Towards Safe AI: Sandboxing DNNs-Based Controllers in Stochastic Games

Bingzhuo Zhong[*1], **Hongpeng Cao**[*1], **Majid Zamani**[2] , **Marco Caccamo**[1]

[1] Technical University of Munich, Germany
[2] University of Colorado Boulder, USA
{bingzhuo.zhong,cao.hongpeng,mcaccamo}@tum.de, majid.zamani@colorado.edu

Abstract

Nowadays, AI-based techniques, such as deep neural networks (DNNs), are widely deployed in autonomous systems for complex mission requirements (e.g., motion planning in robotics). However, DNNs-based controllers are typically very complex, and it is very hard to formally verify their correctness, potentially causing severe risks for safety-critical autonomous systems. In this paper, we propose a construction scheme for a so-called Safe-visor architecture to sandbox DNNs-based controllers. Particularly, we consider the construction under a stochastic game framework to provide a system-level safety guarantee which is robust to noises and disturbances. A supervisor is built to check the control inputs provided by a DNNs-based controller and decide whether to accept them. Meanwhile, a safety advisor is running in parallel to provide fallback control inputs in case the DNNs-based controller is rejected. We demonstrate the proposed approaches on a quadrotor employing an unverified DNNs-based controller.

1 Introduction

Nowadays, deep neural networks (DNNs) are widely employed as one of the main components in autonomous systems for complex tasks, such as autonomous driving (Bojarski et al. 2016) and human-robot collaboration (El-Shamouty et al. 2020). In these systems, DNNs can serve as perception modules for sensing environments (Liu et al. 2020), as motion planers that provide setpoints for low-level controllers (Aradi 2020), or as low-level controllers that directly map raw sensor readings to control commands for actuators (Haarnoja et al. 2019). However, the behaviors of DNNs are sometimes unpredictable if the deployment settings differ from the training settings (Garcıa and Fernández 2015; Huang et al. 2017), which potentially leads to disastrous consequences in safety-critical scenarios (Brunke et al. 2022; Kwiatkowska 2019). This issue motivated researchers to enter the realm of safe DNNs-enabled systems.

Recent results in (Li and Belta 2019; Lavaei et al. 2020; Kazemi and Soudjani 2020) consider exploiting safety specifications in the training procedure to provide probabilistic safety guarantee for DNNs-based controllers. Concretely, the desired safety specifications are encoded in the reward functions for training DNNs in reinforcement learning settings. However, these results are only applicable when reward functions are easy to be designed, while reward functions for some control tasks are difficult to be obtained (e.g., (Bıyık et al. 2022)). Additionally, various progresses has been made for the verification of DNNs. Nevertheless, verifying DNNs is very challenging and it is an NP-complete problem (Katz et al. 2017). Concretely, the challenges are due to the non-linear activation functions which makes DNNs non-linear and non-convex (Ehlers 2017; Katz et al. 2017). Recent satisfiability modulo theory (SMT)-based approaches (Ehlers 2017; Katz et al. 2017, 2019) and mixed-integer linear program (MILP) optimizers-based approaches (Dutta et al. 2018) can be applied to check if adversarial perturbations over the inputs of the DNNs can change the decisions of the DNNs (Katz et al. 2017). However, these approaches are typically applied on linearized input sets for simple DNNs with a few layers and a few hundred neurons per layer (Ivanov et al. 2019). To reduce the complexities in verifying larger DNNs, (Elboher, Gottschlich, and Katz 2020; Prabhakar and Rahimi Afzal 2019) propose DNNs abstraction approaches, with which one can obtain formal guarantees for the original complex DNNs by verifying the simplified DNNs using existing verification tools (Katz et al. 2019; Tran et al. 2020). Unfortunately, it is still difficult to deploy these methods to verify complex DNNs with millions of parameters and complicated architectures.

Instead of verifying the correctness of DNNs, system-level correct-by-construction schemes (Clavière et al. 2021; Xiang et al. 2018) can be deployed to provide safety guarantees for DNNs-enabled systems regardless of the concrete design of the DNNs. In other words, verification of DNNs is not required under these schemes. For systems with discrete state and input sets, shields are proposed in (Humphrey et al. 2016; Alshiekh et al. 2018) to correct erroneous control inputs and enforce safety properties at runtime. As for enforcing safety invariance properties over systems with continuous state sets (i.e. systems are expected to stay within a fixed safety set), Simplex architecture (Sha 2001; Wang, Hovakimyan, and Sha 2013; Jagtap et al. 2020) allows the deployment of unverified controllers for linear control systems. Model predictive control (MPC)-based approaches are proposed in (Hewing et al. 2020; Wabersich and Zeilinger

[*]These authors contributed equally.

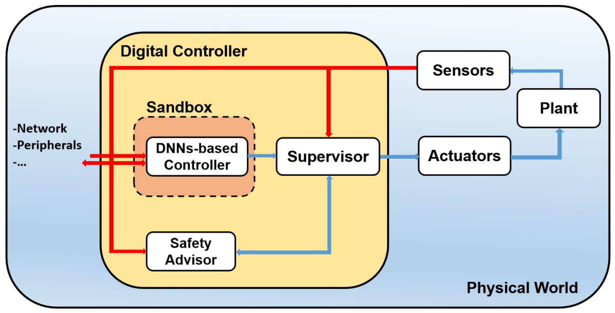

Figure 1: Safe-visor architecture for sandboxing DNNs-based controllers. To apply this architecture, one specifies the minimal desired probability of satisfying the safety specification, denoted by η. At runtime, the supervisor checks the inputs from the DNNs-based controller and only accepts them when η is respected. If the DNNs-based controller is rejected, the safety advisor provides an advisory control command, which maximizes the probability of satisfying the desired safety specification, to actuate the plant.

2018), with which a safety filter is constructed to correct erroneous control inputs provided by the learning-based controllers. Reachability analysis-based approaches are proposed in (Fisac et al. 2018; Ivanov et al. 2019), with which safety guarantees are provided by checking the intersection between the unsafe sets and reachable sets of the systems. Recent results in (Zhong, Zamani, and Caccamo 2019) initialize a system level correct-by-construction architecture, namely *Safe-visor* architecture (cf. Figure 1), utilizing the idea of *sandbox* that is borrowed from the computer security community (Reis, Barth, and Pizano 2009) (see also the technical appendix). While all results above focus on safety invariance properties, the results in (Zhong et al. 2021) further reduce the conservatism of the probabilistic safety guarantees provided in (Zhong, Zamani, and Caccamo 2019), and allows complex logical safety specifications modeled by deterministic finite automata (DFA) (Baier and Katoen 2008), which are widely used to specify properties in robotic applications; see (Maierhofer, Moosbrugger, and Althoff 2022; Yu and Dimarogonas 2022).

In this work, we draw inspiration from (Zhong et al. 2021) and propose an abstraction-based scheme for the construction of Safe-visor architecture to sandbox DNNs-based controllers. In particular, we focus on logical safety specifications characterized by DFAs, and we model the system as general discrete-time stochastic games (gDTSG), which cannot be handled by previous results in (Zhong et al. 2021). Note that gDTSG provides a general framework for modeling various types of systems, including discrete time stochastic control systems (Lavaei et al. 2022), stochastic switched systems (Lavaei, Soudjani, and Zamani 2020), and randomly switched control systems (Patrinos et al. 2014) affected by bounded disturbances. With these settings, we provide a system-level safety guarantee robust to those noises and disturbances that often affect the performance of DNNs in real-life applications. In particular, we propose a generic construction scheme for a Safe-visor architecture under the framework of gDTSGs to sandbox DNNs-based controllers, and provide formal safety guarantees accordingly. Moreover, we demonstrate the effectiveness of the proposed results by controlling a quadrotor to track a ground vehicle by employing a DNNs-based agent. This agent is trained in simulation using deterministic policy gradient algorithm (DDPG) (Lillicrap et al. 2016) to provide setpoints for the low-level controllers of the quadrotor. Experiments in simulations and physical test-bed show that unsafe actions provided by the DNNs-based agent can be effectively corrected using the proposed Safe-visor architecture. The technical appendix of this paper, as well as the code for synthesizing the Safe-visor architecture and training the DNNs-based agent in the case study, are provided in the GitHub repository: https://github.com/Bingzhuo-Zhong/Safe-visor-Stochastic-Game.

