Safe and Reliable Training of Learning-Based Aerospace Controllers

Udayan Mandal
Center for AI Safety
Stanford University
Stanford, USA
udayanm@stanford.edu

Guy Amir
School of CS & Engineering
The Hebrew University of Jerusalem
Jerusalem, Israel
guyam@cs.huji.ac.il

Haoze Wu
Center for AI Safety
Stanford University
Stanford, USA
haozewu@stanford.edu

Ieva Daukantas
Department of Computer Science
IT University of Copenhagen
Copenhagen, Denmark
daukantas@itu.dk

Fletcher Lee Newell
Center for AI Safety
Stanford University
Stanford, USA
flnewell@stanford.edu

Umberto Ravaioli
Google
Mountain View, USA
uravaioli@google.com

Baoluo Meng
GE Aerospace Research
Niskayuna, USA
baoluo.meng@ge.com

Michael Durling
GE Aerospace Research
Niskayuna, USA
durling@ge.com

Kerianne Hobbs
Air Force Research Laboratory
US Air Force
Dayton, USA
kerianne.hobbs@afrl.af.mil

Milan Ganai
Department of Computer Science
Stanford University
Stanford, USA
mganai@stanford.edu

Tobey Shim
Department of Data Science
Stanford University
Stanford, USA
tshim24@stanford.edu

Guy Katz
School of CS & Engineering
The Hebrew University of Jerusalem
Jerusalem, Israel
guykatz@cs.huji.ac.il

Clark Barrett
Center for AI Safety
Stanford University
Stanford, USA
barrett@stanford.edu

Abstract—In recent years, deep reinforcement learning (DRL) approaches have generated highly successful controllers for a myriad of complex domains. However, the opaque nature of these models limits their applicability in aerospace systems and sasfety-critical domains, in which a single mistake can have dire consequences. In this paper, we present novel advancements in both the training and verification of DRL controllers, which can help ensure their safe behavior. We showcase a design-for-verification approach utilizing k-induction and demonstrate its use in verifying liveness properties. In addition, we also give a brief overview of neural Lyapunov Barrier certificates and summarize their capabilities on a case study. Finally, we describe several other novel reachability-based approaches which, despite failing to provide guarantees of interest, could be effective for verification of other DRL systems, and could be of further interest to the community.

Index Terms—AI Safety, Deep Reinforcement Learning, Formal Verification, Deep Neural Network Verification

I. INTRODUCTION

Deep reinforcement learning (DRL) has gained significant popularity in recent years, reaching state-of-the-art performance in various domains. One such domain is aerospace systems, in which DRL models are under consideration for replacing years-old software by learning to efficiently control airborne platforms and spacecraft. However, although they perform well empirically, DRL systems have an opaque decision-making process, making them challenging to reason about. More importantly, this opacity raises critical questions about safety and security (e.g., *How can we ensure that the spacecraft will never violate a velocity constraint? Will it always reach its destination?*) which are difficult to answer. These reliability concerns are a significant obstacle to deploying DRL controllers in real-world systems, where even a single mistake cannot be tolerated.

To cope with this urgent need, a myriad of DRL *training techniques* have been put forth in recent years to enhance the performance of such systems. However, these current approaches suffer from two main drawbacks: (i) they are usually not geared towards improving safety and reliability (which is key in aerospace systems); and (ii) they are heuristic in nature and do not afford any formal guarantees. At the same time, the formal methods community has been developing methods for formally and rigorously assessing the reliability of DRL systems. However, although such methods are useful for identifying whether a system is safe, they are typically not incorporated into the DRL training process, but are rather used only afterwards.

In this work, we begin bridging this gap by proposing a novel design-for-verification approach that can be incorporated during the DRL training process. Our approach both modifies the training loop to be more verification-friendly and also utilizes formal verification (in our case, k-induction), to ensure the correctness of the training. We also report a summary of our recent efforts to use Neural Lyapunov Barrier certificates [26] to generate DRL agents that not only perform well on large batches of data, but also meet rigorous correctness criteria as measured by state-of-the-art verification tools.

Finally, we introduce additional novel reachability-based approaches for providing safety and liveness guarantees about a DRL system. These approaches are derived from prior work on backward-tube reachability, forward-tube reachability, and abstraction-based reachability methods. Moreover, these approaches all follow a similar paradigm: the reachable space covered by all possible paths from the starting state space is over-approximated using a verification engine, and safety and

liveness properties are checked over this over-approximated state space.

To demonstrate the usefulness of our approaches, we apply them to a benchmark satellite-control model developed in collaboration with industry partners (GE Aerospace Research and the U.S. Air Force). We demonstrate that liveness can be verified using our k-induction approach. Additionally, as a point of comparison, we showcase that the certificate-based approach is indeed able to generate a controller that provably behaves safely. Notably, the problem setting and controller complexity are beyond that acheived in previous work on formally verified controllers.

The other reachability-based methods fail on this benchmark. However, we believe that these failed attempts: (i) demonstrate the merits of our successful approaches in handling complex, nontrivial properties; (ii) can be of value to the community, by shedding light on vulnerabilities of alternate methods; and (iii) could be potentially successful when applied over different DRL systems.

We view this work as an important step towards the safe and reliable deployment of DRL controllers in real-world systems, especially in the complex domain of avionics. We additionally hope that our work will further motivate additional research in neural network verification, DRL safety, and specifically, their role in the important domain of DRL-controlled aerospace systems.

The rest of the paper is organized as follows. In Sec. II, we cover background on deep learning, DRL, and verification, and we also introduce Neural Lyapunov Barrier functions. In Sec. III, we introduce our benchmark problem, a 2D spacecraft docking challenge. We subsequently introduce our k-induction technique in Sec. IV, and we present alternative verification approaches in Sec. V. [1] Finally, we conclude in Sec. VI.

II. PRELIMINARIES AND RELATED WORK

A. Safety and Liveness

In this paper, we are interested in obtaining DRL controllers that satisfy safety and liveness properties [2] in discrete-time settings.

Safety. In a sequence satisfying a safety property, *a bad state is never reached*. For the set of system states $\mathcal{X}$, let $\tau \subseteq \mathcal{X}^*$ be the set of potential system trajectories. We say a trajectory α satisfies *safety* property P_1 if and only if each state in α satisfies property P_1. More formally:

$$\forall \alpha : \alpha \in \tau . \forall x \in \alpha . x \vDash P_1. \tag{1}$$

Finite-length trajectories terminating in a "bad" state (where P_1 does not hold) constitute the set of trajectories in violation of the safety property.

Liveness. On the other hand, a liveness property indicates *a good state is eventually reached*. A *liveness* property P_2 is satisfied by trajectory α if and only if there exists a state x in α where P_2 holds. Defining τ^∞ as the set of infinite-length trajectories, we formally specify liveness property P_2 as:

$$\forall \alpha : \alpha \in \tau^\infty . \exists x \in \alpha . x \vDash P_2. \tag{2}$$

[1] Code for these approaches is available at: github.com/NeuralNetworkVerification/artifact-dasc-docking

Infinite-length trajectories which contain no "good" states (i.e., no states where P_2 holds) constitute the set of trajectories in violation of the liveness property.

B. DNNs, DNN Verification, and Dynamical Systems.

Deep Learning. Deep neural networks (DNNs) consist of layers of neurons that perform some (usually nonlinear) transformation of the input [38]. In this paper, we investigate deep reinforcement learning (DRL), where we train a DNN to obtain a *policy*, which maps states to actions that control a system [54].

