Formal Verification of Neural Network Controlled Autonomous Systems

Xiaowu Sun
Department of Electrical and Computer Engineering
University of Maryland, College Park
xsun24@terpmail.umd.edu

Haitham Khedr
Department of Electrical and Computer Engineering
University of Maryland, College Park
hkhedr@umd.edu

Yasser Shoukry
Department of Electrical and Computer Engineering
University of Maryland, College Park
yshoukry@ece.umd.edu

ABSTRACT

In this paper, 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.

CCS CONCEPTS

• **Computing methodologies** → **Artificial intelligence**; *Control methods*; • **Security and privacy** → *Formal methods and theory of security*;

KEYWORDS

Formal Verification, Machine Learning, Satisfiability Solvers

ACM Reference Format:
Xiaowu Sun, Haitham Khedr, and Yasser Shoukry. 2019. Formal Verification of Neural Network Controlled Autonomous Systems. In *22nd ACM International Conference on Hybrid Systems: Computation and Control (HSCC '19), April 16–18, 2019, Montreal, QC, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3302504.3311802

HSCC '19, April 16–18, 2019, Montreal, QC, Canada

ACM ISBN 978-1-4503-6282-5/19/04…$15.00
https://doi.org/10.1145/3302504.3311802

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 [1].

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 [2–8]. 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 [9–18] 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 [19–21]. 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 [19–21].

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 [22–27]. 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 [28–33]. 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 [32, 34].

Using off-the-shelf MILP solvers does not lead to scalable approaches to handle neural networks with hundreds and thousands of neurons [29]. To circumvent this problem, several MILP-like solvers targeted toward the neural network verification problem are proposed. For example, the work reported in [28] proposed a modified Simplex algorithm (originally used to solve linear programs) to take into account ReLU nonlinearities as well. Similarly, the work reported in [29] 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 [35, 36]. A thorough survey on different algorithms for verification of neural networks against input-output range specifications can be found in [37] 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 [38–40]. Reachability analysis is performed by either separately estimating the output set of the neural network and the reachable set of continuous dynamics [38], or by translating the neural network controlled system into a hybrid system [39]. Once the neural network controlled system is translated into a hybrid system, off-the-shelf existing verification tools of hybrid systems, such as SpaceEx [41] for piecewise affine dynamics and Flow* [42] for nonlinear dynamics, can be used to verify safety properties of the system. Another related technique is the safety verification using barrier certificates [43]. 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 paper, the case in which the robotic system is equipped with a LiDAR scanner that is used to sense the environment. 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 paper 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.

For brevity, we omit here the proofs of the main technical results and report them in an extended version of the paper [44].

2 PROBLEM FORMULATION

2.1 Notation

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 $\text{atan2}(z) = \text{atan2}(y, x)$.

2.2 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 $\{O_1, \ldots, O_o\}$, with $O_i \subset \mathcal{W}$ 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., $\mathcal{O} = \{\partial W, O_1, \ldots, O_o\}$. The dynamics of the robot is described by a discrete-time linear system of the form:

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

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

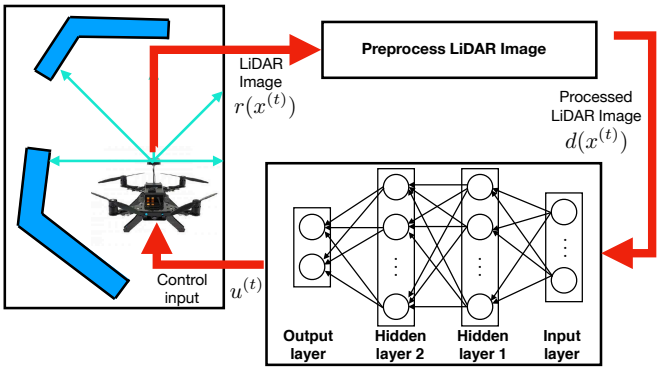

Figure 1: Pictorial representation of the problem setup under consideration.

linearizable, the state space model (1) corresponds to its feedback linearized dynamics. We denote by $\zeta(x) \in \mathbb{R}^2$ the natural projection of x onto the workspace $\mathcal{W}$, i.e., $\zeta(x^{(t)})$ is the position of the robot at time t.