2 Preliminaries

2.1 Notations and Concepts

We denote by $\mathbb{R}$ and $\mathbb{N}$ the sets of real and natural numbers, respectively. They are annotated with subscripts to restrict them in a usual way, *e.g.*, $\mathbb{R}_{\geq 0}$ denotes the set of non-negative real numbers. Given a matrix M, $M^\top$ denotes the transpose of M. For $a, b \in \mathbb{R}$ (resp. $a, b \in \mathbb{N}$) with $a \leq b$, the closed, open and half-open intervals in $\mathbb{R}$ (resp. $\mathbb{N}$) are denoted by $[a, b]$, (a, b) ,$[a, b)$ and $(a, b]$, respectively. Given $x \in \mathbb{R}^n$, $\|x\|$ denotes the Euclidean norm of x. Given a set X, $X^{\mathbb{N}}$ denotes the Cartesian product among the countable infinite number of set X. Given sets X, Y, and their Cartesian product $X \times Y$, a relation $\mathscr{R} \subseteq X \times Y$ relates $x \in X$ with $y \in Y$ if $(x, y) \in \mathscr{R}$. Given functions $f : X \to Y$ and $g : Y \to Z$, we denote by $g \circ f : X \to Z$ the composition of f and g. Throughout this paper, we assume that all sets are Borel sets and all functions are measurable. We refer the interested readers to the technical appendix for details of these concepts. Here, we prefer to not dive into these technical assumptions to make the paper more readable.

2.2 Problem Formulation

In this paper, we model the systems as general discrete-time stochastic games (gDTSG) to provide safety guarantees that are robust to noises and disturbances. As a key insight, the stochasticities in the gDTSG model noises. Disturbances are modeled by an adversarial player in the gDTSG with objectives that are apposed to that of controller inputs. Following standard conventions, control and adversary inputs are referred to as Player I and Player II, respectively.

Definition 2.1 *A gDTSG is a tuple $\mathfrak{D} = (X, U, W, X_0, T, Y, h)$, where $X \subseteq \mathbb{R}^s$ is the state set; $U \subset \mathbb{R}^m$ and $W \subset \mathbb{R}^p$ are input sets of Player I and II, respectively; $X_0 \subseteq X$ is the set of initial states; $T : \mathcal{B}(X) \times X \times U \times W \to [0, 1]$ is a conditional stochastic kernel, with which for any $x(k) \in X$, $u(k) \in U$, and $w(k) \in W$, and any set $\mathcal{Q} \subseteq X$, one has*

$$\mathbb{P}\{x(k+1) \in \mathcal{Q} \,|\, x(k), u(k), w(k)\} = \int_{\mathcal{Q}} T(\mathrm{d}x(k+1)|x(k), u(k), w(k)), k \in \mathbb{N},$$

where $\mathbb{P}\{x(k+1) \in \mathcal{Q} \,|\, x(k), u(k), w(k)\}$ denotes the probability of $x(k+1) \in \mathcal{Q}$ given $x(k)$,$u(k)$, and $w(k)$; $Y \subseteq \mathbb{R}^q$ is the output set; and $h : X \to Y$, with $y = h(x)$, is the output function. Alternatively, the stochastic kernel T can be described by: $x(k+1) = f(x(k), u(k), w(k), \varsigma(k))$, in which $\varsigma(k)$, with $k \in \mathbb{N}$, being a sequence of independent and identically distributed (i.i.d.) random variables.

Remark 2.2 *To provide formal safety guarantees regardless of how Player II chooses adversarial inputs, we design a Safe-visor architecture over Player I considering that Player II will select its action after Player I at each time step and in a rational fashion against the choice of Player I. Note that such setting is common for robust control problems. Moreover, we consider that full state information of the gDTSG is available, and safety properties are defined over the output of the gDTSG.*

In this paper, we focus on regular safety properties, denoted by ϕ, over a finite time horizon $H \in \mathbb{N}$. Examples of this class of properties include those expressed by safe linear temporal logic formulae over finite traces (safe-LTL$_F$) (Saha et al. 2014), which are widely deployed to specify safety specifications for autonomous systems (Faruq et al. 2018). Throughout this paper, we refer to this class of properties by tuple (ϕ, H). As explained in (Baier and Katoen 2008), properties (ϕ, H) admit minimal bad prefixes (Kupferman and Vardi 2001, Section 2.2) (see also the technical appendix) recognized by deterministic finite automata (DFA) (Baier and Katoen 2008), as defined below.

Definition 2.3 *A DFA is a tuple $\mathcal{A} = (Q, q_0, \Pi, \tau, F)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, Π is a finite set of alphabet, $\tau : Q \times \Pi \to Q$ is a transition function, and $F \subseteq Q$ is a set of accepting states.*

A *finite word* $\sigma := (\sigma_0, \sigma_1, \ldots, \sigma_{k-1}) \in \Pi^k$ is accepted by $\mathcal{A}$ if there exists a *finite state run* $q := (q_0, q_1, \ldots, q_k) \in Q^{k+1}$ such that $q_{z+1} = \tau(q_z, \sigma_z)$, $\sigma_z \in \Pi$ for all $0 \leq z < k$, and $q_k \in F$. The set of words accepted by $\mathcal{A}$ is called the *language of* $\mathcal{A}$ and denoted by $\mathcal{L}(\mathcal{A})$. Next, we define a labeling function which connects a gDTSG $\mathfrak{D}$ to a DFA $\mathcal{A}$.

Definition 2.4 *Consider a gDTSG $\mathfrak{D} = (X, U, W, X_0, T, Y, h)$, a safety property (ϕ, H) with its associated DFA $\mathcal{A} = (Q, q_0, \Pi, \tau, F)$, and a finite output sequence of $\mathfrak{D}$ defined as*

$$y_H := (y(0), y(1), \ldots, y(H)) \in Y^{H+1}, H \in \mathbb{N}. \quad (2.1)$$

The trace of y_H over Π is $\sigma = \bar{L}(y_H) = (\sigma_0, \ldots, \sigma_H)$, in which $\sigma_k = L(y(k))$ for all $k \in [0, H]$, with labeling functions *$L : Y \to \Pi$ and $\bar{L} : Y^{\mathbb{N}} \to \Pi^{\mathbb{N}}$. Moreover, one has y_H satisfies ϕ, denoted by $y_H \models \phi$, if $\bar{L}(y_H) \notin \mathcal{L}(\mathcal{A})$.*

Intuitively, $y_H \models \phi$ indicates that the accepting states of $\mathcal{A}$ are not reached considering the trace of y_H. Now, we formulate the main problem we aim to solve in this paper.

Problem 2.5 *Consider a gDTSG $\mathfrak{D}$ as in Definition 2.1, and a desired safety specification (ϕ, H). Given the minimal desired probability of satisfying (ϕ, H), denoted by η, design a Safe-visor architecture as in Figure 1 (if existing) for Player I of $\mathfrak{D}$ such that inequality*

$$\mathbb{P}\{y_H \models \phi\} \geq \eta, \quad (2.2)$$

holds regardless of how Player II provides adversarial inputs, where $\mathbb{P}\{y_H \models \phi\}$ denotes the probability of y_H satisfying ϕ, with y_H as in (2.1).

3 Construction of Safe-visor Architecture

In general, the construction of a Safe-visor architecture includes offline computation of a safety advisor enforcing the desired safety specification, and online implementation of a supervisor that decides whether to accepts inputs provided by the DNNs-based controllers. Here, we leverage the results in (Zhong et al. 2023, Section 5) to synthesize the safety advisor. Our main contribution is the construction of a supervisor associated with this safety advisor. To build the safety advisor and the supervisor, a finite abstraction of the original gDTSG is required.

Building a finite abstraction. Given a gDTSG $\mathfrak{D}$, we first construct its finite abstraction, denoted by $\widehat{\mathfrak{D}} = (\hat{X}, \hat{U}, \hat{W}, \hat{X}_0, \hat{T}, Y, \hat{h})$, following (Zhong et al. 2023, Section 4.1). Concretely, we first partition the continuous state and input sets of $\mathfrak{D}$ via finite numbers of bounded cells and select representative points for these cells to construct the finite state and input sets of $\widehat{\mathfrak{D}}$. Then, we compute a matrix $\hat{T}$ to characterize the transitions among the finite states of $\widehat{\mathfrak{D}}$. After $\widehat{\mathfrak{D}}$ is constructed, one needs to check if there exists an (ϵ, δ)-approximate probabilistic relations (Lavaei, Soudjani, and Zamani 2021) between $\mathfrak{D}$ and $\widehat{\mathfrak{D}}$, as defined below.

