Discovering Closed-Loop Failures of Vision-Based Controllers via Reachability Analysis

Kaustav Chakraborty[1], Somil Bansal[1]

***Abstract*—Machine learning driven image-based controllers allow robotic systems to take intelligent actions based on the visual feedback from their environment. Understanding when these controllers might lead to system safety violations is important for their integration in safety-critical applications and engineering corrective safety measures for the system. Existing methods leverage simulation-based testing (or falsification) to find the failures of vision-based controllers, i.e., the visual inputs that lead to closed-loop safety violations. However, these techniques do not scale well to the scenarios involving high-dimensional and complex visual inputs, such as RGB images. In this work, we cast the problem of finding closed-loop vision failures as a Hamilton-Jacobi (HJ) reachability problem. Our approach blends simulation-based analysis with HJ reachability methods to compute an approximation of the backward reachable tube (BRT) of the system, i.e., the set of unsafe states for the system under vision-based controllers. Utilizing the BRT, we can tractably and systematically find the system states and corresponding visual inputs that lead to closed-loop failures. These visual inputs can be subsequently analyzed to find the input characteristics that might have caused the failure. Besides its scalability to high-dimensional visual inputs, an explicit computation of BRT allows the proposed approach to capture non-trivial system failures that are difficult to expose via random simulations. We demonstrate our framework on two case studies involving an RGB image-based neural network controller for (a) autonomous indoor navigation, and (b) autonomous aircraft taxiing.**

I. Introduction

RECENT advances in computer vision and deep learning have enabled autonomous systems to employ vision-based controllers to perceive their environment and react to it for accomplishing various tasks, including agile navigation [1], manipulation [2], autonomous driving [3], and autonomous aircraft landing and taxiing [4]. Despite their success, such vision-based controllers can fall prey to issues when they are subjected to inputs that are scarcely encountered in the training dataset or are outside the training distribution. For example, a visual policy trained exclusively with well-illuminated images might fail to predict good actions in dark conditions; an autonomous car that is predominately shown to take right turns in an expert dataset, may fail to learn to make left turns. Such vision failures can cascade to catastrophic system failures and compromise system safety. Thus, to successfully adopt vision-based neural network controllers in safety-critical applications, it is vital to analyze them and understand when and why they result in a system failure. In addition to reasoning about system safety, these failure modes might be useful in engineering corrective measures for the system.

While techniques from adversarial learning and robust optimization have been used to find "adversarial" inputs for vision components, they tend to focus on the component-level safety analysis, i.e., detecting failures or errors only within the vision component, ignoring their effect on the downstream system and the overall robot safety. Indeed, not all vision failures are equal from the robot safety standpoint. For instance, the same prediction error by a visual policy for a high-speed drone can be much more catastrophic near a wall compared to an empty hallway. Thus, it is imperative that we analyze these vision modules in conjunction with the robot dynamics. To that end, formal verification techniques have been used for system-level (or closed-loop) safety analysis of dynamical systems; however, their direct application to vision-based controllers remains impractical due to these controllers' high-dimensional and complicated input spaces and the lack of mathematical models relating the robot state to the visual input at that state. Simulation-based testing has been a promising approach to overcoming these challenges; by treating the system as a black box, one can search for system trajectories that result in a failure under the vision-based controller. However, this process (also called falsification) can be highly time-consuming, and it struggles with exposing long-tail of system failures.

In this work, we cast the problem of finding closed-loop vision failures as a Hamilton-Jacobi (HJ) reachability problem and compute the Backward Reachable Tube (BRT) of the system. Given a set of unsafe states (e.g., obstacles for a navigation robot), the BRT is the set of all starting states of the system which ultimately reach an unsafe state under the vision-based controller. Thus, the BRT captures all possible unsafe states of the system. The sequences of visual inputs corresponding to the states in the BRT can be, therefore, classified as the inputs that result in closed-loop system failures, enabling us to discover failures in a systematic manner. Finding closed-loop failures with the help of a BRT also circumvents the need for a direct search in the high-dimensional input space, leading to a tractable discovery of the closed-loop failures.

Typically, the BRT computation requires an analytical model of closed-loop system dynamics. However, such a model is challenging to obtain for vision-based controllers due to the lack of a mathematical model between the system state and the corresponding visual input (and, by extension, the control input at that state). Our key idea in overcoming this challenge is to blend level set-based reachability methods with simulation-based methods to compute a numerical ap-

[1]Authors are with the Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, CA 90089, USA {kaustavc, somilban}@usc.edu. This research is supported in part by the DARPA Assured Neuro Symbolic Learning and Reasoning (ANSR) program and by the NSF CAREER program (2240163).

Project Website: http://vatsuak.github.io/failure-detection

proximation of the BRT. Level set methods compute the BRT over a state-space grid; even though level set methods have typically been employed in the settings where an analytical model of the closed-loop dynamics is available, they only need the system dynamics at the state grid points. Motivated by this observation, we leverage readily available photo-realistic simulators to obtain the visual inputs corresponding to the state grid points and, subsequently, the control inputs. This allows us to compute (approximate) BRTs under a vision-based controller without knowing an analytical model of the environment but rather only from its samples. Our approach is particularly suitable for the vision-based controllers that are trained in simulation or where simulators are readily available for testing.

