berberich, and F. Allgöwer. Training robust neural networks using lipschitz bounds. *IEEE Control Systems Letters*, 2020.

[103] F. Pedregosa. Hyperparameter optimization with approximate gradient. *arXiv:1602.02355*, 2016.

[104] K. Pei, Y. Cao, J. Yang, and S. Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 1–18. ACM, 2017.

[105] E. Plaku and S. Karaman. Motion planning with temporal-logic specifications: Progress and challenges. *AI Communications*, pages 1–12, 2015.

[106] A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.

[107] G. Pola, A. Girard, and P. Tabuada. Approximately bisimilar symbolic models for nonlinear control systems. *Automatica*, 44(10):2508–2516, 2008.

[108] L. Pulina and A. Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *International Conference on Computer Aided Verification*, pages 243–257. Springer, 2010.

[109] Q. Qingwen, D. Wenfeng, L. Meiqing, and Y. Yang. Cooperative jamming resource allocation of UAV swarm based on multi-objective DPSO. In *Proc. Chinese Control and Decision Conference*, pages 5319–5325, June 2018.

[110] Y. Quanming, W. Mengshuo, J. E. Hugo, G. Isabelle, H. Yi-Qi, L. Yu-Feng, T. Wei-Wei, Y. Qiang, and Y. Yang. Taking human out of learning applications: A survey on automated machine learning. *arXiv:1810.13306*, 2018.

[111] K. Rakelly, A. Zhou, D. Quillen, C. Finn, and S. Levine. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[112] S. V. Rakovic and M. Baric. Parameterized robust control invariant sets for linear systems: Theoretical advances and computational remarks. *IEEE Transactions on Automatic Control*, 55(7):1599–1614, 2010.

[113] C. E. Rasmussen and C. Williams. Gaussian processes for machine learning. *the MIT Press*, 2006.

[114] V. Renganathan and T. Summers. Spoof resilient coordination for distributed multi-robot systems. In *Int. Symp. Multi-Robot and Multi-Agent Systems*, pages 135–141. IEEE, 2017.

[115] A. Robey, H. Hu, L. Lindemann, H. Zhang, D. V. Dimarogonas, S. Tu, and N. Matni. Learning control barrier functions from expert demonstrations. In *59th IEEE Conference on Decision and Control*, 2020.

[116] F. Rossi and N. Mattei. Building ethically bounded AI. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, pages 9785–9789, 2019.

[117] W. Ruan, X. Huang, and M. Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. *arXiv preprint*, 2018.

[118] W. Ruan, M. Wu, Y. Sun, X. Huang, D. Kroening, and M. Kwiatkowska. Global robustness evaluation of deep neural networks with provable guarantees for l0 norm. *arXiv preprint*, 2018.

[119] M. Rungger and P. Tabuada. Computing robust controlled invariant sets of linear systems. *IEEE Transactions on Automatic Control*, 62(7):3665–3670, 2017.

[120] M. Rungger and M. Zamani. SCOTS: a tool for the synthesis of symbolic controllers. In *Proceedings of the 19th international conference on hybrid systems: Computation and control*, pages 99–104, 2016.

[121] S. Sadraddini and C. Belta. Formal guarantees in data-driven model identification and control synthesis. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*, pages 147–156. ACM, 2018.

[122] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia. Automated composition of motion primitives for multi-robot systems from safe LTL specifications. In *Int. Conf. Intelligent Robots and Systems*, pages 1525–1532, 2014.

[123] U. Santa Cruz and Y. Shoukry. Nnlander-verif: A neural network formal verification framework for vision-based autonomous aircraft landing. In *NASA Formal Methods Symposium*, pages 213–230. Springer, 2022.

[124] SatEX. Satex solver. `https://yshoukry.bitbucket.io/SatEX/`, 2018.

[125] W. Saunders, G. Sastry, A. Stuhlmueller, and O. Evans. Trial without error: Towards safe reinforcement learning via human intervention. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2067–2069, 2018.

[126] K. Scheibler, L. Winterer, R. Wimmer, and B. Becker. Towards verification of artificial neural networks. In *Workshop on Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV)*, pages 30–40, 2015.

[127] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

[128] S. A. Seshia, A. Desai, T. Dreossi, D. Fremont, S. Ghosh, E. Kim, S. Shivakumar, M. Vazquez-Chanlatte, and X. Yue. Formal specification for deep neural networks. *arXiv preprint*, 2018.

[129] S. A. Seshia, D. Sadigh, and S. S. Sastry. Towards verified artificial intelligence. *arXiv preprint*, 2016.