Definition 3.1 *((ϵ, δ)-approximate probabilistic relations) Consider a gDTSG $\mathfrak{D} = (X, U, W, X_0, Y, h)$ and its finite abstraction $\widehat{\mathfrak{D}} = (\hat{X}, \hat{U}, \hat{W}, \hat{X}_0, \hat{T}, \hat{Y}, \hat{h})$, with $\hat{Y} \subseteq Y$. Then, $\widehat{\mathfrak{D}}$ is (ϵ, δ)-stochastically simulated by $\mathfrak{D}$, denoted by $\widehat{\mathfrak{D}} \preceq^{\delta}_{\epsilon} \mathfrak{D}$, if there exist relations $\mathscr{R} \subseteq X \times \hat{X}$ and $\mathscr{R}_w \subseteq W \times \hat{W}$ s.t.*

- (Cond. 1) $\forall (x, \hat{x}) \in \mathscr{R}$, $\|h(x) - \hat{h}(\hat{x})\| \leq \epsilon$;
- (Cond. 2) *$\forall (x, \hat{x}) \in \mathscr{R}$, and $\forall \hat{u} \in \hat{U}$, $\exists u \in U$ such that $\forall w \in W$, $\exists \hat{w} \in \hat{W}$ with $(w, \hat{w}) \in \mathscr{R}_w$ such that $(x^+, \hat{x}^+) \in \mathscr{R}$ holds with a probability of at least $1 - \delta$, in which $(x, \hat{x})$ and $(x^+, \hat{x}^+)$ are the state pairs at time instants k and $k+1$, respectively, with $k \in \mathbb{N}$;*
- (Cond.3) *$\forall x_0 \in X_0$, $\exists \hat{x}_0 \in \hat{X}_0$ such that $(x_0, \hat{x}_0) \in \mathscr{R}$.*

Note that one can deploy existing results, such as (Zhong et al. 2023, Section 4) and (van Huijgevoort and Haesaert 2022), to check if the relation as in Definition 3.1 exists.

Remark 3.2 *Consider a controller $\hat{\mathbf{C}}$ over $\widehat{\mathfrak{D}}$.* (Cond. 2) *indicates that if an approximate probabilistic relation exists between $\widehat{\mathfrak{D}}$ and $\mathfrak{D}$, then for any $\hat{u} \in \hat{U}$, and $(x, \hat{x}) \in \mathscr{R}$, there exists a stochastic kernel $\mathscr{L}_T((x^+, \hat{x}^+)|x, \hat{x}, \hat{u}, w, \hat{w})$ and an* interface function *$u := \nu(x, \hat{x}, \hat{u}) \in U$ such that if one refines $\hat{\mathbf{C}}$ to $\mathfrak{D}$ with this function, one has $\|y - \hat{y}\| \leq \epsilon$ with a probability of at least $1 - \delta$ at each time step. Note that such distance-based relation plays a crucial role to provide safety guarantees. As a key insight, y is controlled indirectly by controlling $\hat{y}$ while keeping y sufficiently closed to $\hat{y}$ with some probability.*

Construction of the safety advisor. To construct the safety advisor, we first leverage Definition 3.3, which is

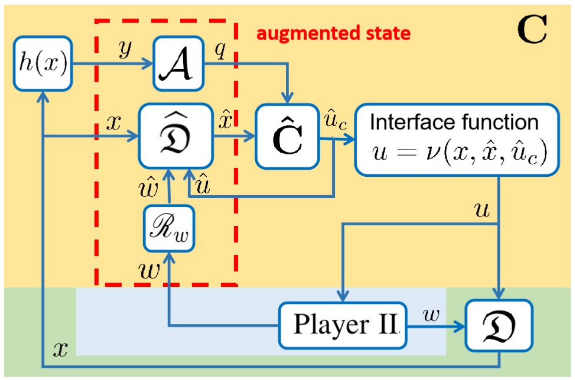

Figure 2: Safety advisor (yellow region), which is a controller over $\mathfrak{D}$ (green region), with augmented state $(x, \hat{x}, q, \hat{u}, w)$ (red dashed rectangle).

adopted from (Zhong et al. 2023), to synthesize a controller, denoted by $\hat{\mathbf{C}}$, over $\widehat{\mathfrak{D}}$.

Definition 3.3 *Consider a gDTSG* $\mathfrak{D}=(X,U,W,X_0,Y,h)$*, its finite abstraction* $\widehat{\mathfrak{D}}=(\hat{X},\hat{U},\hat{W},\hat{X}_0,\hat{T},Y,\hat{h})$ *with* $\widehat{\mathfrak{D}}\preceq_{\epsilon}^{\delta}\mathfrak{D}$*, and a safety property* (ϕ,H) *with its associated DFA* $\mathcal{A}=(Q,q_0,\Pi,\tau,F)$*. A controller* $\hat{\mathbf{C}}:\mathbb{N}\rightrightarrows\hat{U}$ *is constructed s.t.*

$$\hat{\mathbf{C}}(H-n-1)\in \arg\min_{\hat{u}\in\hat{U}}\max_{\hat{w}\in\hat{W}}\Big((1-\delta)\times\sum_{\hat{x}'\in\hat{X}}\underline{V}_{*,n}(\hat{x}',\overline{q}_*(\hat{x}',q))\hat{T}(\hat{x}'|\hat{x},\hat{u},\hat{w})+\delta\Big), \tag{3.1}$$

in which $\underline{V}_{*,n}:\hat{X}\times Q\rightarrow[0,1]$ *is a cost-to-go function initialized as* $\underline{V}_{*,n+1}(\hat{x},q):=1$ *when* $q\in F$*, and* $\underline{V}_{*,n+1}(\hat{x},q):=0$ *otherwise, and recursively computed as*

$$\underline{V}_{*,n+1}(\hat{x},q):=\begin{cases}\min\limits_{\hat{u}\in\hat{U}}\max\limits_{\hat{w}\in\hat{W}}\Big((1-\delta)\sum\limits_{\hat{x}'\in\hat{X}}\underline{V}_{*,n}(\hat{x}',\overline{q}_*(\hat{x}',q))\\ \quad\times\hat{T}(\hat{x}'|\hat{x},\hat{u},\hat{w})+\delta\Big), & \text{if } q\notin F;\\ 1, & \text{if } q\in F,\end{cases} \tag{3.2}$$

in which $\overline{q}_*(\hat{x}',q):=\arg\max_{q'\in Q'_\epsilon(\hat{x}')}\underline{V}_{*,n}(\hat{x}',q')$*, with* $Q'_\epsilon(\hat{x}'):=\{q'\in Q\mid\exists x\in X,q'=\tau(q,L\circ h(x)),\text{with } h(x)\in N_\epsilon(\hat{h}(\hat{x}'))\}$*, and* $\mathcal{N}_\epsilon(\hat{y}):=\{y\in Y\mid\|y-\hat{y}\|\le\epsilon\}$*.*

Using controller $\hat{\mathbf{C}}$, we build the safety advisor $\mathbf{C}$ as in Figure 2. Here, the safety advisor utilizes an augmented state $(x,\hat{x},q,\hat{u},w)$, which contains states x, $\hat{x}$, and q of $\mathfrak{D}$, $\widehat{\mathfrak{D}}$, and $\mathcal{A}$, respectively, the control input $\hat{u}$ fed to $\widehat{\mathfrak{D}}$, and the adversary input w from Player II of $\mathfrak{D}$. The running mechanism of $\mathbf{C}$ at each time step is summarized in Algorithm 1.