In summary, the major contributions of this paper are,

- We propose a framework that bridges formal guarantees of HJ reachability with simulation-based methods for a closed-loop safety analysis under vision-based controllers.
- We demonstrate the generalizability of our method by studying two distinct systems, which involve a 3D and a 5D dynamical system using RGB image-based neural network controllers.
- We analyze the obtained image sequences to deduce common failure "scenarios" for the system.

II. Related Work

Component-level safety analysis. A number of recent works have proposed tools to formally verify the input-output properties of neural networks (NN) [5]–[7]. Even for image-inputs, there have been significant advances in generating adversarial input perturbations [8] that result in erroneous outputs, as well as analysis of neuron activations [9] to different input images. However, finding the failures of the vision-module in isolation from the rest of the system does not address the closed-loop system safety challenges.

Closed-loop verification. Closed-loop verification techniques combine the output of input-output NN verification methods with reachability analysis to provide guarantees on the closed-loop performance of NN controllers [10]–[12]. While these approaches work well for state-based controllers with low-dimensional input spaces, they do not scale to high-dimensional vision-based controllers. An additional challenge with perception inputs is that it is challenging to even define the observation space for verification – for example, not all $256 \times 256 \times 3$ arrays make a valid real-world RGB image. To overcome these challenges, there have been attempts to abstract the observation space through GANs [4], piecewise affine abstractions [13], or a geometric sensor mapping [14]. However, the obtained failures are only as accurate as the abstraction itself. Moreover, obtaining accurate abstractions for complex real-world image inputs can be challenging.

Closed-loop falsification. On the other hand, the development and wide availability of photo-realistic simulators and datasets [15]–[17] has presented an opportunity to use them as a black-box representation between the system state and the corresponding visual input, without requiring an explicit analytical model of robot sensor. This has led to development of approaches that find closed-loop perception failures through forward simulation (also referred to as falsification) [18], [19]. Even though promising, these approaches often rely on heuristics to effectively search over the image space for finding the failures, including low-dimensional feature encoding and adaptive Bayesian sampling in the feature space [20], [21]. However, it is not immediately apparent how to obtain informative low-dimensional feature encoding for complex RGB images. In addition, forward simulation approaches can be computationally prohibitive for finding rare, long-tail failure cases. To overcome this challenge, [22] leverages linear system dynamics along with a simulator to efficiently search for closed-loop failures. Our work builds upon this line of work to integrate sampling-based falsification approaches with HJ reachability analysis for exposing closed-loop failures of vision-based controllers for general non-linear systems.

III. Problem Setup

Let us consider a robot in an environment, E. The environment broadly refers to all the factors that are external to the robot (e.g. buildings in which a robot is navigating, the goal/obstacle location, or even characteristics like different weather conditions, time of the day, or camera parameters that might effect the vision of the robot).

We model the robot as a dynamical system with state $\mathbf{x} \in \mathbb{R}^n$, control $u \in \mathscr{U}$ (a compact set), and dynamics:

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

Let S denote the robot's sensor mapping from a state $\mathbf{x}$ to an output (or observation), e.g., a depth or an RGB image $I = S(\mathbf{x}, E)$. In this work, we specifically focus on vision sensors for which I is often high-dimensional. Even though obtaining an analytical model of S is non-trivial for vision sensors, we assume access to a simulator that allows us to query I for a state $\mathbf{x}$. This allows us to leverage recent advances in photorealistic simulators to find vision failures.

Let π denote an output-feedback control policy, that maps observations to the control input u:

$$u := \pi(I, \mathbf{x}, E) \tag{2}$$

For vision-based policies, π often involves neural networks. In our work, π, could be a single end-to-end learning model or can be composed of different submodules. For example, in robot navigation, π might consist of a learning-based visual route planner that serves as an initial guess for an optimization-based trajectory planner for robot control. Let $\zeta_{\mathbf{x}}^{\pi}(\tau)$ be the robot's state achieved at time τ when it starts at state $\mathbf{x}$ at time $t = 0$, and follows the policy π over $[0, \tau]$. Finally, let us denote a set of undesirable or unsafe states by $\mathscr{O} \subset \mathbb{R}^n$. For example, $\mathscr{O}$ could represent obstacles for a navigation task, or off-runway positions for an autonomous aircraft.

In this work, we are interested in finding sequences of input images that lead to a closed-loop system failure. In other words, we wish to find the set of images $\mathscr{I}_{unsafe}$, which, when seen by the visual controller, leads the system into $\mathscr{O}$. We hypothesize that analyzing $\mathscr{I}_{unsafe}$, will then uncover particular properties of the visual inputs that cause the robotic system to fail.

Note that we are interested in finding the visual inputs that lead to a closed-loop failure of the system and not just that of the vision module. However, finding such closed-loop failures is challenging due to (a) high-dimensional observations I, and (b) the lack of an analytical model of S. Our approach of blending simulation-based techniques with HJ-reachability analysis is a key contribution towards overcoming these challenges.

Running example (TaxiNet). Now we introduce the aircraft taxiing problem [4] that we will use as a running example to illustrate the key concepts. Here, the robot is a Cessna 208B Grand Caravan modelled as a three-dimensional nonlinear system with dynamics:

$$\dot{p}_x = v\,cos(\theta) \quad \dot{p}_y = v\,sin(\theta) \quad \dot{\theta} = u \tag{3}$$

where p_x is the crosstrack error (CTE), p_y is the downtrack position (DTP) and θ is the heading error (HE) of the aircraft in degrees from the centreline (Fig. 2(a) shows how these quantities are measured). v is the linear velocity of the aircraft kept constant at 5 m/s, and the control u is the angular velocity.