[130] Y. Shoukry, P. Nuzzo, A. Balkan, I. Saha, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada. Linear temporal logic motion planning for teams of underactuated robots using satisfiability modulo convex programming. In *2017 IEEE 56th annual conference on decision and control (CDC)*, pages 1132–1137. IEEE, 2017.

[131] Y. Shoukry, P. Nuzzo, A. Balkan, I. Saha, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada. Linear temporal logic motion planning for teams of underactuated robots using satisfiability modulo convex programming. In *Proc. IEEE Conf. Decision and Control*, pages 1132–1137, 2017.

[132] Y. Shoukry, P. Nuzzo, I. Saha, A. Sangiovanni-Vincentelli, S. Seshia, G. Pappas, and P. Tabuada. Scalable lazy SMT-based motion planning. In *Proc. IEEE Conf. Decision and Control*, pages 6683–6688, 2016.

[133] Y. Shoukry, P. Nuzzo, A. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada. SMC: Satisfiability modulo convex optimization. In *Proc. Int. Conf. Hybrid Systems: Computation and Control*, Apr. 2017.

[134] Y. Shoukry, P. Nuzzo, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada. SMC: Satisfiability Modulo Convex optimization. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control (HSCC)*, pages 19–28. ACM, 2017.

[135] Y. Shoukry, P. Nuzzo, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada. Smc: Satisfiability modulo convex programming [40pt]. *Proceedings of the IEEE*, 106(9):1655–1679, 2018.

[136] S. E. Z. Soudjani, C. Gevaerts, and A. Abate. FAUST ^{2} : Formal abstractions of uncountable-state stochastic processes. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 272–286. Springer, 2015.

[137] S. Srisakaokul, Z. Wu, A. Astorga, O. Alebiosu, and T. Xie. Multiple-implementation testing of supervised learning software. In *Proceedings of the AAAI-18 Workshop on Engineering Dependable and Secure Machine Learning Systems (EDSMLS)*, 2018.

[138] J. Steinhardt, P. W. W. Koh, and P. S. Liang. Certified defenses for data poisoning attacks. In *Advances in Neural Information Processing Systems*, pages 3520–3532, 2017.

[139] X. Sun, W. Fatnassi, U. Santa Cruz, and Y. Shoukry. Provably safe model-based meta reinforcement learning: An abstraction-based approach. In *60th IEEE Conference on Decision and Control*, pages 2963–2968, 2021.

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

[141] X. Sun, R. Nambiar, M. Melhorn, Y. Shoukry, and P. Nuzzo. DoS-resilient multi-robot temporal logic motion planning. In *IEEE International Conference on Robotics and Automation*, 2019.

[142] X. Sun and Y. Shoukry. Provably correct training of neural network controllers using reachability analysis. *arXiv preprint arXiv:2102.10806*, 2021.

[143] X. Sun and Y. Shoukry. Neurosymbolic motion and task planning for linear temporal logic tasks. *arXiv preprint arXiv:2210.05180*, 2022.

[144] X. Sun and Y. Shoukry. NNsynth: Neural network guided abstraction-based controller synthesis for stochastic systems. In *61st IEEE Conference on Decision and Control*, 2022.

[145] Y. Sun, X. Huang, and D. Kroening. Testing deep neural networks. *arXiv preprint*, 2018.

[146] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening. Concolic testing for deep neural networks. *arXiv preprint*, 2018.

[147] P. Tabuada. *Verification and control of hybrid systems: a symbolic approach*. Springer Science & Business Media, 2009.

[148] P. Tabuada. *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer US, 2009.

[149] P. Tabuada and G. J. Pappas. Linear time logic control of discrete-time linear systems. *IEEE Trans. Automatic Control*, 51(12):1862–1877, 2006.

[150] A. J. Taylor, A. Singletary, Y. Yue, and A. D. Ames. A control barrier perspective on episodic learning via projection-to-state safety. In *IEEE Control Systems Letters*, pages 1019–1024, 2021.

[151] Y. Tian, K. Pei, S. Jana, and B. Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. *arXiv preprint arXiv:1708.08559*, 2017.

[152] V. Tjeng and R. Tedrake. Verifying neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.

[153] C. E. Tuncali, G. Fainekos, H. Ito, and J. Kapinski. Simulation-based adversarial test generation for autonomous vehicles with machine learning components. *arXiv preprint arXiv:1804.06760*, 2018.