Construction of the supervisor. Next, we discuss the design of the supervisor, which is the main contribution of this paper. Given a gDTSG $\mathfrak{D}$ and a safety specification (ϕ,H) with its associated DFA $\mathcal{A}$, the design of the supervisor is depicted in Figure 3. Here, the supervisor consists of a *augmented state* and a *decision maker*. The augmented state of the supervisor, denoted by $(x,\hat{x},q,\hat{u},w)$, is the same as that of the safety advisor, and we simply say the augmented state of the Safe-visor architecture in the rest of this paper for the sake of brevity. At runtime, x, $\hat{x}$, q, and w in the augmented states are updated as described in Algorithm 1. Meanwhile, different from step 8 of Algorithm 1, $\hat{u}$ here is updated as $\hat{u}:=\hat{u}'$, in which $\hat{u}'$ is determined based on the decision of the supervisor (cf. Definition 3.5, either accepting or rejecting the DNNs-based controller). With the augmented state, the decision maker of the supervisor decides whether or not to accept the input from the DNNs-based controller at time instant k, denoted by $u_{dnn}(k)$, in the following way:

Algorithm 1: Running mechanism of the safety advisor.

```
Input: A gDTSG $\mathfrak{D}$, a safety specification $(\phi, H)$ with its associated DFA $\mathcal{A}=(Q,q_0,\Pi,\tau,F)$, safety advisor $\mathbf{C}$, with its associate augmented state $(x,\hat{x},q,\hat{u},w)$, and the current state $x(k)$ of $\mathfrak{D}$.
Output: $u(k)$ for controlling $\mathfrak{D}$.
1: if $k=0$ then
2:     Update $\hat{x}(k)$ such that $(x(k),\hat{x}(k))\in\mathscr{R}$ (cf. (Cond. 3) of Definition 3.1).
3: else
4:     Update $\hat{x}(k)$ according to $x(k)$, the stochastic kernel $\mathscr{L}_T$, and $w(k-1)$ (cf. (Cond. 2) of Definition 3.1 and Remark 3.2).
5: end if
6: Update $q$ of $\mathcal{A}$ as $q(k)=\tau(q(k-1),L\circ h(x(k)))$.
7: Compute $u(k)$ by refining $\hat{u}_c$, which is offered by $\hat{\mathbf{C}}$ based on $\hat{x}(k)$ and $q(k)$, to $\mathfrak{D}$ with the interface function $\nu$ (cf. Figure2).
8: Updates $\hat{u}(k)$ as $\hat{u}(k):=\hat{u}_c$.
9: Update $w(k)$ after Player II has made decision.
```

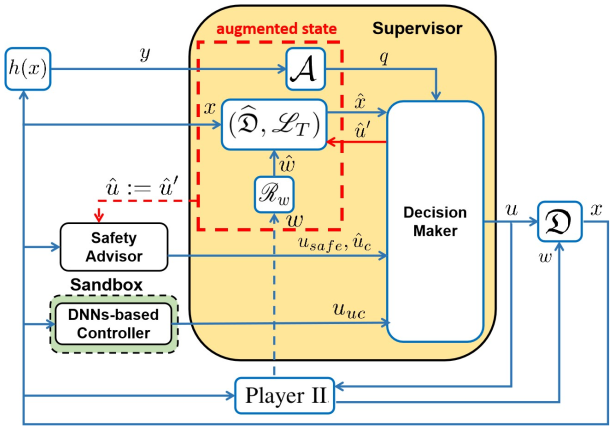

Figure 3: Supervisor in the architecture (yellow region).

- Step 1: Assume that $u_{dnn}(k)$ is accepted. If the (ϵ,δ)-approximate probabilistic relation between $\mathfrak{D}$ and $\widehat{\mathfrak{D}}$ does not hold any more, reject $u_{dnn}(k)$ without going through Step 2 and feed input from the safety advisor, denoted by u_{safe}, to $\mathfrak{D}$; proceed to Step 2, otherwise;
- Step 2: Estimate the probability of satisfying (ϕ,H), de-

noted by $\mathcal{E}_{pv}(k)$, assuming that $u_{dnn}(k)$ is accepted. Accept $u_{dnn}(k)$ if $\mathcal{E}_{pv}(k) \geq \eta$; otherwise, feed u_{safe} to $\mathfrak{D}$.

Step 1 aims at maintaining the (ϵ, δ)-approximate probabilistic relation between $\mathfrak{D}$ and $\widehat{\mathfrak{D}}$, which is crucial for providing safety guarantee (cf. Remark 3.2). One can check Step 1 with the following proposition.

Proposition 3.4 *Consider a gDTSG* $\mathfrak{D} = (X, U, W, X_0, Y, h)$ *and its finite abstraction* $\widehat{\mathfrak{D}} = (\hat{X}, \hat{U}, \hat{W}, \hat{X}_0, \hat{T}, Y, \hat{h})$ *with* $\widehat{\mathfrak{D}} \preceq_{\epsilon}^{\delta} \mathfrak{D}$ *with respect to the relation* $\mathscr{R}$ *and* $\mathscr{R}_w$ *as in Definition 3.1. If the set* U_f *in* (3.3) *is not empty, then the* (ϵ,δ)*-approximate probabilistic relation can be maintained between* $\mathfrak{D}$ *and* $\widehat{\mathfrak{D}}$ *at the time instant* $k+1$ *when* $u_{dnn}(k)$ *is applied to* $\mathfrak{D}$ *at the time instant* k*.*

For the set U_f defined in (3.3), f and $\hat{f}$ are defined as in Definition 2.1, and $x(k) \in X$, $\hat{x}(k) \in \hat{X}$, $\varsigma(k)$ and $\hat{\varsigma}(k)$ denote current states of $\mathfrak{D}$ and $\widehat{\mathfrak{D}}$, noises affecting $\mathfrak{D}$ and $\widehat{\mathfrak{D}}$, respectively. As a key insight, if $U_f \neq \emptyset$, then, by definition of U_f, there exists at least one $\hat{u} \in \hat{U}$ corresponding to $u_{dnn}(k)$ such that $(x(k+1), \hat{x}(k+1)) \in \mathscr{R}$ holds with the probability of at least $1-\delta$, indicating that the approximate probabilistic relation between $\mathfrak{D}$ and $\widehat{\mathfrak{D}}$ is maintained (cf. (Cond.2) of Definition 3.1). In general, checking the non-emptiness of U_f depends on the concrete form of f. In Section 4, we show how to check whether U_f is empty using Proposition 3.4 via a case study. With Proposition 3.4, we present the supervisor for Problem 2.5 as follows.

Definition 3.5 *Consider a gDTSG* $\mathfrak{D} = (X, U, W, X_0, Y, h)$ *and its finite abstraction* $\widehat{\mathfrak{D}} = (\hat{X}, \hat{U}, \hat{W}, \hat{X}_0, \hat{T}, Y, \hat{h})$ *with* $\widehat{\mathfrak{D}} \preceq_{\epsilon}^{\delta} \mathfrak{D}$*, a safety specification* (ϕ, H) *with its associated DFA* $\mathcal{A} = (Q, q_0, \Pi, \tau, F)$*, a labeling function* $L : Y \rightarrow \Pi$ *associated with* $\mathcal{A}$ *as in Definition 2.4, and* η *to be the minimal desired probability of satisfying* (ϕ, H)*. For all* $k \in [0, H-1]$*, the validity of an input* $u_{dnn}(k)$ *from the DNNs-based controller is checked as follows:*

(i) *Reject* $u_{dnn}(k)$ *if* U_f *as in* (3.3) *is empty;*

(ii) *If* U_f *is not empty, compute* $\mathcal{E}_{pv}(k)$ *as*

$$\mathcal{E}_{pv}(k) := \mathcal{C}_1(k)\mathcal{C}_2(k), \tag{3.6}$$

with $\mathcal{C}_1(k)$ *and* $\mathcal{C}_2(k)$ *as in* (3.4) *and* (3.5)*, respectively,* $\underline{V}_{*,H-k-1}$ *and* $\bar{q}_*$ *as in Definition 3.3,*

$$\hat{u}^* := arg \min_{\hat{u} \in U_f} \mathcal{E}_{pv}(k), \tag{3.7}$$

and $\hat{X}'_{-\epsilon}(q(z-1)) := \{\hat{x} \in \hat{X} \,|\, \forall x \in X, \tau(q(z-1), L \circ h(x)) \notin F, h(x) \in N_{\epsilon}(\hat{h}(\hat{x}))\}$*, in which* $N_{\epsilon}(\hat{h}(\hat{x}))$ *is as in Definition 3.3. If* $\mathcal{E}_{pv}(k) \geq \eta$*, the supervisor accepts* $u_{dnn}(k)$ *and update the augmented state with* $\hat{u}' := \hat{u}^*$ *(cf. Figure 3), with* $\hat{u}^*$ *being computed as in* (3.7)*; otherwise, it rejects* $u_{dnn}(k)$ *and set* $\hat{u}'$ *as* $\hat{u}' := \hat{u}_c$*, with* $\hat{u}_c$ *provided by the safety advisor (cf. Algorithm 1).*

By leveraging the supervisor in Definition 3.5, we propose the main result of this paper.