DNN Verification. Given (i) a trained DNN (e.g., a DRL agent) N; (ii) a pre-condition P on the DNN's inputs, limiting the input assignments; and (iii) a post-condition Q on the DNN's outputs, the goal of DNN verification is to determine whether the property $P(x) \rightarrow Q(N(x))$ holds for any neural network input x. In many DNN verifiers (a.k.a., *verification engines*), this task is equivalently reduced to determining the satisfiability of the formula $P(x) \wedge \neg Q(N(x))$. If the formula is satisfiable (SAT), then there is an input that satisfies the pre-condition and violates the post-condition, which means the property is violated. On the other hand, if the formula is unsatisfiable (UNSAT), then the property holds. It has been shown [49] that verification of piece-wise-linear DNNs is NP-complete. In recent years, the formal methods community has put forth various techniques for verifying and improving DNN reliability [1], [5], [6], [9], [13], [17], [23], [70]. These techniques include SMT-based methods [8], [45], [50], [52], optimization-based methods [15], [30], [55], [68], methods based on abstraction-refinement [10], [22], [31], [32], [58], [59], [65], methods based on shielding [24], [51], [63], and more.

Discrete-Time Dynamical Systems. We consider discrete-time dynamical systems, particularly systems whose trajectories satisfy the equation:

$$x_{t+1} = f(x_t, u_t), \tag{3}$$

in which the *transition function* f takes as inputs the current state $x_t \in \mathcal{X}$ and a control $u_t \in \mathcal{U}$ and produces as output the subsequent state x_{t+1}. To control these systems, we employ a policy $\pi : \mathcal{X} \rightarrow \mathcal{U}$ that takes in a state $x \in \mathcal{X}$ and outputs a control action $u = \pi(x)$. In DRL, the controller π is realized by a trained DNN agent. These learning-based controllers have proven to be effective in many real-world settings including robotics [26], biomedical systems [28], and energy management [44], due to their expressive power and ability to generalize to unseen, complex environments [67].

C. Control Lyapunov Barrier Functions

The problem of verifying safety and liveness properties in a dynamical system can be solved by finding a function $V : \mathcal{X} \mapsto \mathbb{R}$ with certain properties. Control theory identifies two fundamental types of functions [53].

Lyapunov Functions. Lyapunov functions, a.k.a., *Control Lyapunov functions*, capture the energy level at a particular state: over time, energy is dissipated along a trajectory until the system attains zero-energy equilibrium [41]. Lyapunov functions can guarantee *asymptotic stability*, which ensures the system eventually converges to some goal state (thereby

satisfying a liveness property). Lyapunov functions must be (*i*) equal to 0 at equilibrium, (*ii*) strictly positive at all other states; and (*iii*) monotonically decreasing [18], [19], [36].

Barrier Functions. Barrier functions [4], a.k.a., *Control Barrier Functions*, guarantee that a system never enters an unsafe region (i.e., a "bad" state) in the state space. This is achieved by setting the function value to be above some threshold for unsafe states and then verifying that the system can never transition to a state where the function is above the threshold [3], [12], [72]. Previous work [60], [61], [69], [75] demonstrates how to obtain Barrier functions for various safety-critical tasks such as pedestrian avoidance, neural radiance field-based obstacle navigation [57], and multi-agent control.

Control Lyapunov Barrier Functions. Often, it is necessary to ensure both safety and liveness properties simultaneously. In such cases, we can employ a *Control Lyapunov Barrier Function* (CLBF), which integrates the properties and guarantees of both Control Lyapunov functions and Control Barrier functions [27]. CLBFs can solve reach-while-avoid tasks [29], which we discuss next.

Reach-while-Avoid Tasks. The goal of *Reach-while-Avoid* (RWA) tasks is to find a controller π for a dynamical system such that every trajectory $\{x_1, x_2...\}$ produced under this controller (i) never enters an unsafe ("bad") state; and (ii) eventually enters a goal ("good") region or state. We can formally define the problem as:

Definition 1 (Reach-while-Avoid Task)**.**
Input: *A dynamical system with a set of initial states $\mathcal{X}_I \subseteq \mathcal{X}$, a set of goal states $\mathcal{X}_G \subseteq \mathcal{X}$, and a set of unsafe states $\mathcal{X}_U \subseteq \mathcal{X}$, where $\mathcal{X}_I \cap \mathcal{X}_U = \varnothing$ and $\mathcal{X}_G \cap \mathcal{X}_U = \varnothing$*
Output: *A controller π such that for every trajectory $\tau = \{x_1, x_2...\}$ satisfying $x_1 \in \mathcal{X}_I$:*
  1) Reach: $\exists\, t \in \mathbb{N}.\ x_t \in \mathcal{X}_G$
  2) Avoid: $\forall\, t \in \mathbb{N}.\ x_t \notin \mathcal{X}_U$

Some solutions for RWA tasks rely on control theoretic principles. The approach in [27] trains Lyapunov and Barrier certificates to solve RWA tasks. Hamilton-Jacobi (HJ) reachability-based methods [11]) have also been employed to solve RWA tasks [34], [43], [66]. Safe DRL is closely connected to RWA, with its goal being to maximize cumulative rewards while minimizing costs along a trajectory [14]. It has been solved with both Lyapunov/Barrier methods [20], [73] and HJ reachability methods [35], [74].

D. Other Verification Approaches

Reachability Analysis. Reachability analysis methods aim to define and compute the set of final reachable states and then verify that this set (i) does not include any bad states, and (ii) is contained within the goal region. Reachability methods include forward-tube and backward-tube verification [40], which either propagate states forward from the starting set or backward from the goal set. Other related work in reachability analysis includes hybrid system verifiers [46], growing the set of reachable states over a discrete action space [48], approximating reachable states during forward and backward reachability [39], and

reformulating the dynamics of a system for easier reachability verification [37].

Bounded Model Checking and k-induction. Bounded model checking uses a symbolic analysis over k copies of a system to check whether a bad state is reachable in k or fewer steps from the starting set of states. k-induction is similar, except that it starts from an arbitrary state and can thus be used to prove that a bad state is never reached. Bounded model checking has been explored in the WhiRL tool [33] using the neural network verifier Marabou [50], [71]. [64] implements another tool for checking adversarial cases and coverage using bounded model checking for artificial neural networks. WhiRL 2.0 [7] adds k-induction capabilities to WhiRL.

Design-for-Verification. Design-for-verification broadly encompasses any method which aims to modify the design and training process to make verification easier. The Trainify framework [47] uses a CEGAR-based approach to grow an easily verifiable state space by repeatedly retraining the DRL system. [25] motivates an optimized DRL training approach to reduce the number of safety violations, easing formal verification. This approach was also implemented in Marabou [50], [71].

III. 2D DOCKING PROBLEM

We adopt as a motivating case study benchmark the 2D docking problem presented in [62]. The goal is to train a DRL controller to safely navigate a deputy spacecraft to a chief spacecraft within two-dimensional space. The reference frame is defined such that the chief spacecraft is always at the origin $(0, 0)$. The state of the deputy spacecraft is $\boldsymbol{x} = [x, y, \dot{x}, \dot{y}]$, where (x, y) are the position of the spacecraft and $(\dot{x}, \dot{y})$ are the respective directional velocities.