[154] C. E. Tuncali, J. Kapinski, H. Ito, and J. V. Deshmukh. Reasoning about safety of learning-enabled components in autonomous cyber-physical systems. In *Proceedings of the 55th Annual Design Automation Conference (DAC)*. ACM, 2018.

[155] M. Turchetta, F. Berkenkamp, and A. Krause. Safe exploration in finite markov decision processes with gaussian processes. In *Advances in Neural Information Processing Systems*, pages 4312–4320, 2016.

[156] A. Verma, H. Le, Y. Yue, and S. Chaudhuri. Imitation-projected programmatic reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 15752–15763, 2019.

[157] K. P. Wabersich and M. N. Zeilinger. Linear model predictive safety certification for learning-based control. In *57th IEEE Conference on Decision and Control*, 2018.

[158] K. P. Wabersich and M. N. Zeilinger. A predictive safety filter for learning-based control of constrained nonlinear dynamical systems. In *Automatica*, 2021.

[159] J. Wang, J. Sun, P. Zhang, and X. Wang. Detecting adversarial samples for deep neural networks through mutation testing. *arXiv preprint*, 2018.

[160] L. Wang, E. A. Theodorou, and M. Egerstedt. Safe learning of quadrotor dynamics using barrier certificates. In *IEEE International Conference on Robotics and Automation*, pages 2460–2465, 2018.

[161] G. Weiss, Y. Goldberg, and E. Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In *Proceedings of the 35th International Conference on Machine Learning*, pages 5247–5256, 2018.

[162] L. Wen, J. Duan, S. E. Li, S. Xu, and H. Peng. Safe reinforcement learning for autonomous vehicles through parallel constrained policy optimization. In *IEEE 23rd International Conference on Intelligent Transportation Systems*, 2020.

[163] M. Wicker, X. Huang, and M. Kwiatkowska. Feature-guided black-box safety testing of deep neural networks. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 408–426. Springer, 2018.

[164] Wikipedia. List of autonomous car fatalities. `https://en.wikipedia.org/wiki/List_of_autonomous_car_fatalities`.

[165] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding horizon temporal logic planning. *IEEE Trans. Automatic Control*, 57(11):2817–2830, 2012.

[166] W. Xiang and T. T. Johnson. Reachability analysis and safety verification for neural network control systems. *arXiv preprint arXiv:1805.09944*, 2018.

[167] 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*, pages 123–144. Springer, 2019.

[168] W. Xiang, P. Musau, A. A. Wild, D. M. Lopez, N. Hamilton, X. Yang, J. Rosenfeld, and T. T. Johnson. Verification for machine learning, autonomy, and neural networks survey. *arXiv preprint arXiv:1810.01989*, 2018.

[169] W. Xiang, H.-D. Tran, and T. T. Johnson. Reachable set computation and safety verification for neural networks with relu activations. *arXiv preprint arXiv:1712.08163*, 2017.

[170] L. Xiao, X. Lu, D. Xu, Y. Tang, L. Wang, and W. Zhuang. Uav relay in vanets against smart jamming with reinforcement learning. *IEEE Trans. on Vehicular Technology*, 67(5):4087–4097, May 2018.

[171] W. Xiao, C. Belta, and C. G. Cassandras. Adaptive control barrier functions. In *IEEE Transactions on Automatic Control*, 2022.

[172] Y. Xu, G. Ren, J. Chen, Y. Luo, L. Jia, X. Liu, Y. Yang, and Y. Xu. A One-Leader Multi-Follower Bayesian-Stackelberg Game for Anti-Jamming Transmission in UAV Communication Networks. *IEEE Access*, 6:21697–21709, 2018.

[173] C. Yun, S. Sra, and A. Jadbabaie. Small relu networks are powerful memorizers: a tight analysis of memorization capacity. *Advances in neural information processing systems*, 2019.

[174] M. Zamani, G. Pola, M. Mazo, and P. Tabuada. Symbolic Models for Nonlinear Control Systems Without Stability Assumptions. *IEEE Transactions on Automatic Control*, 57(7), 2012.

[175] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid. Deeproad: Gan-based metamorphic autonomous driving system testing. *arXiv preprint*, 2018.

[176] Z. Zhang, G. Ernst, S. Sedwards, P. Arcaini, and I. Hasuo. Two-layered falsification of hybrid systems guided by monte carlo tree search. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[177] Y. Zou, J. Zhu, X. Wang, and L. Hanzo. A survey on wireless security: Technical challenges, recent advances, and future trends. *Proceedings of the IEEE*, pages 1–39, 2016.