Theorem 3.6 *Consider a gDTSG* $\mathfrak{D} = (X, U, W, X_0, Y, h)$ *and a safety specification* (ϕ, H)*. By leveraging the supervisor in Definition 3.5 at all time* $k \in [0, H-1]$ *in the Safe-visor architecture for Player I of* $\mathfrak{D}$*, one has*

$$\mathbb{P}\{y_H \models \phi\} \geq \eta, \tag{3.8}$$

regardless of how Player II provides adversarial inputs, with y_H *being output sequences of* $\mathfrak{D}$ *as in* (2.1)*.*

Detailed proofs of Theorem 3.6 are provided in the technical appendix, and we are providing here a sketch of proof. **Sketch of Proof of Theorem 3.6** Consider a DFA $\mathcal{A} = (Q, q_0, \Pi, \tau, F)$ associated with (ϕ, H).

- $\mathcal{C}_1(k)$ denotes the minimal probability of F not being reached (i.e. $y_H \models \phi$) over the time horizon $[0, k]$, while one has $(x(z), \hat{x}(z)) \in \mathscr{R}$, $\forall z \in [0, k]$. Considering the definition of $\hat{X}'_{-\epsilon}(q(z-1))$ as in Definition 3.5, one can verify $\forall \hat{x} \in \hat{X}'_{-\epsilon}(q(z-1))$ with $q(z-1) \in Q$, $\nexists x$ with $(x, \hat{x}) \in \mathscr{R}$ such that F is reached at time step z. In other words, if $\hat{x}(z) \in \hat{X}'_{-\epsilon}(q(z-1))$, one can ensure that F is not reached at the time z by ensuring $(x(z), \hat{x}(z)) \in \mathscr{R}$. Hence, given $\hat{x}(z-1)$, $\hat{u}(z-1)$, and $q(z-1)$,

$$\mathcal{C}'(z) := \min_{\hat{w} \in \hat{W}} (1-\delta) \sum_{\hat{x} \in \hat{X}'_{-\epsilon}(q(z-1))} \hat{T}(\hat{x} \,|\, \hat{x}(z-1), \hat{u}(z-1), \hat{w}), \tag{3.9}$$

  denotes the minimal probability of F not being reached at time z while $(x(z), \hat{x}(z)) \in \mathscr{R}$ still holds.
- $\mathcal{C}_2(k)$ is the probability of F not being reached over the time horizon $[k+1, H]$, while (i) $(x(k+1), \hat{x}(k+1)) \in \mathscr{R}$ holds, given $\hat{x}(k)$, $q(k)$, and $\hat{u}(k) = \hat{u}^*$; (ii) $\mathfrak{D}$ is controlled by the safety advisor within $[k+1, H]$.

Hence, $\mathcal{C}_1(k)\mathcal{C}_2(k)$ denotes the lower bound on the probability of F not being reached over $[0, H]$. Since (3.7) ensures that all $\hat{u}$ which maintains the (ϵ,δ)-approximate probabilistic relation between $\mathfrak{D}$ and $\widehat{\mathfrak{D}}$ would not result in $\mathcal{E}_{pv} < \eta$. By accepting $u_{dnn}(k)$ at each time instant k where $\mathcal{C}_1(k)\mathcal{C}_2(k) \geq \eta$ holds, one can ensure that (3.8) holds. ■

Remark 3.7 *By leveraging the results in (Zhong et al. 2023, Corollary 5.12), one can obtain an upper bound on* $\mathbb{P}\{y_H \models \phi\}$*, denoted by* $\mathbf{v} \in [0,1]$*, when the system starts from the initial state set* X_0 *and the safety advisor is applied over the whole time horizon* $[0, H]$*. Accordingly,* (3.8) *is achievable whenever* $\eta \geq \mathbf{v}$*.*

Finally, we summarized in Algorithm 2 the running mechanism of the proposed Safe-visor architecture at each time step k. Moreover, we should add that the number of operations required for computing $\mathcal{E}_{pv}(k)$ in (3.6) is proportional to cardinality of sets $\hat{X}$, $\hat{W}$, and U_f. Concretely,

- $\mathcal{C}_1(k)$: One can verify that $\mathcal{C}_1(k) = \mathcal{C}_1(k-1)\mathcal{C}'(k)$, with $\mathcal{C}'(k)$ as in (3.9). Since $\mathcal{C}_1(k-1)$ has already been computed at time step $k-1$, one only needs $\mathcal{C}'(k)$ to obtain $\mathcal{C}_1(k)$ at time k. On the other hand, $\forall q \in Q$, set $\hat{X}'_{-\epsilon}(q)$ can be computed offline, and $\hat{T}$ is readily computed when constructing the finite abstraction of the original gDTSG. Hence, the number of operations required for computing $\mathcal{C}_1(k)$ at time instant k is proportional to the cardinality of the set $\hat{X}'_{-\epsilon}(q(k-1))$ and $\hat{W}$.

$$U_f := \Big\{ \hat{u} \in \hat{U} \,\big|\, \forall (w,\hat{w}) \in \mathscr{R}_w, \mathbb{P}\big\{(x',\hat{x}') \in \mathscr{R}\big\} \geq 1-\delta \text{ holds, with } x' = f(x(k), u_{dnn}(k), w, \varsigma(k)), \hat{x}' = \hat{f}(\hat{x}(k), \hat{u}, \hat{w}, \hat{\varsigma}(k)) \Big\}, \tag{3.3}$$

$$\mathcal{C}_1(k) := \prod_{z=1}^{k} \Big((1-\delta) \min_{\hat{w} \in \hat{W}} \sum_{\hat{x} \in \hat{X}'_{-\epsilon}(q(z-1))} \hat{T}\big(\hat{x} \,\big|\, \hat{x}(z-1), \hat{u}(z-1), \hat{w}\big) \Big), \tag{3.4}$$

$$\mathcal{C}_2(k) := (1-\delta)\Big(1 - \max_{\hat{w} \in \hat{W}} \sum_{\hat{x} \in \hat{X}} \underline{V}_{*,H-k-1}\big(\hat{x}, \bar{q}_*(\hat{x}(k), q(k))\big) \hat{T}\big(\hat{x} \,\big|\, \hat{x}(k), \hat{u}^*, \hat{w}\big)\Big), \tag{3.5}$$

Algorithm 2: Running mechanism of the Safe-visor.

Input: A gDTSG $\mathfrak{D}$, specification (ϕ, H) with its associated DFA $\mathcal{A}$, safety advisor **C** as in Figure 2, supervisor as in Definition 3.5, and $u_{dnn}(k)$ from the DNNs-based controller
Output: $u(k)$ for controlling $\mathfrak{D}$

```
1: Compute $\mathcal{C}_1(k)$ as in (3.4).
2: Update $x(k)$, $\hat{x}(k)$, and $q(k)$ in the augmented state as in Algorithm 1.
3: Decide whether to accept $u_{dnn}(k)$.
4: if $u_{dnn}(k)$ is accepted then
5:     Set $u(k) = u_{dnn}(k)$ and update $\hat{u}(k)$ in the augmented state as $\hat{u}(k) = \hat{u}^*$ with (3.7).
6: else
7:     Obtain $u_{\text{safe}}(k)$ and $\hat{u}_c(k)$ from the safety advisor, set $u(k) = u_{\text{safe}}(k)$, and update $\hat{u}(k)$ in the augmented state as $\hat{u}(k) = \hat{u}_c(k)$.
8: end if
9: Update $w(k)$ in the augmented state based on the decision of Player II.
```

- $\mathcal{C}_2(k)$: Since $\underline{V}_{*,H-k-1}$ and $\hat{T}$ have already been computed when synthesizing the safety advisor, the number of operations required for computing $\mathcal{C}_2(k)$ is proportional to the number of elements in sets $\hat{X}$, $\hat{W}$, and U_f.