The goal of the aircraft is to follow the centreline as closely as possible using the images obtained through a camera mounted on its right wing. For this purpose, the aircraft uses a Convolutional Neural Network (CNN), which returns the estimated CTE, $\hat{p}_x$, and the estimated HE, $\hat{\theta}$. A proportional controller (P-Controller) then takes these predicted tracking errors to return the control input, as follows:

$$u := tan(-0.74\hat{p}_x - 0.44\hat{\theta}) \tag{4}$$

Hence, the policy π is a composition of the CNN and the P-Controller. Intuitively, the P-controller is designed to steer the aircraft towards the centreline based on the state estimate provided by the CNN.

The image observations are obtained using the X-Plane flight simulator that can render the RGB image, I, from a virtual camera (S) mounted on the right wing of the aircraft at any state and a given time of the day (see Fig. 2(c), (d) for representative images). Note that the CNN here is also trained in simulation using the data collected from X-Plane. Please see [4] for the training details.

We define the unsafe states for the aircraft as $\mathscr{O} = \{\mathbf{x} : |p_x| \geq 10\}$, which corresponds to the aircraft leaving the runway. The environment E is the runway 04 of Grant County International Airport. Our goal is to find the set of input images that eventually drive the aircraft off the runway under the control policy in (4).

IV. Background: Hamilton-Jacobi Reachability

In this work, we will find closed-loop system failures using HJ reachability analysis. In reachability analysis, one is interested in computing the Backward Reachable Tube (BRT) – the set of all initial states, such that an agent starting from those states will reach the target set $\mathscr{O}$ within the time horizon $[t,T]$ under policy $\pi(\mathbf{x})$.

We define the BRT for a closed loop system as follows:

$$\mathscr{V} := \{\mathbf{x} : \exists \tau \in [t,T], \zeta^{\pi}_{\mathbf{x}}(\tau) \in \mathscr{O}\} \tag{5}$$

HJ reachability allows us to compute the BRT for general nonlinear systems, while handing cases of control and disturbance inputs to the system over arbitrary shaped target sets. In HJ reachability, the BRT can be computed using level-set methods [23], [24]. In the level-set methods, the target set is represented as a sub-zero level set of a function $l(\mathbf{x})$, i.e., $\mathscr{O} = \{\mathbf{x} : l(\mathbf{x}) \leq 0\}$. Usually, $l(\mathbf{x})$ is given by the signed distance function to $\mathscr{O}$. With this formulation, the BRT computation can be reframed as an optimal control problem that requires the computation of a value function defined as:

$$V(\mathbf{x},t) = \min_{\tau \in [t,T]} l(\zeta^{\pi}_{\mathbf{x}}(\tau)) \tag{6}$$

The value function in (6) can be computed recursively using the dynamic programming principle. This results in a partial differential equation referred to as Hamilton-Jacobi-Bellman Variational Inequality (HJB-VI) [23]:

$$\begin{aligned} &\min\{D_t V(\mathbf{x},t) + H(\mathbf{x},t), l(\mathbf{x}) - V(\mathbf{x},t)\} = 0 \\ &\text{with } V(\mathbf{x},T) = l(\mathbf{x}) \end{aligned} \tag{7}$$

Here, D_t and ∇ represent the time and spatial gradients of the value function. $H := \langle \nabla V(\mathbf{x},t), f(\mathbf{x},\pi(\mathbf{x})) \rangle$ is the called the Hamiltonian. Intuitively, (7) is a continuous-time counterpart of the Bellman equation in discrete time. Once the value function is computed, the BRT is given as the set of states from which entering into the target set cannot be avoided, i.e., the optimal signed distance to $\mathscr{O}$ is negative. Thus, the BRT is given by the subzero level set of the value function:

$$\mathscr{V} = \{\mathbf{x} : V(\mathbf{x},t) \leq 0\} \tag{8}$$

In the next section, we discuss how $\mathscr{V}$ can be used to find closed-loop failure inputs for vision-based policies.

V. Closed-Loop Failure Discovery via HJ Reachability

We cast the problem of finding closed-loop vision failures as a HJ reachability problem. Specifically, given the set of undesirable states $\mathscr{O}$, the sensor mapping can be composed with the vision-based controller to obtain the closed-loop, state-feedback policy, $\hat{\pi}$ for a given environment:

$$u = \pi(I,\mathbf{x},E) \;=\; \pi(S(\mathbf{x},E),\mathbf{x},E) \implies u = \hat{\pi}(\mathbf{x}) \tag{9}$$

Given the policy $\hat{\pi}$, we compute the BRT $\mathscr{V}$ by solving the HJB-VI in (7). Intuitively, $\mathscr{V}$ represents the set of all initial states of the robot that will eventually enter $\mathscr{O}$ under the vision-based controller. Once computed, $\mathscr{V}$ can be used to obtain the failure inputs as:

$$\mathscr{I}_{unsafe} = \{I : I = S(\mathbf{x},E), \mathbf{x} \in \mathscr{V}\}, \tag{10}$$

which can be subsequently analyzed to find the common characteristics of the failure inputs. Finding $\mathscr{I}_{unsafe}$ through a BRT allows us tractably search for failures over the high-dimensional input (image) space by converting it into a search over the state space, which is typically much lower dimensional.