2.3 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 $\theta^{(t)}_{\text{lidar}} \in \mathbb{R}$ the heading angle of the LiDAR at time t. Similarly, we denote by $\theta^{(t)}_i = \theta^{(t)}_{\text{lidar}} + (i-1)\frac{2\pi}{N}$, with $i \in \{1, \ldots, N\}$, the angle of the ith laser beam at time t where $\theta^{(t)}_1 = \theta^{(t)}_{\text{lidar}}$ and by $\theta^{(t)} = (\theta^{(t)}_1, \ldots, \theta^{(t)}_N)$ the vector of the angles of all the laser beams. While the heading angle of the LiDAR, $\theta^{(t)}_{\text{lidar}}$, changes as the robot pose changes over time, i.e., $\theta^{(t)}_{\text{lidar}} = f(x^{(t)})$ for some nonlinear function f, in this paper we focus on the case when the heading angle of the LiDAR, $\theta^{(t)}_{\text{lidar}}$, 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.:

$$r_i(x^{(t)}) = \min_{O_i \in O} \min_{z \in O_i} \|z - \zeta(x^{(t)})\|_2$$
$$\text{s.t.} \quad \text{atan2}\left(z - \zeta(x^{(t)})\right) = \theta_i. \tag{2}$$

In this paper, 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:

$$d_i(x^{(t)}) = \left(r_i(x^{(t)})\cos\theta_i,\ r_i(x^{(t)})\sin\theta_i\right),$$
$$d(x^{(t)}) = \left(d_1(x^{(t)}), \ldots d_N(x^{(t)})\right). \tag{3}$$

2.4 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 [45]. For such neural network architecture, the neural network controller $u^{(t)} = f_{\text{NN}}(d(x^{(t)}))$ can be written as:

$$h^{1(t)} = \max\left(0,\ W^0 d(x^{(t)}) + w^0\right),$$
$$h^{2(t)} = \max\left(0,\ W^1 h^{1(t)} + w^1\right),$$
$$\vdots$$
$$h^{L(t)} = max\left(0,\ W^{L-1} h^{L-1(t)} + w^{L-1}\right),$$
$$u^{(t)} = W^L h^{L(t)} + w^L, \tag{4}$$

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.

2.5 Robot Trajectories and Safety Specifications

The trajectories of the robot whose dynamics are described by (1) when controlled by the neural network controller (2)-(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 2.1 (Safe Trajectory). A robot trajectory η_{x_0} is called safe if $\zeta(\eta_{x_0}(t)) \in \mathcal{W},\ \zeta(\eta_{x_0}(t)) \notin O_i,\ \forall O_i \in O,\ \forall t \in \mathbb{N}$.

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

Problem 2.2. *Consider the autonomous robot whose dynamics are governed by* (1) *which is controlled by the neural network controller described by* (4) *which processes LiDAR images described by* (2)-(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.*

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

Definition 3.2. 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 $(x_a, x_b) \in Q$,

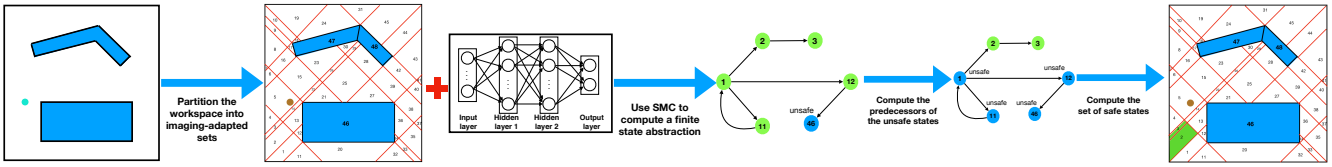

Figure 2: Pictorial representation of the proposed framework.

(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 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 nondeterministic) $\mathcal{S}_{\mathcal{F}} = (\mathcal{F}, \delta_{\mathcal{F}})$ 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 (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 (3) which maps the robot position to the processed observations) is an affine map as shown in Section 4. Therefore, as summarized in Algorithm 1, the first step is to compute such partitioning $\mathcal{W}^\star$ of the workspace (WKSP-PARTITION, line 2 in Algorithm 1). 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 1) 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.