The real-time applicability of the Safe-visor architecture will be shown in Section 4 via a case study (cf. Table 1).

4 Case Studies

To showcase the proposed construction scheme, we cosider a case study of controlling a quadrotor using a DNNs-based agent to track a ground vehicle in 1) simulation with 1.0×10^4 different realization of noise; 2) experiment on the physical test-bed. The physical test-bed includes: 1) a

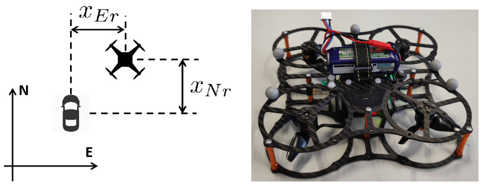

Figure 4: Case study for controlling a quadrotor tracking a ground vehicle.

quadrotor as in Figure 4 (right); 2) Vicon motion capture system for capturing the position and velocity of the quadrotor at runtime; and 3) a ground control station (GCS) with Ubuntu 20.04 (Intel Core i9-10900K CPU (3.7 GHz) and 32 GB of RAM). The simulations are performed via MATLAB 2019b on the GCS. In the experiment on the physical test-bed, the safety advisor, the supervisor, and the DNNs-based controller are running on the GCS. Based on the decision of the supervisor, the GCS sends desired accelerations (i.e. the control input, cf. (4)) to the quadrotor at runtime.

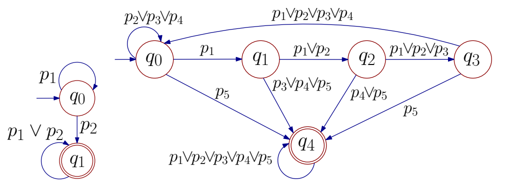

Figure 5: Left: DFA $\mathcal{A}_E$, with accepting state q_1, alphabet $\Pi = \{p_1, p_2\}$, and labeling function $L : Y \to \Pi$ with $L(y) = p_1$ when $y \in [-0.5, 0.5]$, and $L(y) = p_2$ when $y \in (-\infty, -0.5) \cup (0.5, +\infty)$. Right: DFA $\mathcal{A}_N$, with accepting state q_4, alphabet $\Pi = \{p_1, p_2, p_3, p_4, p_5\}$, and labeling function $L : Y \to \Pi$ with $L(y) = p_1$ when $y \in [-0.3, 0.3]$, $L(y) = p_2$ when $y \in [-0.4, -0.3) \cup (0.3, 0.4]$, $L(y) = p_3$ when $y \in [-0.45, -0.4) \cup (0.4, 0.45]$, $L(y) = p_4$ when $y \in [-0.5, -0.45) \cup (0.45, 0.5]$, and $L(y) = p_5$ when $y \in (-\infty, -0.5) \cup (0.5, +\infty)$.

Modeling and safety specifications: By employing the feedback linearization technique in (Ghaffari 2021), the relative motion between the quadrotor and the ground vehicle on N and E axes (see Figure 4 (left)) can be modeled as:

$$\begin{cases} x_{\mathsf{i}}(k+1) = Ax_{\mathsf{i}}(k) + Bu_{\mathsf{i}}(k) + Dw_{\mathsf{i}}(k) + R\varsigma_{\mathsf{i}}(k), \\ y_{\mathsf{i}}(k) = Cx_{\mathsf{i}}(k), \qquad k \in \mathbb{N}, \mathsf{i} \in \{N, E\}, \end{cases} \tag{4.1}$$

where $A = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$, $B = [\frac{\Delta t^2}{2}; \Delta t]$, $D = -B$, and $C = [1; 0]^\top$, with $\Delta t = 0.1s$ being the sampling time, and $R = \begin{bmatrix} 0.004 & 0 \\ 0 & 0.045 \end{bmatrix}$ being obtained through experimental trials on our physical test-bed. Here, for $\mathsf{i} \in \{N, E\}$, $x_{\mathsf{i}}(k) := [x_{\mathsf{i}r}(k); v_{\mathsf{i}r}(k)]$ with $x_{\mathsf{i}r}(k)$ and $v_{\mathsf{i}r}(k)$ being the relative position and relative velocity between the quadrotor and the vehicle on i axis, respectively; $u_{\mathsf{i}}(k) \in [-2.5, 2.5]$ (m/s^2) denotes the acceleration of the quadrotor on i axis as the control input; $w_{\mathsf{i}}(k) \in [-0.6, 0.6]$ (m/s^2) denotes the acceleration of the vehicle on i axis as the adversary input; $\varsigma_{\mathsf{i}}(k)$ is a standard Gaussian random variable; and $y_{\mathsf{i}}(k)$ is the output. Within 1 min (time horizon $H = 600$), the following safety specifications are desired: (1) (ϕ_E, H): y_E should be within $[-0.5, 0.5]$ (m); (2) (ϕ_N, H): y_N should be within $[-0.5, 0.5]$ (m); additionally, if y_N reaches $[-0.3, 0.3]$ (m) at any time instant k, then y_N should be within $[-0.4, 0.4]$ (m) at time instant $k+1$ and within $[-0.45, 0.45]$ (m) at time instant $k+2$, instead of $[-0.5, 0.5]$ (m). Their associated DFAs $\mathcal{A}_E$ and $\mathcal{A}_N$ are shown in Figure 5. Next, we design the Safe-visor architecture with respect to (ϕ_E, H) and (ϕ_N, H), denoted by sva_E and sva_N, respectively.

Construction of Safe-visor architecture: To construct the safety advisor as introduced in Section 3, we first build the finite abstraction of the model as in (4.1) by selecting $X = [-0.5, 0.5] \times [-0.4, 0.4]$ and partition it uniformly with grid cells whose sizes are $(0.02, 0.02)$. Then, we uniformly divide the input set $[-2.5, 2.5]$ for the quadrotor and the input set $[-0.6, 0.6]$ for the ground vehicle with 25 and 12 cells, respectively. As a result, we get a finite gDTSG with 2001 states (denoted by $\hat{X}$), 25 control inputs for Player I (denoted by $\hat{U}'$), and 12 adversarial inputs for Player II (denoted by $\hat{W}$). By employing the results in (Zhong et al. 2023, Section 4.3), the finite abstraction is (ϵ, δ)-stochastically simulated by the original model with respect to the relation $\mathscr{R} := \{(x, \hat{x}) \mid (x - \hat{x})^\top M (x - \hat{x}) \leq \epsilon^2\}$, and $\mathscr{R}_w := \{(w, \hat{w}) | (w - \hat{w})^\top (w - \hat{w}) \leq \tilde{\epsilon}^2\}$, with $\delta = 0$, $\epsilon = 0.0674$, $\tilde{\epsilon} = 0.05$, $M = \left[\begin{smallmatrix} 1.4632 & 0.1757 \\ 0.1757 & 0.0666 \end{smallmatrix}\right]$, and a interface function $u := K(x - \hat{x}) + \hat{u}$ where $K = [-16.66; -4.83]^\top$ (cf. Figure 2), and $\hat{U} := \{\hat{u} \in \hat{U}' | ||\hat{u}|| \leq 0.12\}$ is used to build the safety advisor. Having the finite abstraction and the approximate probabilistic relation, we synthesize the safety advisors for sva_E and sva_N as discussed in Section 3. The total offline computation time[1] for sva_E and sva_N are approximately 1.2 hours and 5.2 hours, respectively.