However, existing HJ reachability methods typically require an analytical expression of $\hat{\pi}$ to solve the HJB-VI, which in turn requires an analytical model of the sensor mapping S. Unfortunately, obtaining such models of S remains a challenging problem, especially for visual sensors; consequently, obtaining closed-loop policy $\hat{\pi}$ in an analytical form is difficult. To overcome this challenge, we leverage level set methods [25] to compute a numerical approximation of the BRT. Level set methods solve the HJB-VI numerically over a uniformly discretized state-space grid. Thus, it is sufficient to know the output of $\hat{\pi}$ at the discretized states. To compute the controller

output at the grid points, we turn to photorealistic simulators that allow us to query I, and subsequently u, at any state $\mathbf{x}$. Thus, blending simulator-based sampling with level set methods from HJ reachability allows us to approximate the BRT from the samples of S, in place of a rigorous model of S. The approximate BRT can then be used to find the failure inputs as in (10).

In summary, our proposed approach consists of three main steps: **(1)** obtaining the closed-loop dynamics of the system via sampling over a uniform grid of the robot's statespace, **(2)** computing the BRT to extract the set of images leading to closed-loop system failures, and **(3)** analyzing these images to uncover their interesting properties. Overall, the proposed approach enables the failure analysis of complicated and high-dimensional visual controllers without possessing an explicit analytical model of the environment or the sensor.

VI. Case Studies

A. Autonomous Aircraft Taxiing (TaxiNet)

For the running example, we obtain the BRT over the statespace: $[-11, 11]m$ for p_x, $[100, 250]m$ for the p_y and $[-28°, 28°]$ for θ. In order to compute the BRT we use a uniform $101 \times 101 \times 101$ grid along p_x, p_y and θ directions and leverage the Level Set Toolbox [25] to solve the HJB-VI numerically.

To obtain $\hat{\pi}(\mathbf{x}_i)$ at any grid point $\mathbf{x}_i$, we obtain the $360 \times 200 \times 3$ RGB image observation from the virtual camera (S) in the X-plane simulator (see Figs. 2(c),(d). This image is fed through the CNN (C) to obtain the estimates of crosstrack and heading errors, which are subsequently utilized by the P-controller (P) in (4) to obtain the control input. Hence $\hat{\pi}$ for this study is the composition, $P \circ C \circ S$.

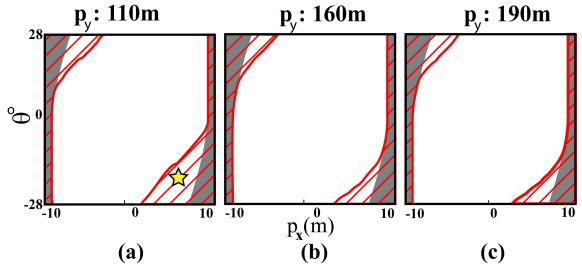

Fig. 1. Closed-loop BRTs of the ideal system (the grey area) and the actual system (the striped red area) during the morning when the aircraft starts at p_y **(a)** 110m, **(b)** 160m, and **(c)** 210m. The vision-based controller leads to particularly unsafe behaviors near the boundary of the runway.

The slices of the obtained BRT (the red striped area) are shown in Fig. 1 for different values of downtrack position p_y. With the change in p_y, the CNN is able to observe different regions along the length of the runway which effects its prediction capabilities. The ability of the aircraft in successfully accomplishing its taxiing task (reflected through the area enclosed by the BRT) was hence seen to be critically dependent on the starting p_y. For comparison purposes, we also compute the system BRT assuming an "ideal" CNN (the shaded grey region). In this case, it would mean that the CNN predicts the system state perfectly (referred to as the ideal system henceforth). Any observed performance drop from this assessment on reintroducing the CNN in the pipeline (called the actual system henceforth) can be attributed to the vision module. Note that, as expected, the BRT for the ideal system is a proper subset of the BRT for the actual system. We now leverage the obtained BRT to analyze different failure modes of the vision-based controller.

Failures near the boundary of the runway. Upon comparing the BRTs for the actual and the ideal systems, it is evident that the vision module particularly leads to more safety violations near the boundary (left and right) of the runway. To analyze these failures further, we plot the error heatmap between the predicted state (by the CNN) and the actual state for different p_x and θ for $p_y = 160m$ in Fig. 2(b). We observe that the top-left and the bottom-right areas of the heatmap display an unusually high localization error by the CNN. On querying some representative images (2(c),(d)) observed by the CNN at these critical states, we find that when the aircraft is closer to the boundary of the runway while looking away from the center lane, only a reduced portion of the runway is visible in the image observations, leading to erroneous predictions from the CNN. Such errors, in turn, lead to unsafe control decisions, leading to a striking resemblance between the BRT (Fig. 1) and the heatmap (Fig. 2(b)).