A. Dynamics

The system dynamics are defined according to the linearly-approximate Clohessy-Wiltshire relative orbital motion equations in a non-inertial Hill's reference frame [21], [42]. The control input to the system is $\boldsymbol{u} = [F_x, F_y]$, where F_x and F_y are the thrust forces applied to the deputy spacecraft in the x and y directions. We follow [62], setting the deputy spacecraft mass to $m = 12$ kg and the mean motion to $n = 0.001027$ rad/s. The continuous time state dynamics of the system are given by the following differential equations:

$$\dot{\boldsymbol{x}} = [\dot{x}, \dot{y}, \ddot{x}, \ddot{y}] \tag{4}$$

$$\ddot{x} = 2n\dot{y} + 3n^2 x + \frac{F_x}{m} \tag{5}$$

$$\ddot{y} = -2n\dot{x} + \frac{F_y}{m} \tag{6}$$

Integration using a discrete time step T yields a closed-form next-state function. Given a state $\boldsymbol{x} = [x, y, \dot{x}, \dot{y}]$ and control inputs $\boldsymbol{u} = [F_x, F_y]$, the spacecraft's next state $\boldsymbol{x}_i' = [x', y', \dot{x}', \dot{y}']$ after an elapsed time T is:

$$x' = \left(\frac{2\dot{y}}{n} + 4x + \frac{F_x}{mn^2}\right) + \left(\frac{2F_y}{mn}\right) + \left(-\frac{F_x}{mn^2} - \frac{2\dot{y}}{n} - 3x\right)$$
$$\cdot \cos{(nT)} + \left(\frac{-2F_y}{mn^2} + \frac{\dot{x}}{n}\right)\sin{(nT)} \quad (7)$$

$$y' = \left(-\frac{2\dot{x}}{n} + y + \frac{4F_y}{mn^2}\right) + \left(\frac{-2F_x}{mn} - 3\dot{y} - 6nx\right)T + -\frac{3F_y}{2m}t^2$$
$$+ \left(-\frac{4F_y}{mn^2} + \frac{2\dot{x}}{n}\right)\cos{(nT)} + \left(\frac{2F_x}{mn^2} + \frac{4\dot{y}}{n} + 6x\right)$$
$$\cdot \sin{(nT)} \quad (8)$$

$$\dot{x}' = \left(\frac{2F_x}{mn}\right) + \left(\frac{-2F_y}{mn} + x\right)\cos{(nT)} + \left(\frac{F_x}{mn} + 2\dot{y}\right.$$
$$\left. + 3nx\right)\sin{(nT)} \quad (9)$$

$$\dot{y}' = \left(\frac{-2F_x}{mn} - 3\dot{y} - 6nx\right) + \left(-\frac{3F_y}{m}\right)T + \left(\frac{2F_x}{mn} + 4\dot{y}\right.$$
$$\left. + 6nx\right)\cos{(nT)} + \left(\frac{4F_y}{mn} - 2\dot{x}\right)\sin{(nT)} \quad (10)$$

B. Liveness —— Docking Region

The problem as given in [62] defines a *docking region* which is a circle of radius 0.5 meters centered at the origin. The goal is for the deputy spacecraft to eventually enter this region. To simplify the verification query, it is easier to use linear bounds for the goal region, so we use a square centered at the origin with sides parallel to the axes of length 0.7 meters (note that this square fits inside the docking region of [62]). Formally, our liveness condition is:

$$\forall \alpha : \alpha \in \tau^\infty . \ \exists t. \ |\alpha_t.x| \leq 0.35 \wedge |\alpha_t.y| \leq 0.35, \quad (11)$$

where α_t is the state at time t in trajectory α, and $\alpha_t.x$ and $\alpha_t.y$ are the x and y components of α_t.

C. Safety — Velocity Threshold

To minimize the risk to both spacecraft, a safety constraint is imposed on the magnitude of the velocity of the deputy spacecraft. The constraint depends on the distance from the deputy. Formally, [62] requires the following state invariant:

$$\sqrt{\dot{x}^2 + \dot{y}^2} \leq 0.2 + 2n\sqrt{x^2 + y^2} \quad (12)$$

We therefore define the unsafe region to be the negation of (12).

Again, we desire to instead use a linear constraint in order to be compatible with our formal tools. We use the Euclidean norm approximation of [16], which approximates the norm by projecting it onto vectors in all different directions and taking the one with the maximum magnitude. We use the two inequalities:

$$\max_{i \in [1, n_{directions}]}\left(u_1 \cdot \cos\left(\frac{2(i-1)\pi}{n_{directions}}\right) + u_2\right.$$
$$\left. \cdot \sin\left(\frac{2(i-1)\pi}{n_{directions}}\right)\right) \leq \sqrt{u_1^2 + u_2^2} \quad (13)$$

and

$$\frac{1}{\cos(\pi/n_{directions})} \max_{i \in [1, n_{directions}]}\left(u_1 \cdot \cos\left(\frac{2(i-1)\pi}{n_{directions}}\right)\right.$$
$$\left. + u_2 \cdot \sin\left(\frac{2(i-1)\pi}{n_{directions}}\right)\right) \geq \sqrt{u_1^2 + u_2^2}, \quad (14)$$

where $n_{directions}$ is a positive integer. Larger values of $n_{directions}$ yield more precise approximations. We can simplify this by noting that:

$$\sqrt{u_1^2 + u_2^2} = \sqrt{|u_1|^2 + |u_2|^2},$$

and then focusing our search only on vectors in the first quadrant. Assuming $n_{directions}$ is a multiple of 4, we get:

$$\text{under}(u_1, u_2) = \max_{i \in [1, n_{directions}/4+1]}\left(|u_1| \cdot \cos\left(\frac{2(i-1)\pi}{n_{directions}}\right)\right.$$
$$\left. + |u_2| \cdot \sin\left(\frac{2(i-1)\pi}{n_{directions}}\right)\right) \quad (15)$$
$$\leq \sqrt{u_1^2 + u_2^2}$$

and

$$\text{over}(u_1, u_2) = \frac{1}{\cos(\pi/n_{directions})} \max_{i \in [1, n_{directions}/4+1]}\left(|u_1|\right.$$
$$\left. \cdot \cos\left(\frac{2(i-1)\pi}{n_{directions}}\right) + |u_2| \cdot \sin\left(\frac{2(i-1)\pi}{n_{directions}}\right)\right)$$
$$\geq \sqrt{u_1^2 + u_2^2}. \quad (16)$$

Using these constraints, we can over-approximate the unsafe region as

$$\text{over}(\dot{x}_t, \dot{y}_t) > 0.2 + 2n \cdot \text{under}(x_t, y_t). \quad (17)$$

This is a piece-wise linear constraint. Moreover, both the absolute value function and the maximum function can be easily encoded in neural network verification tools such as Marabou. In our experiments, we use $n_{directions} = 400$.

D. DNN Setup

As in [62], we use Ray RLib's Proximal Policy Optimization (PPO) reinforcement learning algorithm to learn the system dynamics, but we make four important alterations to improve downstream verification, part of our *design for verification* scheme.

1) Scenario Regions: To improve performance near the docking region, we reduce the docking distance during training from 0.5 meters to 0.25 meters. We also simplify the problem by reducing the initial position of the deputy spacecraft from a radius of 150 meters to only 5 meters. Scaling back up to larger initial positions is part of an ongoing research effort.

2) Speed Observations: We limit the observations of the agent to its x and y positions and respective $\dot{x}$ and $\dot{y}$ velocities, eliminating the agent's observations of its current speed and the distance-dependent velocity constraint described in Equation 12. This makes it less likely that irregular trajectories will be learned because of observations of the safety constraint. As a result, liveness verification becomes easier.

3) Distance Reward: We keep the rewards relating to success or failure, the safety constraint, and delta-v as presented in [62], but we alter the distance change reward to use the L^1 norm of the position of the deputy — i.e., the Manhattan distance from the deputy to the chief, rather than the nonlinear L^2 norm. This is to match the induction invariant described in Section IV. To account for the new distance metric and previously-described

smaller initial distances, we developed a novel reward function for distance change:

$$R_t^{d_{new}} = 2\left(e^{-a_1 d_t^m} - e^{-a_1 d_{t-1}^m}\right) + 2\left(e^{-a_2 d_t^m} - e^{-a_2 d_{t-1}^m}\right), \quad (18)$$

where $d_i^m = |x_i| + |x_i|$, $a_1 = \frac{\ln(2)}{5}$, and $a_2 = \frac{\ln(2)}{0.5}$.

4) Model Architecture: Our DRL controller should be sufficiently small to keep verification time reasonable and sufficiently large to be able to learn the necessary behavior. We found that reducing the hidden layer widths from 256 neurons to 20 neurons, while maintaining two hidden layers, acheives a good balance between verification time and expressive power. Also, we swap the tanh activation functions for ReLU activation functions since ReLU is supported by most neural network verification tools (such as Marabou).