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 1). 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

Algorithm 1 VERIFY-NN($\mathcal{X}, \delta_{\text{NN}}$)

1: **Step 1: Partition the workspace**
2: $(\mathcal{W}^\star, \mathcal{W}')$ = WKSP-PARTITION($\mathcal{W}$, O, θ_p, θ_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: $\quad\delta_{\mathcal{F}}$.ADD-TRANSITION($s$, s')
8: **Step 2.2: Pre-process the neural network**
9: **for** each s in $\mathcal{F}$ **do**
10: $\quad\mathcal{X}_s = \{x \in \mathcal{X} \mid (x, s) \in Q\}$
11: $\quad CE_s$ = PRE-PROCESS($\mathcal{X}_s$, δ_{NN})
12: **Step 2.3: Compute the transition map $\delta_{\mathcal{F}}$**
13: **for** each s in $\mathcal{F}$ and s' in $\mathcal{F}'$ where $s \notin s'$ **do**
14: $\quad\mathcal{X}_s = \{x \in \mathcal{X} \mid (x, s) \in Q\}$
15: $\quad\mathcal{X}_{s'} = \{x \in \mathcal{X} \mid (x, s^\star) \in Q,\ \forall s^\star \in s'\}$
16: $\quad$STATUS = CHECK-FEASIBILITY($\mathcal{X}_s$, $\mathcal{X}_{s'}$, δ_{NN}, CE_s)
17: $\quad$**if** STATUS == `INFEASIBLE` **then**
18: $\quad\quad$**for** each $s^\star$ in s' **do**
19: $\quad\quad\quad\delta_{\mathcal{F}}$.REMOVE-TRANSITION($s$, $s^\star$)
20: $\quad$**else**
21: $\quad\quad$**for** each $s^\star$ in s' **do**
22: $\quad\quad\quad\mathcal{X}_{s^\star} = \{x \in \mathcal{X} \mid (x, s^\star) \in Q\}$
23: $\quad\quad\quad$STATUS = CHECK-FEASIBILITY($\mathcal{X}_s$, $\mathcal{X}_{s^\star}$, δ_{NN}, CE_s)
24: $\quad\quad\quad$**if** STATUS == `INFEASIBLE` **then**
25: $\quad\quad\quad\quad\delta_{\mathcal{F}}$.REMOVE-TRANSITION($s$, $s^\star$)
26: **Step 3: Compute the safe set**
27: **Step 3.1: Mark the abstract states corresponding to obstacles and workspace boundary as unsafe**

$$\mathcal{F}^0_{\text{unsafe}} = \{s \in \mathcal{F} \mid \exists x \in \mathcal{X} : (x, s) \in Q,\ \zeta(x) \in O_i, O_i \in O\}$$

28: **Step 3.2: Iteratively compute the predecessors of the abstract unsafe states**
29: STATUS = `FIXED-POINT-NOT-REACHED`
30: **while** STATUS == `FIXED-POINT-NOT-REACHED` **do**
31: $\quad\mathcal{F}^k_{\text{unsafe}} = \mathcal{F}^{k-1}_{\text{unsafe}} \cup \text{PRE}(\mathcal{F}^{k-1}_{\text{unsafe}})$
32: $\quad$**if** $\mathcal{F}^k_{\text{unsafe}} == \mathcal{F}^{k-1}_{\text{unsafe}}$ **then**
33: $\quad\quad$STATUS = `FIXED-POINT-REACHED`
34: $\mathcal{F}_{\text{safe}} = \mathcal{F} \setminus \mathcal{F}_{\text{unsafe}}$
35: **Step 3.3: Compute the set of safe states**
36: $\mathcal{X}_{\text{safe}} = \{x \in \mathcal{X} \mid \exists s \in \mathcal{F}_{\text{safe}} : (x, s) \in Q\}$
37: **Return** $\mathcal{X}_{\text{safe}}$

states. Therefore, in an attempt to reduce the number of computational steps in our algorithm, we check the 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 13-19 in

Algorithm 1). 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 21-25 in Algorithm 1).