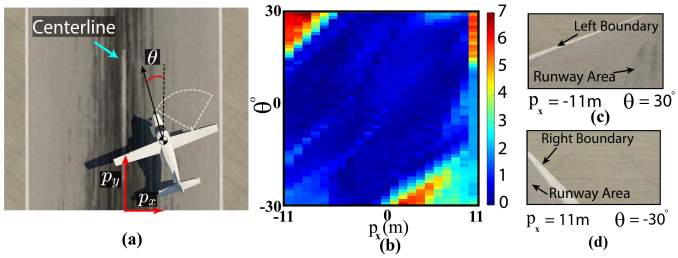

Fig. 2. **(a)** The autonomous aircraft taxiing example. p_x, p_y, θ denote the state of the aircraft; the FoV of the camera is shown with dashed-white lines. **(b)** Heatmap showing the error in prediction over p_x and θ for $p_y = 160m$. Asymmetric images **(c), (d)** seen by the CNN at symmetric locations about the centerline.

Failures due to asymmetric camera placement. It is also interesting to note that even though the BRT is symmetric for the ideal system, it is not the case for the actual system (Fig. 1). Specifically, the size of the BRT is bigger when the vehicle is to the right of the centreline. Upon a closer inspection, it turns out that the camera is mounted on the right wing of the aircraft, leading to asymmetric observations from the two sides of the centreline, as shown in Fig. 2(c), (d). This leads to the aircraft having a wider view of the runway when it starts from the left of the centreline (Fig. 2(c) has more view of the track than Fig. 2(d)), leading to a better localization by the CNN. This example demonstrates the efficacy of the proposed approach in comparing the failures of the controller in different parts of the environment.

Failure due to the presence of runway marking. We analyze the changes in the BRT with variations in the starting positions of the aircraft along the runway, i.e., with p_y. We observe that the BRT corresponding to p_y = 110m (Fig. 1(a)) is peculiarly large, especially around the bottom-right (when the aircraft starts on the right of the runway with a negative heading). We simulate the aircraft trajectory from a state in the BRT (marked with the yellow star in Fig. 1(a)) and query the images observed by the aircraft along the trajectory. The aircraft

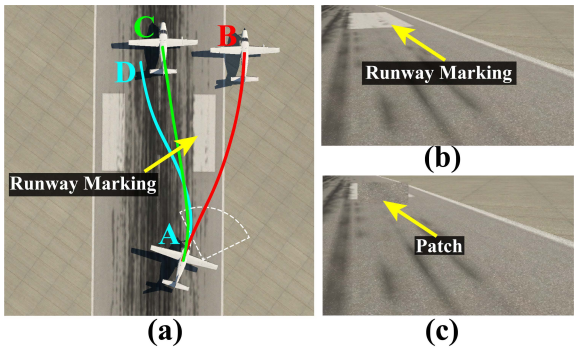

Fig. 3. **(a)** Top-view of the runway in the morning. The trajectory followed by the aircraft under the CNN policy (red line) takes it off the runway. The successful trajectory (in green) takes the aircraft from "A" to "C", on adding the patch over the runway marking during ablation. The trajectory (in cyan) from "A" to "D" is followed at night. **(b)** The actual image that the CNN sees at "A" (yellow star in Fig. 3(a)). The CNN confuses the runway marking as the centreline. **(c)** Modified image with an artificial patch over the runway marking.

trajectory is shown in red in Fig. 3(a). We notice that the CNN confuses a runway side strip marking (Fig. 3(b)) with the centreline and steers the aircraft towards it, ultimately leading the aircraft into the unsafe zone ($|p_x| \geq 10$m). We also perform an ablation study where we mask off the area of the runway marking in the morning with a patch having the same color as the runway (shown in Fig. 3(c)). The corresponding aircraft trajectory is shown in green in Fig.3(a). On being unable to see the runway marking, the aircraft completed its taxiing task successfully, indicating that the runway marking is indeed "fooling" the CNN. This example illustrates that the shape and analysis of the BRT makes it easier to find such subtle failures of the vision-based controller, which are hard to expose via random sampling. The two methods are compared further toward the end of this section.

Effect of the time of the day on the vision failures. To understand the impact of the time of the day on the vision failures, we also compute the BRT of the system at the night time (2100 hrs.).

In Fig. 4(a), we overlay the BRT for the morning and the night time for p_y =110m. The overlaid BRTs provide us a quick way to systematically check the CNN's performance for all states of interest. We saw that the BRT for the night was smaller than that for the morning (around the bottom-right area), indicating that during the night time the CNN was not affected by the runway marking. This was confirmed via the trajectory followed by the aircraft at night (the cyan trajectory in Fig. 3(a)). It was a surprising discovery since night-time performance is typically worse for vision-based systems due to reduced visibility and illumination. However, it seems that the aircraft benefits from not being able to see certain parts of the runway that can potentially confuse its vision system. In this case, the CNN could not see the runway marking at night due to low illumination, obviating a possible confusion with the centreline! In order to understand the effect of the visibility of runway marking on the failure of the CNN's prediction, we feed the CNN with different night-time images at the same state (Fig. 4(c)). However, in each of these images we blend (with varying proportions) a cropped-out version of the runway