IV. USING k-INDUCTION FOR LIVENESS GUARANTEES

In this section, we present an approach for scalably verifying a liveness property for the 2D docking problem presented in Section III using k-induction. We describe the conceptual approach, the experimental framework, and the results.

A. Proving Liveness by k-induction

In order to apply k-induction, we must find a way to reduce a liveness property to a k-inductive property. Typically, this is done by finding a *ranking function*, a function with a well-founded co-domain, which can be shown to always be decreasing by k-induction.

For the spacecraft, an obvious choice for a ranking function is the distance from the deputy to the chief. In order to make the function easier to reason about, we use a linear proxy function for the actual distance, namely the Manhattan distance. Unfortunately, it is not the case that this measure always decreases, as the spacecraft may move away from the target.

Thus, we instead propose a property that ensures the spacecraft eventually starts moving towards the target. The property is expressed as a logical disjunction: after k steps, either the Manhattan distance decreases or the magnitude of the velocity decreases. Again, we approximate the velocity magnitude by the L^1 norm, the sum of the absolute values of $\dot{x}$ and $\dot{y}$. Formally, if the current state is $(x_0, y_0, \dot{x}_0, \dot{y}_0)$ and the future state after k steps is $(x', y', \dot{x}', \dot{y}')$, we must show:

$$(|x'|+|y'|)-(|x_0|+|y_0|) < -\epsilon \quad \bigvee \quad (|\dot{x}'|+|\dot{y}'|)-(|\dot{x}_0|+|\dot{y}_0|) < -\epsilon, \quad (19)$$

where ϵ is some positive value.

Proposition 1. *If property* (19) *holds (for some k) for every state, then eventually the spacecraft will be moving towards the goal (i.e., the L^1 norm of the position will decrease).*

Proof. Suppose that from some starting state, $(x_0, y_0, \dot{x}_0, \dot{y}_0)$, the spacecraft follows a trajectory that never moves towards the goal in the sense that the L^1 norm never decreases. Let $(x_i, y_i, \dot{x}_i, \dot{y}_i)$ be the state after i time steps. This means that for all i, $|x_i| + |y_i| \le |x_{i+1}| + |y_{i+1}|$. Let $V_i = |\dot{x}_i| + |\dot{y}_i|$. By (19), we know that for each V_i, there must be some k, such that $V_{i+k} - V_i < -\epsilon$. Thus, for any n, we can construct a sequence $V_{j_0}, V_{j_1}, V_{j_2}, \ldots V_{j_n}$ such that $j_0 = 0$ and $V_{j_i} - V_{j_{i+1}} > \epsilon$. If we

then take $n > V_0/\epsilon$, we get that $V_{j_n} < 0$, which is impossible. $\square$

Algorithm. We verify (19) using Algorithm 1. We gradually increase k until the property holds, a maximum of $k = k_{max}$ is reached, or a timeout is exceeded.

Algorithm 1: Algorithm for k-induction.

Require: Bounds on state components $x_0, y_0, \dot{x}_0, \dot{y}_0$, values for k_{min}, k_{max}
Ensure: If result = UNSAT, then property (19) holds for all states within the defined bounds.
1: **for** each $k \in [k_{min}, k_{max}]$ **do**
2:     Verify the negation of the distilled property:
$$\neg \begin{pmatrix} (|x'| + |y'|)-(|x_0| + |y_0|) < -\epsilon \\ \bigvee \\ (|\dot{x}'| + |\dot{y}'|)-(|\dot{x}_0| + |\dot{y}_0|) < -\epsilon \end{pmatrix}$$
3:     **if** UNSAT **then**
4:         result = [UNSAT, k]
5:         *break*;
6:     **else**
7:         result = [SAT, k, counterexample k-step trajectory].
8:     **end if**
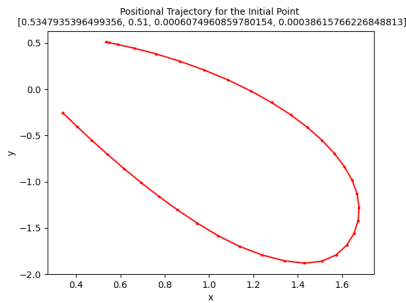9: **end for**
10: **return** result

Input bounds for the state space can be chosen according to the problem specification. It is also important to note that different k_{min} and k_{max} values can be chosen. In practice, in order to make the verification more tractable, we first split the state space into subregions, then call the algorithm on each subregion. For each subregion of the state space, we explore values of k from k_{min} to k_{max}. For each k, a neural network verifier is invoked to check if the negation of the property holds after k steps. There are three possible results of the algorithm.

1) If the negation of the property is satisfiable for each k, the algorithm returns *SAT* along with a counter-example.
2) If the negation of the property is unsatisfiable for some k, this means that the property holds for that value of k. In this case, the algorithm returns *UNSAT* together with the value of k for which unsatisfiability was determined. In this case, verification of the region is complete.
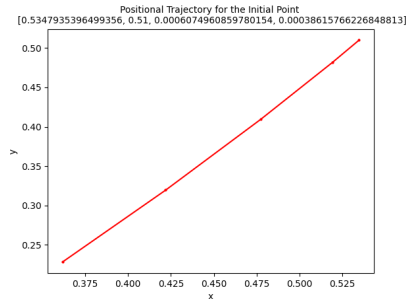3) If a predefined timeout is exceeded, the algorithm terminates and a timeout result is returned.

Experimental Setup. We use Marabou for the neural network verification step. We set the following parameters for Marabou: *"verbosity=0, timeoutInSeconds=5000, numWorkers=10, tighteningStrategy="sbt", solveWithMILP=True"*. Marabou also requires a back-end linear programming engine. We use Gurobi 9.5.

We start with positional bounds of $|x|, |y| \in [-25, 25]$ and velocity bounds of $\dot{x}, \dot{y} \in [-0.2, 0.2]$). We initially divide these into 25 subregions by focusing on 5×5 regions in the positional space. A subregion is further subdivided if Algorithm 1 times out. We set k_{min} to 1, k_{max} to 20, and use a timeout of 1.4 hours for each loop iteration (i.e., 30 hours if all values of k time out).

Results. We end up with 71 subregions. For each subregion, Algorithm 1 returns UNSAT. The minimum returned value for

Positional Trajectory for the Initial Point
[0.5347935396499356, 0.51, 0.0006074960859780154, 0.00038615766226848813]

(a) Initial neural network.

Positional Trajectory for the Initial Point
[0.5347935396499356, 0.51, 0.0006074960859780154, 0.00038615766226848813]

(b) Retrained neural network.

Fig. 1: Design for Verification: An initial controller trajectory compared to a final controller trajectory, with the same initial state. The final controller has a more direct trajectory which is more conducive to verification via k-induction.

k is 1, the maximum is 12, the average is 5, and the median is 3.

Notably, regions close to the goal region are more difficult: they require more subregions and take longer, whereas regions more distant can sometimes be verified without utilizing additional subregions. The minimum runtime (in seconds) for any subregion is 0.02, the maximum is 4295.86, the average is 193.62, and the median is 1.76.

As a sanity check, we validated our results experimentally by running a simulation framework. Starting from randomly sampled points in the state space, we confirmed that the k-inductive property holds on the trajectory starting at each point. These checks also succeeded.

Discussion. Initially, we applied our approach to the neural network controller described in [62]. The original network topology (two hidden layers with 256 nodes each) resulted in lengthy verification times. Moreover, for many regions, the verification failed: we discovered counter-examples for all tested values of k.

Figure 1a shows an example counterexample trajectory from the original neural network. The starting state is $[x = 0.5347935396499356, y = 0.51, \dot{x} = 0.0006074960859780154, \dot{y} = 0.00038615766226848813]$. The controller moves steadily away from the goal, and only after many steps turns the spacecraft around to move towards the goal.

Such trajectories provided motivation for the design changes mentioned in Section III-D. In particular, the changes to the reward function strongly incentivize the controller to move towards the goal region. Figure 1b shows the trajectory using

the verified controller, starting from the same starting state. Note how the spacecraft moves nearly directly towards the goal region.