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 [46, 47], 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 preprocessing of the neural network function δ_{NN} (lines 9-11 in Algorithm 1) 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 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:

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

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

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 27-36 in Algorithm 1.

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.:

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

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.

4.1 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 $O^\star = \bigcup_{O_i \in O} O_i$ 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 $O^\star$ is a unique point characterized as:

$$z_{k,\zeta(x)} = \underset{z \in \mathcal{W}}{\text{argmin}} \|z - \zeta(x)\|_2 \quad \text{s.t.} \quad z \in \textsc{Ray}(\zeta(x),\theta_k) \cap O^\star. \quad (5)$$

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

$$\mathcal{L}_k(\mathcal{R}) = \bigcup_{\zeta(x) \in \mathcal{R}} z_{k,\zeta(x)}. \quad (6)$$

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

Definition 4.1. A set $\mathcal{R} \subset \mathcal{W}$ is said to be an imaging-adapted partition if the following property holds:

$$\mathcal{L}_k(\mathcal{R}) \text{ is a line segment} \quad \forall k \in \{1,\dots,N\}. \quad (7)$$

Figure 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:*

$$d_k(\zeta(x)) = P_{k,\mathcal{R}}\zeta(x) + Q_{k,\mathcal{R}}, \qquad d = (d_1,\dots,d_N) \quad (8)$$

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.

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 2, our algorithm starts by computing a set of line segments $\mathcal{G}$ that will be used to partition the workspace (lines 1-5 in Algorithm 2). 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., $\mathcal{V} = \bigcup_{O_i \in O} \textsc{Vert}(O_i)$. Next, we consider rays originating from all the vertices in $\mathcal{V}$ and pointing in the opposite directions of the angles $\theta_1,\dots,\theta_N$. By intersecting these rays with the obstacles and picking the closest intersection points, we acquire the line segments

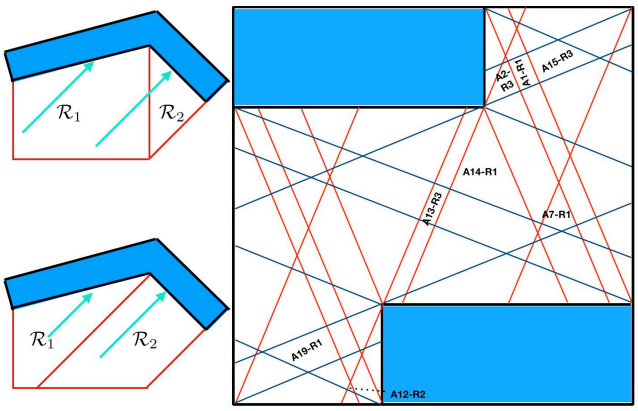

Figure 3: (left-up) A partitioning of the workspace that is *not* imaging-adapted. Within region $\mathcal{R}_1$, the LiDAR ray (cyan arrow) intersects with different obstacle edges depending on the robot position. (left-down) A partitioning of the workspace that is imaging-adapted. For both regions $\mathcal{R}_1$ 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 6.

$\mathcal{G}$ that will be used to partition the workspace. In other words, $\mathcal{G}$ is computed as:

$$\mathcal{G}_k = \{\textsc{Line}(v, z) \mid v \in \mathcal{V},\ z = \operatorname*{argmin}_{z \in \textsc{Ray}(v, \theta_k + \pi) \cap O^\star} \|z - v\|_2\}$$

$$\mathcal{G} = \bigcup_{k=1}^{N} \mathcal{G}_k \tag{9}$$

Thanks to the fact that the vertices v are fixed, finding the intersection between $\textsc{Ray}(v, \theta_k + \pi)$ and $O^\star$ is a standard ray-polytope intersection problem which can be solved efficiently [48].

The next step is to compute the intersection points $\mathcal{P}$ between the line segments $\mathcal{G}$ and the edges of the obstacles $\mathcal{E} = \bigcup_{O_i \in O} \textsc{Edge}(O_i)$. 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 [48]. 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:

$$\text{int}(\mathcal{R}) \cap e = \emptyset \qquad \forall e \in \mathcal{G} \cup \mathcal{E} \tag{10}$$

This process is summarized in lines 8-16 in Algorithm 2. An important property of the regions determined by Algorithm 2 is stated by the following proposition.