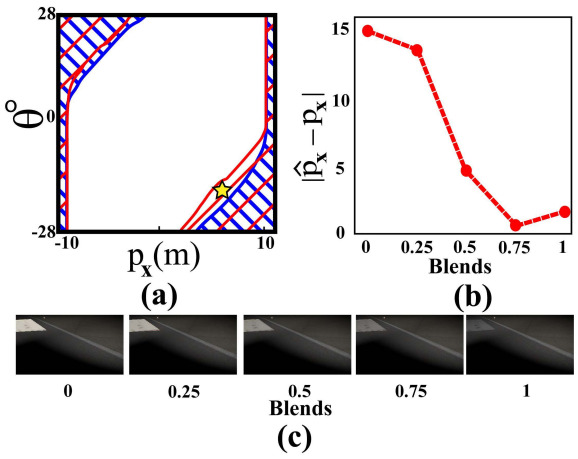

Fig. 4. **(a)** The morning (red shaded), and the night (blue shaded) BRTs overlaid for a p_y of 110m. The state of interest, shown with a yellow star, is only contained in the morning BRT and not in the night BRT. **(b)** The absolute difference in $\hat{p_x}$ and the ground truth p_x vs. the different blends of the runway marking at the state of interest (a lower value is better). **(c)** The images (at the yellow star in (a)) corresponding to different blends. The right-most image has a blend of 1, which is the unmodified image that the aircraft observes at night. The left-most image with a blend of 0 is obtained by manually cropping the runway marking and replacing it by the patch from the image observed in the morning. Any intermediate image is an interpolation between these two according to the blend value.

marking that the CNN sees at the same state in the morning. This changes the effective illumination/brightness only around the marking while keeping the rest of the image the same. We found that the error in the p_x prediction decreased on decreasing the runway marking brightness (Fig. 4(b)), showing that the CNN is indeed able to make correct predictions when the marking is not well illuminated.

Comparing the BRTs for the morning and the night for other p_y, we observe that the night BRT is particularly big around the states where the aircraft starts near the left boundary of the runway (states around the top left corner of the Fig. 5(a)).

Based on the overlaid BRTs, we can predict that when the aircraft starts around the state marked by the yellow star, it should enter an unsafe state when guided by the CNN policy in the night-time but will successfully complete the taxiing task in the morning. The observed images along the aircraft

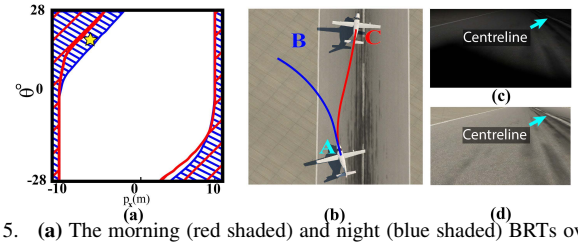

Fig. 5. **(a)** The morning (red shaded) and night (blue shaded) BRTs overlaid for p_y = 190m. The state, shown with a yellow star, is only included in the night BRT. **(b)** Top view of the runway. In the morning, the CNN policy accomplishes the taxiing task by taking the red trajectory from "A" (yellow star in (a)) to "C." At night, the policy takes the aircraft outside the runway along the blue trajectory from "A" to "B". **(c)** The centreline in the image cannot be vividly seen by the CNN at location "A" at night due to poor illumination, whereas it can be seen clearly in the morning **(d)**.

trajectory (Fig. 5(c),(d)) expose that at night time the CNN is indeed unable to properly see the centreline due to illumination issues guiding the aircraft off the runway (blue trajectory from location A to B in Fig. 5(b)). However, such errors are avoided

in the morning (red trajectory from location A to C in Fig. 5(b)) due to a better visibility.

Comparison with forward-simulation methods. Our method took ∼6.5 hours to compute $\hat{\pi}$, the BRT, and isolate the failure cases for the entire system. The majority of this time was spent in computing $\hat{\pi}$, which involves rendering images at state grid points and querying the CNN. For comparison purposes, we also estimated the time that could be taken to find failure cases using a forward simulation of trajectory from each of the states in our state-space grid. The approximate computation time is around ∼67.5 days (it takes an average of 6s per trajectory simulation on a 24GB Nvidia RTX 3090 GPU. We required $\sim 10^5$ input datapoints to sample the statespace appreciably.). The bottleneck in forward simulation primarily comes from a combination of *repeated* image rendering, forward CNN calls, and post-processing of the CNN predictions for the trajectory simulation. On the other hand, our reachability-based approach leverages dynamic programming to alleviate the need for repeated system queries.

Another advantage of our approach is its ability to expose hard-to-find, corner cases. Corner cases, such as the failure due to the presence of a runway marking, occur from precise, sparsely-located starting positions in the aircraft's state space. In this case, it required us to complete an average of 145 simulations just to arrive at a single occurrence of such a failure mode when using a forward simulation-based random search. The number of simulations is only expected to increase further with the number of robot states. Finally, overlaying the BRT slices provides an intuitive way to compare the failures modes of a system across different environmental conditions. These evaluations show the strength of HJ reachability-based methods in systematically discovering the closed-loop failures of high-dimensional, vision-based controllers over random or uniform grid searches.

B. Autonomous Visual Navigation in Indoor Environments

Our second case study analyzes a state-of-the-art visual navigation controller [26] for a wheeled robot navigating an unknown indoor environment. The robot is modeled as a 5-dimensional nonlinear system with dynamics:

$$\dot{p}_x = v\cos(\theta),\ \dot{p}_y = v\sin(\theta),\ \dot{\theta} = \omega,\ \dot{v} = a,\ \dot{\omega} = \alpha$$

Here, p_x and p_y denote the xy-position of the robot, θ is the robot heading, and v and ω are its linear and angular speeds. It is assumed that the robot has access to perfect state estimates. We define the control as $u := (a,\ \alpha)$, where a is the linear acceleration, and α is the angular acceleration. The navigation pipeline includes the CNN C that accepts a $224 \times 224 \times 3$ RGB image I (from an onboard camera S), the instantaneous linear and angular speeds of the robot (v and ω respectively), and a goal position, g to predict an intermediate waypoint position towards which the robot should move. Finally, a model-based spline planner P takes in the predicted waypoint to produce a smooth control profile for the robot. Hence, the closed-loop policy $\hat{\pi}$ is given by $\hat{\pi} := P \circ C \circ S(\mathbf{x}, g, E)$. The environment E consists of various buildings in the Stanford Building Parser Dataset (SBPD) [17].