The successful verification of (19) is not sufficient to establish that the deputy eventually reaches the chief. We would need to establish a second property, namely that once the spacecraft is moving towards its goal, it always gets closer (by at least some ϵ) within k steps. Let x_i, y_i be the position i steps from some starting position (x_0, y_0). This can be formalized with the property:

$$
\begin{aligned}
(|x_1| + |y_1|) - (|x_0| + |y_0|) &< 0 \implies \\
\exists k. \ (|x_k| + |y_k|) - (|x_0| + |y_0|) &< -\epsilon.
\end{aligned}
\tag{20}
$$

Formally verifying this property is left to future work.

B. An Alternative Approach using Polar Coordinates

Before moving to the Manhattan distance, we explored an alternative approach using polar coordinates, which allows the L^2 norm to be used directly in the invariant while maintaining linearity. More specifically, if r is the distance to the origin and θ is the angle from the x-axis, then we can write the equivalent of property (19) as:

$$
r' - r < -\epsilon \lor \dot{r}' - \dot{r} < -\epsilon.
\tag{21}
$$

Note how much simpler property 21 is compared with property (19). However, there remain two challenges: training a polar controller and converting the dynamics to polar coordinates.

Training a controller for the polar system is not straightforward; it requires complex parameter changes, for example, adjusting the learning rate, observation vector order, and the length and normalization constants. However, these challenges are ultimately solvable, and we were able to train a network that takes polar coordinate inputs. The output is still F_x and F_y, as we did not envision changing the physical spacecraft system.

The second challenge proved more difficult. We needed a way to calculate new values of r and θ, given current values of r, θ, $\dot{r}$, and $\dot{\theta}$, as well as F_x and F_y. We did not find closed-form solutions in the literature for the Clohessy–Wiltshire Equations utilizing polar coordinates. We thus converted equations (7) through (10) to polar coordinates using the standard conversion equations:

$$
x = r\cos\theta, \quad y = r\sin\theta, \quad r = \sqrt{x^2 + y^2}, \quad \theta = \tan^{-1}\frac{y}{x} \tag{22}
$$

We encoded the derivation of the equations directly in *Python*, which allowed us to confirm in simulation that our polar neural network had behavior similar to that of the original model. However, attempting formal verification with the new dynamics proved difficult. The new dynamics are highly non-linear. We attempted to use the *OVERT* tool[2] for the purpose of linearizing r and θ. However, the results were too complex and ultimately unsuccessful. It was at this point that we decided to instead use the L^1 norm and revert to standard rectangular coordinates.

We report this effort here in order to highlight both the potential benefits and pitfalls of using a different coordinate

[2]https://github.com/sisl/OVERT.jl

representation. If the dynamics had been more tractable in polar space, this would have been an attractive direction.

V. Alternate Verification Approaches

While exploring the k-induction approaches described above, we concurrently explored an alternative approach using Neural Lyapunov Barrier certificates. The results of that effort represent the most complete verification results we have obtained to date and are reported in [56]. Here, for convenience, we review that approach at a high level and present some details not reported there. We also discuss several reachability-based approaches, which we also applied to the 2D docking problem, but which were, ultimately, unsuccessful.

A. RWA Certificates

Definition 2. *A function $V : \mathcal{X} \mapsto \mathbb{R}$ is an RWA certificate for the task defined in Definition 1 if there exist some $\alpha > \beta \geq \gamma$ and $\epsilon > 0$, such that the following constraints are satisfied.*

$$\forall\, x \in \mathcal{X}. \qquad V(x) \geq \gamma \qquad\qquad (23)$$
$$\forall\, x \in \mathcal{X}_I. \qquad V(x) \leq \beta \qquad\qquad (24)$$
$$\forall\, x \in \mathcal{X} \smallsetminus \mathcal{X}_G. \quad V(x) \leq \beta \rightarrow V(x) - V(f(x, \pi(x))) \geq \epsilon \qquad (25)$$
$$\forall\, x \in \mathcal{X}_U. \qquad V(x) \geq \alpha \qquad\qquad (26)$$

Any tuple of values $(\alpha, \beta, \epsilon, \gamma)$ for which these conditions hold is called a *witness for* the certificate.[3] RWA certificates provide the following guarantee.

Lemma 1. *If V is an RWA certificate for a dynamical system with witness $(\alpha, \beta, \epsilon, \gamma)$, then for every trajectory τ starting from a state $x \in \mathcal{X} \smallsetminus \mathcal{X}_G$ such that $V(x) \leq \beta$, τ will eventually contain a state in $\mathcal{X}_G$ without ever passing through a state in $\mathcal{X}_U$.*

We use reinforcement learning to jointly train neural networks for both the controller and the corresponding RWA certificate.

RWA Training Loss. The training objective for RWA certificates is described below:

$$O_s = c_s \sum_{i \,|\, x_i \in \mathcal{X}_I} \frac{\text{ReLU}(\delta_1 + V(x_i) - \beta)}{\sum_{i \,|\, x_i \in \mathcal{X}_I} 1} \qquad (27)$$

$$O_d = c_d \sum_{i \,|\, x_i \in \mathcal{X} \smallsetminus (\mathcal{X}_U \cup \mathcal{X}_G), V(x_i) < \beta} \frac{\text{ReLU}(\delta_2 + \epsilon + V(x_i') - V(x_i))}{\sum_{i \,|\, x_i \in \mathcal{X} \smallsetminus (\mathcal{X}_U \cup \mathcal{X}_G), V(x_i) < \beta} 1} \qquad (28)$$

$$O_u = c_u \sum_{i \,|\, x_i \in \mathcal{X}_U} \frac{\text{ReLU}(\delta_3 - V(x_i) + \alpha)}{\sum_{i \,|\, x_i \in \mathcal{X}_U} 1} \qquad (29)$$

$$O = O_s + O_d + O_u \qquad (30)$$

Equation (27) penalizes deviations from constraint (24), Equation (28) penalizes deviations from constraint (25), and Equation (29) penalizes deviations from constraint (26). We incorporate parameters $\delta_1 > 0$, $\delta_2 > 0$, and $\delta_3 > 0$, which can

---

[3]These constraints are similar to those in [29] but are specific to discrete-time systems and do not place constraints on a compact safe set, opting to use an unsafe set instead.

be used to tune how strongly the certificate over-approximates adherence to each constraint. Similarly, constants c_s, c_d, c_u can be used to tune the relative weight of the two objectives. The final training objective O in (30) is what the optimizer seeks to minimize, by using stochastic gradient descent (SGD) or other optimization techniques.

γ **lower bound.** It is important to note that the RWA training objective does not explicitly penalize deviations from Equation (23). Instead, because V is implemented as a neural network using floating-point arithmetic, it has only a finite number of possible inputs and outputs, so Equation (23) must hold for some γ. In practice, we can use Marabou to find γ by doing a linear search for the minimum value of V: we simply set γ to some initial value, say α, then repeatedly check $\exists\, x.\, V(x) < \gamma$, updating γ with the new value each time the query is satisfiable, and repeat until the query is unsastisfiable.

Sampling from $\mathcal{X}_U$ and $\mathcal{X} \smallsetminus \mathcal{X}_G$. While $\mathcal{X}_I$ is typically defined as having both upper and lower bounds on state variables, this is not the case for $\mathcal{X}_U$, which often has only lower bounds on state variables (this is the case, for example, for the 2D docking problem defined in Section III).

However, during training, we do impose an upper bound on the states sampled from $\mathcal{X}_U$. Specifically, if the controller operates over n-dimensional states $x = [x_1, x_2, .., x_n]$, we sample points satisfying the following constraints:

$$(x_1 > p_1) \vee (x_2 > p_2) \vee ... \vee (x_n > p_n) \quad (31)$$
$$(x_1 < p_1 + \gamma_1) \wedge (x_2 < p_2 + \gamma_2) \wedge ... \wedge (x_n < p_n + \gamma_n) \quad (32)$$

Here, 31 represents the (given) lower bounds on the unsafe region $\mathcal{X}_U$, and $\gamma_1, ..., \gamma_n$ are chosen to be strictly greater than 0.