After the safety advisors are constructed offline, we implement the supervisors leveraging the results in Theorem 3.6, for which checking the non-emptiness of the set U_f in (3.3) at runtime is necessary (cf. Proposition 3.4). Consider the current state $x(k)$ of the original system, $\hat{x}(k)$ in the current state of the Safe-visor architecture, and input $u_{dnn}(k)$ provided by the DNNs-based controller. The set U_f is not empty if there exists $\hat{u} \in \hat{U}$ such that

$$\begin{aligned} \|Ax(k) + Bu_{dnn}(k) + Dw(k) + R\varsigma(k) - \\ (A\hat{x}(k) + B\hat{u} + D\hat{w}(k) + R\hat{\varsigma}(k))\|_M \leq \epsilon, \quad (4.2) \end{aligned}$$

holds for all $\varsigma \in \mathbb{R}^2$, with $\|\bar{x}\|_M := \sqrt{\bar{x}^T M \bar{x}}$. By setting $\hat{\varsigma}(k) = \varsigma(k)$, (4.2) holds if one has $\|\varphi - B\hat{u}\|_M \leq \epsilon - \gamma$, with $\varphi := A(x(k) - \hat{x}(k)) + Bu_{dnn}(k)$ and $\gamma := \max_{\beta \in \Delta} \|\beta\|_M + \max_{(w,\hat{w}) \in \mathscr{R}_w} \|D(w - \hat{w})\|_M = 0.0152$, where Δ is the set of all possible quantization errors introduced by discretization of the original state set. Since φ can readily be computed at runtime, one can find out if there exists $\hat{u} \in \hat{U}$ such that (4.2) holds efficiently.

[1]The computation of the cost-to-go function $\underline{V}_{*,n+1}(\hat{x}, q)$ in (3.2) for different $(\hat{x}, q) \in \hat{X} \times Q$ are independent from each other. Therefore, the computation can be done in a parallel fashion, which is not implemented in the current code.

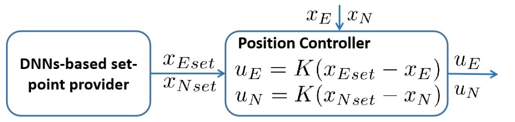
x_E
x_N
DNNs-based set-point provider
x_{Eset}
x_{Nset}
Position Controller
$u_E = K(x_{Eset} - x_E)$
$u_N = K(x_{Nset} - x_N)$
u_E
u_N

Figure 6: DNNs-based controller in real-world experiments.

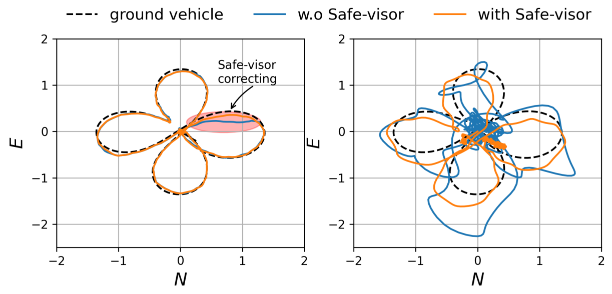

Figure 7: Trajectories of the quadrotor and the ground vehicle in simulation (Left), and real-world experiment(Right).

Experiments and results In the experiments, we consider using a DNNs-based agent to control the quadrotor to track the vehicle. The agent is trained as a setpoints provider for low-level position controller, as depicted in Figure 6, with $K = [1.4781; 1.7309]^\top$. The agent takes the current positions and velocities of the quadrotor and the ground vehicle as inputs, and provides the position and velocity setpoints for the quadrotor. We leverage DDPG algorithm (Lillicrap et al. 2016) to train the agent in simulation, in which the vehicle follows random trajectories. Here, we refer the readers to the technical appendix for more training details.

In both simulation and the experiment on the test-bed, the ground vehicle follows a clover trajectory[2], and we initialize the system with $x_E = x_N = [0; 0]$ and set the minimal desired probability of satisfaction for sva_E and sva_N as $\eta_E = \eta_N = 0.99$. The results of the simulation and the experiment on the physical test-bed are summarized in Table 1. The simulation results show that the desired lower bound of safety probability specified by η_E and η_N are respected, while more than 90% of the inputs from the DNNs-based controllers are accepted. Although the quadrotor tracks the vehicle very well in the simulation without actually violating the safety specification, as marked red in the Figure 7(Left), the architecture rejects some potential risky actions due to the robustness settings as in Remark 2.2. Notably, as shown in Figure 7(Right), the DNNs-based controller behaves worse on the physical test-bed and the safety specifications are violated when the Safe-visor architecture is not deployed (see the bottom part of Figure 8, in which y_{Nr} left the region $[-0.5, 0.5]$), which might be attributed to the mismatch between the model that is used for training and the physical system. For instance, the environmental noises and disturbances are not considered in simulation training,

[2]It is challenging to design adversarial strategies for the ground vehicle as in Remark 2.2, while the probabilistic guarantees provided are valid regardless of how the ground vehicle select inputs.

Safety specifications	P_s (with Safe-visor)	Acceptance rate	P_s (without Safe-visor)	T_{avg} (ms)	T_σ (ms)
(ϕ_E, H) (Simulation)	100%	92.75%	100%	3.0790	1.3873
(ϕ_N, H) (Simulation)	100%	92.40%	100%	3.3238	1.3415
(ϕ_E, H) (Test-bed)	100%	2.50%	0%	4.2301	2.1056
(ϕ_N, H) (Test-bed)	100%	8.00%	0%	3.3957	2.7630

Table 1: Results of simulation and experiment on the physical test-bed, where P_s denotes the percentage of the outputs satisfying the desired safety properties; acceptance rate is the percentage of inputs from the DNNs-based controller being accepted among different runs; T_{avg} and T_σ are the average and the standard deviation of the execution time, respectively.

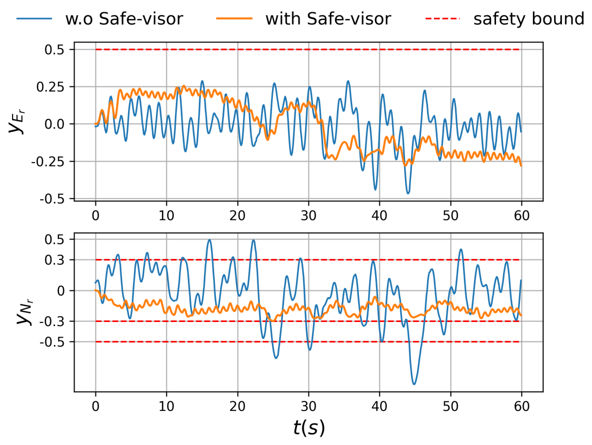

Figure 8: Evolution of y_E and y_N with and without using the Safe-visor architecture.

as it requires robust training strategies, causing difficulties in convergence. Meanwhile, by leveraging the Safe-visor architecture, the desired safey specifications are enforced, while the DNNs-based controller can still be employed.

5 Conclusions

In this paper, we proposed, for the first time, the construction of a Safe-visor architecture for sandboxing DNNs-based controllers in stochastic games with continuous state and input sets. This architecture consists of 1) a supervisor that checks the inputs provided by the DNNs-based controller and rejects them whenever they endanger the overall safety of the systems; 2) a safety advisor that provides inputs maximizing the probability of satisfying the desired safety specifications if the DNNs-based controller is rejected. Both components are built using abstraction-based approaches, and formal safety guarantees are provided based on the approximate probabilistic relation between the original game and its finite abstraction. The compromise between safety and functionality is achieved by setting a minimal desired probability of satisfying the safety specification. Finally, the effectiveness of our results is demonstrated via simulations and experiments on a physical test-bed for a quadrotor.

Acknowledgments

This work was supported in part by the National Science Foundation (NSF) under grant CNS-2039062 and by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research.

References

Alshiekh, M.; Bloem, R.; Ehlers, R.; Könighofer, B.; Niekum, S.; and Topcu, U. 2018. Safe Reinforcement Learning via Shielding. In *Proceedings of Thirty-Second AAAI Conference on Artificial Intelligence*.

Aradi, S. 2020. Survey of deep reinforcement learning for motion planning of autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*.

Baier, C.; and Katoen, J.-P. 2008. *Principles of model checking*. MIT press.

Bıyık, E.; Losey, D. P.; Palan, M.; Landolfi, N. C.; Shevchuk, G.; and Sadigh, D. 2022. Learning reward functions from diverse sources of human feedback: Optimally integrating demonstrations and preferences. *The International Journal of Robotics Research*, 41(1): 45–67.

Bojarski, M.; Del Testa, D.; Dworakowski, D.; Firner, B.; Flepp, B.; Goyal, P.; Jackel, L. D.; Monfort, M.; Muller, U.; Zhang, J.; et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

Brunke, L.; Greeff, M.; Hall, A. W.; Yuan, Z.; Zhou, S.; Panerati, J.; and Schoellig, A. P. 2022. Safe learning in robotics: From learning-based control to safe reinforcement learning. *Annual Review of Control, Robotics, and Autonomous Systems*, 5: 411–444.