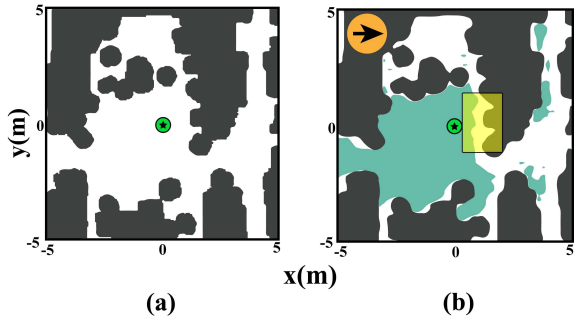

Fig. 6. **(a)** Ground truth occupancy map over a 5×5 area of an office space. The obstacles are shown in dark grey, the free space is shown in white, and the goal area is shown in green. **(b)** The corresponding BRAT slice is shown in teal for $\theta = 0\ rad$ (the initial heading is marked with the arrow inside the orange circle at the top left corner of the plot) with, $v = 0.3\ m/s$, and $\omega = 0\ rad/s$. The yellow-shaded region highlights the states that are not included in the BRAT, indicating unsafe closed-loop behavior while starting from these states.

Given a robot state, we can generate photo-realistic images ($I = S(\mathbf{x}, g, E)$) of indoor scenes taken by a virtual monocular RGB camera S, mounted on the robot. Representative top-view occupancy map from the environment is shown in Fig. 6(a). Note that the obstacles are not known to the robot and it must navigate only using the first-person RGB images.

The CNN is a modified ResNet-50 architecture that is trained entirely in simulation using the SBPD dataset and shown to have a zero-shot sim-to-real generalization [26]. We wish to find and analyze the $\mathscr{I}_{unsafe}$ for the vision-based controller ($\hat{\pi}$) that leads to robot failures. In this case, failure is defined as the robot colliding with the obstacles, such as furniture or walls, before it reaches its goal. To find $\mathscr{I}_{unsafe}$, we compute the Backward Reach-Avoid Tube (BRAT) for the system – the set of states from which the robot reaches its goal without colliding with any obstacles. The complement of the BRAT thus represents the unsafe states for the robot under $\hat{\pi}$. BRAT can be computed by solving the HJB-VI in a similar fashion as BRT, with an additional constraint that the robot trajectory terminates once it reaches the obstacles.

The BRAT is computed for a 5-dimensional grid, of $51 \times 51 \times 21 \times 21 \times 21$ points, over a section of the robot's statespace. $\mathscr{O}$ is given by the set of all obstacles in the environment, and the goal area is given by the positions that are within a distance of $0.3m$ from the position g. For each state on the 5D grid, we render the observed RGB image using the SBPD simulator, query the waypoint using the CNN, and then obtain the control input using the planner P. In Fig. 6(b), we show the 2D projection of the 5D BRAT corresponding to the occupancy map in Fig. 6(a). If the robot starts from any state enclosed by the BRAT (teal area), it will reach the goal area (green circle) under the CNN-policy while avoiding the obstacles (shown in grey). Conversely, the area outside the BRAT represents the unsafe states for the robot. Note that in this case, we do not compute the BRAT for the ideal system because the system is fully controllable, resulting in the ideal BRAT being simply the complement of the obstacle set.

Failure to predict precise waypoints near obstacles. We notice that closer to the obstacles, the BRAT boundary is further from the obstacle boundary (e.g., yellow shaded region in Fig. 6(b)), indicating that the vision-based controller leads

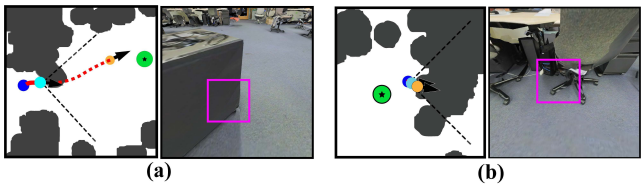

Fig. 7. Simulation cases showing that the robot **(a)** collides with the corner of a table, or **(b)** runs head-on into the chairs. The corresponding robot's trajectories are shown in (red). The cyan marker shows the critical state of the robot. The field-of-view is shown with two dotted black lines from the critical state. The corresponding RGB image is the robot's observation at that state. The orange marker and the dotted red line respectively denote the predicted robot waypoint and trajectory at the critical state. The magenta square in the image highlights the collision region.

to a collision when the robot is near the obstacles. These cases are simulated in Figs. 7(a), and (b). Even though the closed-loop controller is trying to avoid the obstacles (the predicted robot trajectory is going around the obstacles), the trajectory is not turning enough and the robot collides with areas such as the corner of a table (Fig. 7(a)), or the legs of a chair (Fig. 7(b)). Intuitively, the robot needs to predict very precise waypoints when it is closer to the obstacles, since even minor errors can lead to collisions.