A similar issue arises when sampling from $\mathcal{X} \smallsetminus \mathcal{X}_G$. This can often be solved simply by sampling instead from $\mathcal{X} \smallsetminus (\mathcal{X}_G \cup \mathcal{X}_U)$, as the lower bounds on variables in $\mathcal{X}_U$ then create upper bounds for the sampling step.

Masking out $\mathcal{X}_U$. For objective 28, if x_i' lies in $\mathcal{X}_U$, we replace the actual value of $V(x_i')$ with α. This is because we learn correct functional behaviors of $\mathcal{X}_U$ through objective 29 regardless, and thus using the actual value of $V(x_i')$ would lead to unnecessary training effort and excessive penalties.

Certificate Warmup. To improve training, the objective is used to train the certificate V alone for a few iterations, after which training includes both the certificate and the controller. This is done to avoid erratic training of the controller when V has random weights.

RWA Verification. In order to obtain formal guarantees, we use Marabou to formally verify the constraints in Definition 2. Verification of RWA constraints is generally straightforward, but we have to similarly bound $\mathcal{X}_U$ and $\mathcal{X} \smallsetminus \mathcal{X}_G$ to verify constraints 26 and 25 respectively. Instead of using $\mathcal{X} \smallsetminus \mathcal{X}_G$ as the input space for 25, we use instead $\mathcal{X} \smallsetminus (\mathcal{X}_G \cup \mathcal{X}_U)$, which provides the same guarantees. Moreover, instead of using $\mathcal{X}_U$ as the input space for 26, we use the bounded space, call it $\mathcal{X}_U^S$, used for data sampling. To ensure this provides the same guarantees, we check that no states beyond the upper bound of $\mathcal{X}_U^S$ are reachable.

Instead of encoding verification as a single property passed to the DNN verifier, verification is partitioned into muliple queries. This is done by paritioning the input space in the original property into equally sized smaller state spaces, over which the same property is checked. This helps avoid unreasonably long verification times that can occur with a large monolithic query.

Retraining. If any of the RWA verification checks return counterexamples, these are used to augment the training data set, and then training is done again. This process repeats until no more counterexamples are found. We weight counterexamples more heavily in the objective function 30 (compared to points in the initial training dataset) in order to focus the training on removing the counterexamples.

Results and Analysis. As shown in prior work in [56], RWA certificates can provide liveness and safety guarantees for the 2D spacecraft docking problem defined in Section III. More details and a pointer to the code can be found in [56].

B. Reachability Analysis Approaches

In this subsection, we discuss approaches based on *reachability analysis*. While these approaches were ultimately unsuccessful on the case study problem outlined in section III, we still mention them here, as the reasons for their failure may be of interest, and they may be useful on other problems.

Forward-tube and Backward-tube Reachability. Forward-tube and backward-tube reachability attempt to generate a path over abstract state spaces (i.e., sets of states) from the starting state space to the goal state space. At each step along the abstract path, we check that every state in the abstract state set meets any safety guarantees.

In forward-tube reachability, a starting set of states $\mathcal{X}_F^0$ and step size k is defined. Then, a set of states $\mathcal{X}_F^1$ is constructed such that all states reachable from $\mathcal{X}_F^0$ in k steps are contained within $\mathcal{X}_F^1$. This process is continued, and additional sets of states $\mathcal{X}_F^{i+1}$ are constructed, each with the property that they contain the states reachable from $\mathcal{X}_F^i$ in k steps. If at some point, the constructed set is a subset of the goal region, then the liveness property is ensured. However, it can be very challenging to find a sequence of sets of states $\mathcal{X}_F^i$ that eventually lead to a subset of the goal region. This was the case for the spacecraft example.

On the other hand, in backward-tube reachability, we start with $\mathcal{X}_B^0$ set equal to the goal states and define a step size k. Then, a set of states $\mathcal{X}_B^1$ is constructed such that all states reachable from $\mathcal{X}_B^1$ in k steps are contained within $\mathcal{X}_B^0$. Again, this process can be repeated until the set of states includes the initial states. A difficulty with this approach is computing a sufficiently large previous set of states at each step.

Grid Reachability. Grid reachability is a process which first partitions a bounded subset of the state space into cells, then computes a directed graph where each cell is a vertex, and each directed edge (a, b) denotes that vertex b is reachable from vertex a in k steps, for a specific k, as shown in Fig. 2. The goal is to show that for all paths constructed from cells in the defined initial state space, a goal region reachable. However, to ensure liveness, it is also necessary to show that the graph has
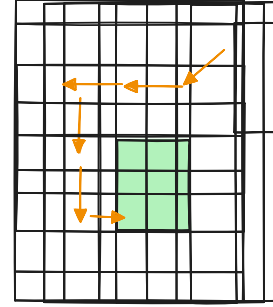

Fig. 2: Grid reachability, with a cell navigating towards the docking region (in green)

Algorithm 2: APPLYING GRID REACHABILITY

1 Let IS be the input space
2 Let k be the step size
3 Divide IS into cells $C = c_0, c_1, ..., c_n$
4 Let vertices $V = C$
5 Initialize edge set E to be the empty set
6 $i = 0$
7 **for** $i \leq n$ **do**
8   Denote set of adjacent cells to c_i as C_r
9   Add c_i to C_r if self-cycles are possible
10   **for** $c_r \in C_r$ **do**
11   **if** *c_r is reachable from c_i in k steps* **then**
12   Add directed edge (c_i, c_r) to E
13 $i = i + 1$
14 Let $G := (V, E)$
15 Check for cycles in G
16 **if** *G is acyclic* **then**
17   Determine cells C_s with no paths leaving input space
18   **return** C_s as cells meeting liveness property

---

no cycles and that it is not possible to reach any cells beyond the partitioned state space.

We applied this technique to the spacecraft example. A challenge is preventing self-cycles in the graph. One strategy for doing this is to construct cells where at least one velocity component never changes sign. It is easy to see that for such cells, the spacecraft cannot remain in the cell forever, so we can ignore self-loops on such cells. For cells containing a velocity sign-change, we use a very narrow velocity range, narrow enough to ensure that the spacecraft leaves the range in k steps. It is also desirable to limit the number of cells reachable from a given cell, to avoid the need to do many reachability checks. This can be ensured by making the cells large enough that it is impossible to cross more than one cell in a single set of k steps.

Analysis of Grid Reachability. We applied grid reachability to a state space with $x, y \in [-10, 10]$ and $\dot{x}, \dot{y} \in [-1.6, 1.6]$ using Algorithm 2. A binary search was conducted using Marabou to determine cell bounds such that cells could only reach adjacent cells. The step size k was chosen to be 1.

We found a variety of cycles of increasing lengths, even as cells were divided further in an attempt to refine the grid

x	y	v_x	v_y
1. (3.90625, 5.46875)	(3.90625, 5.46875)	(-0.0001953125, 0.0001953125)	(-0.0001953125, 0.0001953125)
2. (2.34375, 3.90625)	(2.34375, 3.90625)	(-0.0001953125, -0.0001953125)	(-0.0876953125, -0.0001953125)
3. (0.78125, 2.34375)	(0.78125, 2.34375)	(-0.1751953125, -0.0876953125)	(-0.0876953125, -0.0001953125)
4. (0.78125, 2.34375)	(0.78125, 2.34375)	(-0.1751953125, -0.0876953125)	(0.0001953125, 0.0876953125)
5. (0.78125, 2.34375)	(2.34375, 3.90625)	(-0.0876953125, -0.0001953125)	(0.0001953125, 0.0876953125)
6. (0.78125, 2.34375)	(3.90625, 5.46875)	(-0.0001953125, -0.0001953125)	(0.0001953125, 0.0876953125)
7. (0.78125, 2.34375)	(5.46875, 7.03125)	(0.0001953125, 0.0876953125)	(0.0001953125, 0.0876953125)
8. (2.34375, 3.90625)	(7.03125, 8.59375)	(-0.0001953125, 0.0001953125)	(0.0001953125, 0.0876953125)
9. (0.78125, 2.34375)	(8.59375, 10.15625)	(-0.0876953125, -0.0001953125)	(-0.0876953125, -0.0001953125)

Fig. 3: Spurious trajectory with grid reachability

abstraction. Moreover, we found that all cells had paths leaving the input space. We showcase one such trajectory of cells with this behavior in Fig. 3. In this trajectory, we see that for the first three steps, the velocity component ranges are negative, thereby guiding the spacecraft towards the goal region, but there is a path from cell 3 to cell 4 that induces a positive velocity component, allowing the path to diverge.