Proposition 4.3. *Consider a workspace partition $\mathcal{R}$ that is computed by Algorithm 2 and satisfies* (10)*. 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 (6)*.*

We conclude this section by stating our first main result, quantifying the correctness and complexity of Algorithm 2.

Theorem 4.4. *Given a workspace with polytopic obstacles and a set of laser angles $\theta_1, \ldots, \theta_N$, then Algorithm 2 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 2 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 $\mathcal{G} \cup \mathcal{E}$.

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:

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

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

Algorithm 2 WKSP-PARTITION ($\mathcal{W}, \mathcal{O}, \theta, \theta_p$)

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

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

Step 3: Construct the partitions
Cycles = Find-Vertices-Of-Simple-Cycle(Graph($\mathcal{P}$, $\mathcal{G} \cup \mathcal{E}$))
Cycles′ = Find-Vertices-Of-Simple-Cycle (Graph($\mathcal{P}'$, $\mathcal{G}' \cup \mathcal{E}$)).
for $c \in$ Cycles **do**
  $\mathcal{R}$ = Convex-Hull(c)
  $\mathcal{W}^\star$.ADD($\mathcal{R}$)
for $c \in$ Cycles′ **do**
  $\mathcal{R}'$ = Convex-Hull(c)
  $\mathcal{W}'$.ADD($\mathcal{R}'$)
Return $\mathcal{W}^\star$, $\mathcal{W}'$

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

$$\mathcal{F} = \{(k_1, k_3, \ldots, k_n) \in \mathbb{N}^{n-1} \mid 1 \le k_1 \le r, \quad 1 \le k_i \le \frac{\overline{x}_i - \underline{x}_i}{\epsilon}, i = 3, \ldots, n\} \tag{11}$$

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:

$$Q = \{(x, s) \in \mathcal{X} \times \mathcal{F} \mid s = (k_1, k_3, \ldots, k_n), x = (\zeta(x), x_3, \ldots, x_n), \zeta(x) \in \mathcal{R}_{k_1}, \underline{x}_i + \epsilon(k_i - 1) \le x_i < \underline{x}_i + \epsilon k_i, i = 3, \ldots, n\}. \tag{12}$$

Finally, we define the state transition function $\delta_{\mathcal{F}}$ of $\mathcal{S}_{\mathcal{F}}$ as follows: $(k_1', k_3', \ldots k_n') \in \delta_{\mathcal{F}}((k_1, k_3, \ldots k_n))$ if

$$\exists x = (\zeta(x), x_3, \ldots, x_n) \in \mathcal{R}_{k_1}, \underline{x}_i + \epsilon(k_i - 1) \le x_i < \underline{x}_i + \epsilon k_i,$$
$$x' = (\zeta(x'), x_3', \ldots, x_n') \in \mathcal{R}_{k_1'}, \underline{x}_i + \epsilon(k_i' - 1) \le x_i' < \underline{x}_i + \epsilon k_i',$$
$$\text{s.t.} \quad x' = Ax + Bf_{NN}(d(x)). \tag{13}$$

It follows from the definition of $\delta_{\mathcal{F}}$ in (13) 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.

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 [46, 47] 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:

$$\exists\, x, x' \in \mathbb{R}^n, u \in \mathbb{R}^m, d \in \mathbb{R}^{2N}, \tag{14}$$
$$(b^l, h^l, t^l) \in \mathbb{B}^{M_l} \times \mathbb{R}^{M_l} \times \mathbb{R}^{M_l}, \quad l \in \{1, \ldots, L\}$$

subject to:

$$\zeta(x) \in \mathcal{R}_{k_1} \ \wedge\ \underline{x}_i + \epsilon(k_i - 1) \le x_i < \underline{x}_i + \epsilon k_i,\ i = 3, \ldots, n \tag{15}$$
$$\wedge \zeta(x') \in \mathcal{R}_{k_1'} \wedge \underline{x}_i + \epsilon(k_i' - 1) \le x_i' < \underline{x}_i + \epsilon k_i',\ i = 3, \ldots, n \tag{16}$$
$$\wedge\ x' = Ax + Bu \tag{17}$$
$$\wedge\ d_k = P_{k, \mathcal{R}_{k_1}} \zeta(x) + Q_{k, \mathcal{R}_{k_1}}, \qquad k = 1, \ldots, N \tag{18}$$
$$\wedge \left(t^1 = W^0 d + w^0\right) \ \wedge \ \left(\bigwedge_{l=2}^{L} t^l = W^{l-1}h^{l-1} + w^l\right) \tag{19}$$
$$\wedge \left(u = W^L h^L + w^L\right) \tag{20}$$
$$\wedge \bigwedge_{l=1}^{L} \bigwedge_{j=1}^{M_j} b_j^l \rightarrow \left[\left(h_j^l = t_j^l\right) \wedge \left(t_j^l \ge 0\right)\right] \tag{21}$$
$$\wedge \bigwedge_{l=1}^{L} \bigwedge_{j=1}^{M_j} \neg b_j^l \rightarrow \left[\left(h_j^l = 0\right) \wedge \left(t_j^l < 0\right)\right] \tag{22}$$

where (15)-(16) encode the state space partitions corresponding to the states s and s'; (17) encodes the dynamics of the robot; (18) encodes the imaging function that maps the robot position into LiDAR image; (19)-(22) 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_j^l. 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 [47].

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 (15)-(22) to prune combinations of assignments of the ReLU indicator variables.

To find such constraints, we consider an SMC problem with the fewer constraints (15), (18)-(22). 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:

$$\bigvee_{c \in \mathcal{R}\text{-Conflicts}} \neg c \tag{23}$$

to the original SMC encoding (15)-(22) to prune the set of possible assignments to the ReLU indicator variables. In Section 6, we show that pre-processing would result in several orders of magnitude reduction in the execution time.

5.3 Correctness of Algorithm 1

We end our discussion with the following results which assert the correctness of the whole framework described in this paper. 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 5.1. *Consider the finite state abstraction $\mathcal{S}_{\mathcal{F}} = (\mathcal{F}, \delta_{\mathcal{F}})$ where $\mathcal{F}$ is defined by* (11) *and $\delta_{\mathcal{F}}$ is defined by* (13) *and computed by means of solving the SMC formulas* (15)-(23). *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* (12), *the following holds:* $\mathcal{S}_{NN} \preccurlyeq_Q \mathcal{S}_{\mathcal{F}}$.

Recall that Algorithm 1 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 5.2. *Consider the safe set $\mathcal{X}_{safe}$ computed by Algorithm 1. Then any trajectory η_x with $\eta_x(0) \in \mathcal{X}_{safe}$ is a safe trajectory.*

While Theorem 5.2 establishes the correctness of the proposed framework in Algorithm 1, two points needs to be investigated namely (i) complexity of Algorithm 1 and (ii) maximality of the set $\mathcal{X}_{\text{safe}}$. Although Algorithm 2 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 1 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 paper. In Section 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 1 is not guaranteed to search for the maximal $\mathcal{X}_{\text{safe}}$.

Table 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

6 RESULTS

We implemented the proposed verification framework as described by Algorithm 1 on top of the SMC solver named *SATEX* [49]. All experiments were executed on an Intel Core i7 2.5-GHz processor with 16 GB of memory.

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 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).

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 2, for a neural network with a total of 32 neurons, only 11 ReLU assignments are feasible (within the region under consideration).

Table 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

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 generated counterexamples, along with the number of feasible ReLU assignments are given in Table 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, 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.

Table 3: Execution time of the SMC-based pre-processing as a function of the workspace region. Region indices are shown in Figure 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

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 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.

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 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 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.

ACKNOWLEDGMENTS

The material presented in this paper is based upon work supported by the National Science Foundation award number #1845194.

REFERENCES

[1] Wikipedia, "List of autonomous car fatalities," https://en.wikipedia.org/wiki/List_of_autonomous_car_fatalities.
[2] 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.
[3] T. Everitt, G. Lea, and M. Hutter, "AGI safety literature review," *arXiv preprint*, 2018.
[4] M. Charikar, J. Steinhardt, and G. Valiant, "Learning from untrusted data," in *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. ACM, 2017, pp. 47–60.