Clavière, A.; Asselin, E.; Garion, C.; and Pagetti, C. 2021. Safety Verification of Neural Network Controlled Systems. In *Proceedings of 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops*, 47–54.

Dutta, S.; Jha, S.; Sankaranarayanan, S.; and Tiwari, A. 2018. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*, 121–138. Springer.

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

El-Shamouty, M.; Wu, X.; Yang, S.; Albus, M.; and Huber, M. F. 2020. Towards safe human-robot collaboration using deep reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 4899–4905. IEEE.

Elboher, Y. Y.; Gottschlich, J.; and Katz, G. 2020. An abstraction-based framework for neural network verification. In *International Conference on Computer Aided Verification*, 43–65. Springer.

Faruq, F.; Parker, D.; Laccrda, B.; and Hawes, N. 2018. Simultaneous Task Allocation and Planning Under Uncertainty. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3559–3564. IEEE.

Fisac, J. F.; Akametalu, A. K.; Zeilinger, M. N.; Kaynama, S.; Gillula, J.; and Tomlin, C. J. 2018. A general safety framework for learning-based control in uncertain robotic systems. *IEEE Transactions on Automatic Control*, 64(7): 2737–2752.

Garcıa, J.; and Fernández, F. 2015. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1): 1437–1480.

Ghaffari, A. 2021. Analytical Design and Experimental Verification of Geofencing Control for Aerial Applications. *IEEE/ASME Transactions on Mechatronics*, 26: 1106–1117.

Haarnoja, T.; Ha, S.; Zhou, A.; Tan, J.; Tucker, G.; and Levine, S. 2019. Learning to Walk Via Deep Reinforcement Learning. In *Robotics: Science and Systems*.

Hewing, L.; Wabersich, K. P.; Menner, M.; and Zeilinger, M. N. 2020. Learning-based model predictive control: Toward safe learning in control. *Annual Review of Control, Robotics, and Autonomous Systems*, 3: 269–296.

Huang, S. H.; Papernot, N.; Goodfellow, I. J.; Duan, Y.; and Abbeel, P. 2017. Adversarial Attacks on Neural Network Policies. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*.

Humphrey, L.; Könighofer, B.; Könighofer, R.; and Topcu, U. 2016. Synthesis of Admissible Shields. In *Hardware and Software: Verification and Testing. Lecture Notes in Computer Science*, 134–151. Springer.

Ivanov, R.; Weimer, J.; Alur, R.; Pappas, G. J.; and Lee, I. 2019. 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*, 169–178.

Jagtap, P.; Abdi, F.; Rungger, M.; Zamani, M.; and Caccamo, M. 2020. Software Fault Tolerance for Cyber-Physical Systems via Full System Restart. *ACM Transactions on Cyber-Physical Systems*, 4(4): 1–20.

Katz, G.; Barrett, C.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International conference on computer aided verification*, 97–117. Springer.

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

Kazemi, M.; and Soudjani, S. 2020. Formal Policy Synthesis for Continuous-State Systems via Reinforcement Learning. In *Proceedings of International Conference on Integrated Formal Methods*, 3–21.

Kupferman, O.; and Vardi, M. Y. 2001. Model checking of safety properties. *Formal Methods in System Design*, 19(3): 291–314.

Kwiatkowska, M. Z. 2019. Safety Verification for Deep Neural Networks with Provable Guarantees (Invited Paper). In *Proceedings of 30th International Conference on Concurrency Theory*, volume 140, 1–5.

Lavaei, A.; Somenzi, F.; Soudjani, S.; Trivedi, A.; and Zamani, M. 2020. Formal controller synthesis for continuous-space MDPs via model-free reinforcement learning. In *Proceedings of ACM/IEEE 11th International Conference on Cyber-Physical Systems*, 98–107.

Lavaei, A.; Soudjani, S.; Abate, A.; and Zamani, M. 2022. Automated verification and synthesis of stochastic hybrid systems: A survey. *Automatica*, 146: 110617.

Lavaei, A.; Soudjani, S.; and Zamani, M. 2020. Compositional abstraction-based synthesis for networks of stochastic switched systems. *Automatica*, 114: 108827.

Lavaei, A.; Soudjani, S.; and Zamani, M. 2021. Compositional Abstraction-based Synthesis of General MDPs via Approximate Probabilistic Relations. *Nonlinear Analysis: Hybrid Systems*, 39.

Li, X.; and Belta, C. 2019. Temporal Logic Guided Safe Reinforcement Learning Using Control Barrier Functions. *arXiv:1903.09885v1*.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2016. Continuous control with deep reinforcement learning. In *Proceedings of International Conference on Learning Representations(Poster)*.

Liu, L.; Ouyang, W.; Wang, X.; Fieguth, P.; Chen, J.; Liu, X.; and Pietikäinen, M. 2020. Deep learning for generic object detection: A survey. *International journal of computer vision*, 128(2): 261–318.

Maierhofer, S.; Moosbrugger, P.; and Althoff, M. 2022. Formalization of Intersection Traffic Rules in Temporal Logic. In *2022 IEEE Intelligent Vehicles Symposium (IV)*, 1135–1144.

Patrinos, P.; Sopasakis, P.; Sarimveis, H.; and Bemporad, A. 2014. Stochastic model predictive control for constrained discrete-time Markovian switching systems. *Automatica*, 50(10): 2504–2514.

Prabhakar, P.; and Rahimi Afzal, Z. 2019. Abstraction based output range analysis for neural networks. *Advances in Neural Information Processing Systems*, 32.

Reis, C.; Barth, A.; and Pizano, C. 2009. Browser security: lessons from google chrome. *Communications of the ACM*, 52(8): 45–49.

Saha, I.; Ramaithitima, R.; Kumar, V.; Pappas, G. J.; and Seshia, S. A. 2014. Automated composition of motion primitives for multi-robot systems from safe LTL specifications. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1525–1532.

Sha, L. 2001. Using simplicity to control complexity. *IEEE Software*, 20–28.

Tran, H.-D.; Yang, X.; Manzanas Lopez, D.; Musau, P.; Nguyen, L. V.; Xiang, W.; Bak, S.; and Johnson, T. T. 2020. NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *International Conference on Computer Aided Verification*, 3–17. Springer.

van Huijgevoort, B. C.; and Haesaert, S. 2022. Similarity quantification for linear stochastic systems: A coupling compensator approach. *Automatica*, 144: 110476.

Wabersich, K. P.; and Zeilinger, M. N. 2018. Safe exploration of nonlinear dynamical systems: A predictive safety filter for reinforcement learning. *arXiv preprint arXiv:1812.05506*.

Wang, X.; Hovakimyan, N.; and Sha, L. 2013. L1Simplex Fault-Tolerant Control of Cyber-Physical Systems. In *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, 41–50.

Xiang, W.; Musau, P.; Wild, A. A.; Lopez, D. M.; Hamilton, N.; Yang, X.; Rosenfeld, J.; and Johnson, T. T. 2018. Verification for machine learning, autonomy, and neural networks survey. *arXiv preprint arXiv:1810.01989*.

Yu, P.; and Dimarogonas, D. V. 2022. Distributed Motion Coordination for Multirobot Systems Under LTL Specifications. *IEEE Transactions on Robotics*, 38(2): 1047–1062.

Zhong, B.; Lavaei, A.; Cao, H.; Zamani, M.; and Caccamo, M. 2021. Safe-visor architecture for sandboxing (AI-based) unverified controllers in stochastic cyber–physical systems. *Nonlinear Analysis: Hybrid Systems*, 43: 101110.

Zhong, B.; Lavaei, A.; Zamani, M.; and Caccamo, M. 2023. Automata-based controller synthesis for stochastic systems: A game framework via approximate probabilistic relations. *Automatica*, 147: 110696.

Zhong, B.; Zamani, M.; and Caccamo, M. 2019. Sandboxing Controllers for Stochastic Cyber-Physical Systems. In *International Conference on Formal Modeling and Analysis of Timed Systems, Lecture Notes in Computer Science*, 247–264. Springer.