Ultimately, the grid abstraction does not lend itself well to the liveness task because such spurious paths are difficult to rule out. While further refinement of the grid approach is possible and could eventually yield a workable approach, we determined that the complexity and difficulty were too high, and abandoned it in favor of the certificate approach mentioned earlier.

VI. Conclusion

We have presented methods for verifying safety and liveness properties for DRL systems using k-induction, Neural Lyapunov Barrier Certificates, and reachability analysis. We explore their effectiveness on a 2D spacecraft docking problem posed in previous work. For this problem, we show how a k-induction based approach can be used alongside a design-for-verification training scheme to provide liveness guarantees. We also discuss how Neural Lyapunov Barrier Certificates can be used to provide both liveness and safety guarantees. While reachability analysis ultimately did not provide any formal guarantees, we discuss the approach and its limitations. In future work, we plan to explore scaling these methods to more complex and realistic control systems.

VII. Acknowledgements

References

[1] P. Alamdari, G. Avni, T. Henzinger, and A. Lukina. Formal Methods with a Touch of Magic. In *Proc. 20th Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 138–147, 2020.

[2] B. Alpern and F. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2:117–126, 09 1987.

[3] A. Ames, X. Xu, J. W. Grizzle, and P. Tabuada. Control barrier function based quadratic programs for safety critical systems. *Trans. on Automatic Control*, 2017.

[4] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada. Control barrier functions: Theory and applications. In *European Control Conf.*, 2019.

[5] G. Amir, D. Corsi, R. Yerushalmi, L. Marzari, D. Harel, A. Farinelli, and G. Katz. Verifying Learning-Based Robotic Navigation Systems. In *Proc. 29th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 607–627, 2023.

[6] G. Amir, O. Maayan, T. Zelazny, G. Katz, and M. Schapira. Verifying Generalization in Deep Learning. In *Proc. 35th Int. Conf. on Computer Aided Verification (CAV)*, pages 438–455, 2023.

[7] G. Amir, M. Schapira, and G. Katz. Towards Scalable Verification of Deep Reinforcement Learning. In *Proc. 21st Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 193–203, 2021.

[8] G. Amir, H. Wu, C. Barrett, and G. Katz. An SMT-Based Approach for Verifying Binarized Neural Networks. In *Proc. 27th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 203–222, 2021.

[9] G. Amir, T. Zelazny, G. Katz, and M. Schapira. Verification-Aided Deep Ensemble Selection. In *Proc. 22nd Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 27–37, 2022.

[10] G. Anderson, S. Pailoor, I. Dillig, and S. Chaudhuri. Optimization and Abstraction: a Synergistic Approach for Analyzing Neural Network Robustness. In *Proc. 40th ACM SIGPLAN Conf. on Programming Languages Design and Implementations (PLDI)*, pages 731–744, 2019.

[11] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin. Hamilton-Jacobi reachability: A brief overview and recent advances. In *Conf. on Decision and Control*, 2017.

[12] G. Basile and G. Marro. Controlled and conditioned invariant subspaces in linear system theory. *Journal of Optimization Theory and Applications*, 3:306–315, 1969.

[13] S. Bassan, G. Amir, D. Corsi, I. Refaeli, and G. Katz. Formally Explaining Neural Networks within Reactive Systems. In *Proc. 23rd Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 10–22, 2023.

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

[15] R. Bunel, I. Turkaslan, P. Torr, P. Kohli, and P. Mudigonda. A Unified View of Piecewise Linear Neural Network Verification. In *Proc. 32nd Conf. on Neural Information Processing Systems (NeurIPS)*, pages 4795–4804, 2018.

[16] J.-T. Camino, C. Artigues, L. Houssin, and S. Mourgues. Linearization of euclidean norm dependent inequalities applied to multibeam satellites design. *Computational Optimization and Applications*, 73:679–705, 2019.

[17] M. Casadio, E. Komendantskaya, M. Daggitt, W. Kokke, G. Katz, G. Amir, and I. Refaeli. Neural Network Robustness as a Verification Property: A Principled Case Study. In *Proc. 34th Int. Conf. on Computer Aided Verification (CAV)*, pages 219–231, 2022.

[18] Y.-C. Chang and S. Gao. Stabilizing neural control using self-learned almost lyapunov critics. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1803–1809, 2021.

[19] Y.-C. Chang, N. Roohi, and S. Gao. Neural lyapunov control. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[20] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8092–8101. Curran Associates, Inc., 2018.

[21] W. Clohessy and R. Wiltshire. Terminal guidance system for satellite rendezvous. *Journal of the aerospace sciences*, 27(9):653–658, 1960.

[22] E. Cohen, Y. Elboher, C. Barrett, and G. Katz. Tighter Abstract Queries in Neural Network Verification. In *Proc. 24th Int. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, 2023.

[23] D. Corsi, G. Amir, G. Katz, and A. Farinelli. Analyzing Adversarial Inputs in Deep Reinforcement Learning, 2024. Technical Report. https://arxiv.org/abs/2402.05284.

[24] D. Corsi, G. Amir, A. Rodriguez, C. Sanchez, G. Katz, and R. Fox. Verification-Guided Shielding for Deep Reinforcement Learning, 2024. Technical Report. http://arxiv.org/abs/2406.06507.

[25] D. Corsi, E. Marchesini, A. Farinelli, and P. Fiorini. Formal verification for safe deep reinforcement learning in trajectory generation. In *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*, pages 352–359, 2020.

[26] C. Dawson, S. Gao, and C. Fan. Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods for robotics and control. *IEEE Transactions on Robotics*, 2023.

[27] C. Dawson, Z. Qin, S. Gao, and C. Fan. Safe nonlinear control using robust neural lyapunov-barrier functions. In *Conference on Robot Learning*, pages 1724–1735. PMLR, 2022.

[28] J. L. C. B. de Farias and W. M. Bessa. Intelligent control with artificial neural networks for automated insulin delivery systems. *Bioengineering*, 9(11):664, 2022.

[29] A. Edwards, A. Peruffo, and A. Abate. A general verification framework for dynamical and control models via certificate synthesis, 2023.

[30] R. Ehlers. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In *Proc. 15th Int. Symp. on Automated Technology for Verification and Analysis (ATVA)*, pages 269–286, 2017.

[31] Y. Elboher, E. Cohen, and G. Katz. Neural Network Verification using Residual Reasoning. In *Proc. 20th Int. Conf. on Software Engineering and Formal Methods (SEFM)*, pages 173–189, 2022.

[32] Y. Elboher, J. Gottschlich, and G. Katz. An Abstraction-Based Framework for Neural Network Verification. In *Proc. 32nd Int. Conf. on Computer Aided Verification (CAV)*, pages 43–65, 2020.

[33] T. Eliyahu, Y. Kazak, G. Katz, and M. Schapira. Verifying learning-augmented systems. *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021.

[34] J. F. Fisac, M. Chen, C. J. Tomlin, and S. S. Sastry. Reach-avoid problems with time-varying dynamics, targets and constraints. In *Proceedings of the 18th international conference on hybrid systems: computation and control*, pages 11–20, 2015.