Table 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

[5] J. Steinhardt, P. W. W. Koh, and P. S. Liang, "Certified defenses for data poisoning attacks," in *Advances in Neural Information Processing Systems*, 2017, pp. 3520–3532.

[6] 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*. ACM, 2017, pp. 27–38.

[7] A. Paudice, L. Muñoz-González, and E. C. Lupu, "Label sanitization against label flipping poisoning attacks," *arXiv preprint*, 2018.

[8] 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.

[9] 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*. ACM, 2017, pp. 1–18.

[10] 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.

[11] 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*. Springer, 2018, pp. 408–426.

[12] Y. Sun, X. Huang, and D. Kroening, "Testing deep neural networks," *arXiv preprint*, 2018.

[13] 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.

[14] J. Wang, J. Sun, P. Zhang, and X. Wang, "Detecting adversarial samples for deep neural networks through mutation testing," *arXiv preprint*, 2018.

[15] 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.

[16] 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.

[17] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "Deeproad: Gan-based metamorphic autonomous driving system testing," *arXiv preprint*, 2018.

[18] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, "Concolic testing for deep neural networks," *arXiv preprint*, 2018.

[19] T. Dreossi, A. Donzé, and S. A. Seshia, "Compositional falsification of cyber-physical systems with machine learning components," in *NASA Formal Methods Symposium*. Springer, 2017, pp. 357–372.

[20] 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.

[21] 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.

[22] Z. Kurd and T. Kelly, "Establishing safety criteria for artificial neural networks," in *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*. Springer, 2003, pp. 163–169.

[23] S. A. Seshia, D. Sadigh, and S. S. Sastry, "Towards verified artificial intelligence," *arXiv preprint*, 2016.

[24] 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.

[25] J. Leike, M. Martic, V. Krakovna, P. A. Ortega, T. Everitt, A. Lefrancq, L. Orseau, and S. Legg, "AI safety gridworlds," *arXiv preprint*, 2017.

[26] F. Leofante, N. Narodytska, L. Pulina, and A. Tacchella, "Automated verification of neural networks: Advances, challenges and perspectives," *arXiv preprint*, 2018.

[27] 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)*, 2015, pp. 30–40.

[28] 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*. Springer, 2017, pp. 97–117.

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

[30] 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.

[31] W. Ruan, X. Huang, and M. Kwiatkowska, "Reachability analysis of deep neural networks with provable guarantees," *arXiv preprint*, 2018.

[32] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, "Output range analysis for deep feedforward neural networks," in *NASA Formal Methods Symposium*. Springer, 2018, pp. 121–138.

[33] L. Pulina and A. Tacchella, "An abstraction-refinement approach to verification of artificial neural networks," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 243–257.

[34] V. Tjeng and R. Tedrake, "Verifying neural networks with mixed integer programming," *arXiv preprint arXiv:1711.07356*, 2017.

[35] 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.

[36] 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.

[37] 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.

[38] W. Xiang and T. T. Johnson, "Reachability analysis and safety verification for neural network control systems," *arXiv preprint arXiv:1805.09944*, 2018.

[39] 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 International Conference on Hybrid Systems: Computation and Control (HSCC), to appear*. ACM, 2019.

[40] 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.

[41] 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)*. Springer, 2011, pp. 379–395.

[42] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Flow*: An analyzer for non-linear hybrid systems," in *International Conference on Computer Aided Verification (CAV)*. Springer, 2013, pp. 258–263.

[43] 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.

[44] X. Sun, H. Khedr, and Y. Shoukry, "Formal verification of neural network controlled autonomous systems," *arXiv preprint arXiv:1810.13072*, 2018.

[45] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[46] 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)*. ACM, 2017, pp. 19–28.

[47] ——, "Smc: Satisfiability modulo convex programming [40pt]," *Proceedings of the IEEE*, vol. 106, no. 9, pp. 1655–1679, 2018.

[48] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars, *Computational geometry: algorithms and applications*. Springer-Verlag TELOS, 2008.

[49] (2018, Jun.) SatEX Solver. [Online]. Available: https://yshoukry.bitbucket.io/SatEX/