[35] M. Ganai, Z. Gong, C. Yu, S. L. Herbert, and S. Gao. Iterative reachability estimation for safe reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

[36] M. Ganai, C. Hirayama, Y.-C. Chang, and S. Gao. Learning stabilization control from observations by learning lyapunov-like proxy models. *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2913–2920, 2023.

[37] O. Gates, M. Newton, and K. Gatsis. Scalable forward reachability analysis of multi-agent systems with neural network controllers, 2023.

[38] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[39] S. Govindaraju and D. Dill. Verification by approximate forward and backward reachability. In *1998 IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers (IEEE Cat. No.98CB36287)*, pages 366–370, 1998.

[40] A. Gupta and I. Hwang. Safety verification of model based reinforcement learning controllers, 2020.

[41] W. Haddad and V. Chellaboina. Nonlinear dynamical systems and control: A lyapunov-based approach. *Nonlinear Dynamical Systems and Control: A Lyapunov-Based Approach*, 01 2008.

[42] G. W. Hill. Researches in the lunar theory. *American journal of Mathematics*, 1(1):5–26, 1878.

[43] K.-C. Hsu, V. Rubies-Royo, C. J. Tomlin, and J. F. Fisac. Safety and liveness guarantees through reach-avoid reinforcement learning. In *Proceedings of Robotics: Science and Systems*, Virtual, 7 2021.

[44] T. Huang, S. Gao, and L. Xie. A neural lyapunov approach to transient stability assessment of power electronics-interfaced networked microgrids. *IEEE transactions on smart grid*, 13(1):106–118, 2021.

[45] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety Verification of Deep Neural Networks. In *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*, pages 3–29, 2017.

[46] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers, 2018.

[47] P. Jin, J. Tian, D. Zhi, X. Wen, and M. Zhang. Trainify: A cegar-driven training and verification framework for safe deep reinforcement learning. In *Computer Aided Verification: 34th International Conference, CAV 2022, Haifa, Israel, August 7–10, 2022, Proceedings, Part I*, page 193–218, Berlin, Heidelberg, 2022. Springer-Verlag.

[48] K. D. Julian and M. J. Kochenderfer. A reachability method for verifying dynamical systems with deep neural network controllers, 2019.

[49] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proc. 29th Int. Conf. on Computer Aided Verification (CAV)*, pages 97–117, 2017.

[50] G. Katz, D. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, D. Dill, M. Kochenderfer, and C. Barrett. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *Proc. 31st Int. Conf. on Computer Aided Verification (CAV)*, pages 443–452, 2019.

[51] B. Könighofer, F. Lorber, N. Jansen, and R. Bloem. Shield Synthesis for Reinforcement Learning. In *Proc. Int. Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, pages 290–306, 2020.

[52] L. Kuper, G. Katz, J. Gottschlich, K. Julian, C. Barrett, and M. Kochenderfer. Toward Scalable Verification for Safety-Critical Deep Networks, 2018. Technical Report. https://arxiv.org/abs/1801.05950.

[53] B. Li, S. Wen, Z. Yan, G. Wen, and T. Huang. A survey on the control lyapunov function and control barrier function for nonlinear-affine control systems. *IEEE/CAA Journal of Automatica Sinica*, 10(3):584–602, 2023.

[54] Y. Li. Deep Reinforcement Learning: An Overview, 2017. Technical Report. http://arxiv.org/abs/1701.07274.

[55] A. Lomuscio and L. Maganti. An Approach to Reachability Analysis for Feed-Forward ReLU Neural Networks, 2017. Technical Report. http://arxiv.org/abs/1706.07351.

[56] U. Mandal, G. Amir, H. Wu, I. Daukantas, F. Newell, U. Ravaioli, B. Meng, M. Durling, M. Ganai, T. Shim, G. Katz, and C. Barrett. Formally Verifying Deep Reinforcement Learning Controllers with Lyapunov Barrier Certificates. In *Proc. 24th Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, 2024.

[57] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[58] M. Ostrovsky, C. Barrett, and G. Katz. An Abstraction-Refinement Approach to Verifying Convolutional Neural Networks. In *Proc. 20th. Int. Symposium on Automated Technology for Verification and Analysis (ATVA)*, pages 391–396, 2022.

[59] P. Prabhakar and Z. Afzal. Abstraction Based Output Range Analysis for Neural Networks, 2020. Technical Report. https://arxiv.org/abs/2007.09527.

[60] Z. Qin, T.-W. Weng, and S. Gao. Quantifying safety of learning-based self-driving control using almost-barrier functions. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12903–12910. IEEE, 2022.

[61] Z. Qin, K. Zhang, Y. Chen, J. Chen, and C. Fan. Learning safe multi-agent control with decentralized neural barrier certificates. In *ICLR*, 2021.

[62] U. J. Ravaioli, J. Cunningham, J. McCarroll, V. Gangal, K. Dunlap, and K. L. Hobbs. Safe reinforcement learning benchmark environments for aerospace control systems. In *2022 IEEE Aerospace Conference (AERO)*, pages 1–20. IEEE, 2022.

[63] A. Rodriguez, G. Amir, D. Corsi, C. Sanchez, and G. Katz. Shield Synthesis for LTL Modulo Theories, 2024. Technical Report. http://arxiv.org/abs/2406.04184.

[64] L. H. Sena, I. V. Bessa, M. R. Gadelha, L. C. Cordeiro, and E. Mota. Incremental bounded model checking of artificial neural networks in cuda. In *2019 IX Brazilian Symposium on Computing Systems Engineering (SBESC)*, pages 1–8, 2019.

[65] G. Singh, T. Gehr, M. Puschel, and M. Vechev. An Abstract Domain for Certifying Neural Networks. In *Proc. 46th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*, 2019.

[66] O. So and C. Fan. Solving stabilize-avoid optimal control via epigraph form and deep reinforcement learning. In *Proceedings of Robotics: Science and Systems*, 2023.

[67] V. Talpaert, I. Sobh, B. R. Kiran, P. Mannion, S. Yogamani, A. El-Sallab, and P. Perez. Exploring applications of deep reinforcement learning for real-world autonomous driving systems, 2019.

[68] V. Tjeng, K. Xiao, and R. Tedrake. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *Proc. 7th Int. Conf. on Learning Representations (ICLR)*, 2019.

[69] M. Tong, C. Dawson, and C. Fan. Enforcing safety for vision-based controllers via control barrier functions and neural radiance fields. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10511–10517. IEEE, 2023.

[70] M. Usman, D. Gopinath, Y. Sun, Y. Noller, and C. Păsăreanu. NNrepair: Constraint-based Repair of Neural Network Classifiers, 2021. Technical Report. http://arxiv.org/abs/2103.12535.

[71] H. Wu, O. Isac, A. Zeljić, T. Tagomori, M. Daggitt, W. Kokke, I. Refaeli, G. Amir, K. Julian, S. Bassan, et al. Marabou 2.0: A Versatile Formal Analyzer of Neural Networks. In *Proc. 36th Int. Conf. on Computer Aided Verification (CAV)*, 2024.

[72] X. Xu, P. Tabuada, J. W. Grizzle, and A. D. Ames. Robustness of control barrier functions for safety critical control. *Int. Federation of Automatic Control*, 2015.

[73] Y. Yang, Y. Jiang, Y. Liu, J. Chen, and S. E. Li. Model-free safe reinforcement learning through neural barrier certificate. *IEEE Robotics and Automation Letters*, 2023.

[74] D. Yu, H. Ma, S. Li, and J. Chen. Reachability constrained reinforcement learning. In *International Conference on Machine Learning*, pages 25636–25655. PMLR, 2022.

[75] H. Yu, C. Hirayama, C. Yu, S. Herbert, and S. Gao. Sequential neural barriers for scalable dynamic obstacle avoidance. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11241–11248. IEEE, 2023.