However, it is challenging for the CNN to reason about the exact geometry of these complex obstacles solely from the first-person RGB images, which are particularly occluded near the obstacles, leading to inevitable errors in the waypoint prediction.

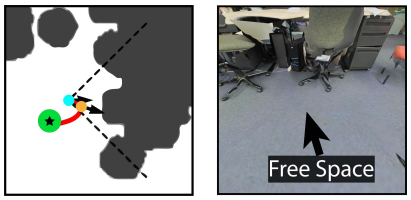

Fig. 8. Allowing the robot visibility of the free space leads to a better waypoint prediction taking the robot to the goal.

To confirm the CNN's reduction in performance near obstacles, we moved the robot's starting position away from the chairs for the simulation in fig. 7(b). The CNN could then see a less occluded image (Fig. 8 right) and predicted a better waypoint to eventually reach the goal (Fig. 8 left).

Failure to discern the traversability of narrow gaps. Further analyzing the BRAT slices and the unsafe states indicate that the closed-loop controller tries to steer the robot through gaps that are too narrow for it to traverse through, given its geometry (a circular base with radius 0.2m). Such states are shown in Fig. 9(a) (inside the magenta-shaded region), and the robot trajectories from such states are simulated in Fig. 9(b),(c). Intuitively, from the RGB images, it seems that the gap is traversable, leading a RGB input-based CNN to predict a waypoint through the gap in order to reach the goal faster. However, the gap is actually not traversable (as indicated by the absence of the gap in the occupancy map which is expanded by the size of the robot), ultimately leading to a collision with the obstacles.

Even when the gap is traversable, the CNN seems to struggle with steering the robot through narrow passages, as indicated by the unsafe states near the top of Fig. 9(a). These states ultimately need to go through a narrow passage in order to reach the goal which the closed-loop controller is unable to do reliably. Such failures also indicate the shortcomings of the

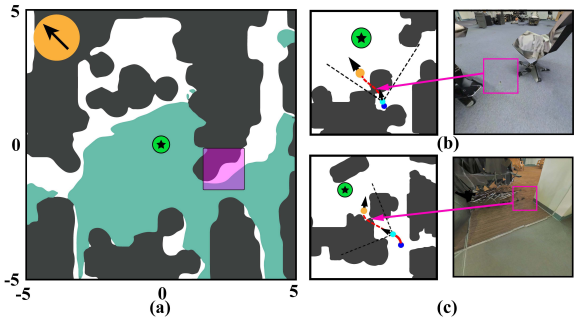

Fig. 9. **(a)** The BRAT slice corresponding to $\theta = 3\pi/4$, $v = 0.3\ m/s$, and $\omega = 3\ rad/s$. The magenta square shows the states that are not covered by the BRAT. **(b)** Simulation from one such state shows that the CNN fails to discern the traversability of narrow passages. The CNN predicts a waypoint that leads the robot through the space between the chair and the table even though it does not possess enough clearance, eventually leading to a collision. The narrow passage is marked with a magenta square on the RGB image seen by the robot (right). **(c)** A similar case is observed between a stack of chairs and the wall.

choice of sensor for the system. For example, we hypothesize that adding another layer of depth estimation to the inputs could alleviate these failure modes, as it will provide the robot with better traversability information. Targeted training of CNN near the narrow passages might also improve the robot performance in such scenarios.

Misunderstanding certain obstacles as traversable. Our BRAT slices indicate that the robot is able to traverse through hallways reasonably well; however, sometimes, it fails. We

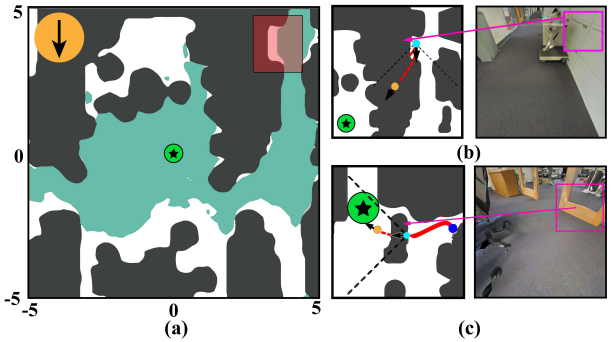

Fig. 10. **(a)** Notice the highlighted area in the top-right location of the BRAT for the robot heading of $-\pi/2$ radians. Even though the robot faces down (wrt the top view), it cannot escape from the recessed region. **(b)** On simulating the robot from one of the highlighted states, we saw that the CNN predicts a waypoint into the wall to its right and crashes the robot. We show the specific wall and its corresponding location on the top view with the magenta arrow. **(c)** Another situation was observed where the robot crashed into a glass door due to the low height of the wooden pane around it. We show the glass door and its corresponding location on the top view with the magenta arrow.

highlight such states in the red-shaded region in Fig. 10(a). This is surprising since the robot simply needs to continue to move straight in such cases. Simulating the robot trajectory (Fig. 10(b)) from these states indicates that the CNN confidently predicts the waypoints inside the wall in order to reach the goal faster, ultimately leading to a collision.

To understand why the robot fails near this wall but not others, we change the color of the wall with the color picked from a different wall, as shown in the right image in Fig. 11(a). This resulted in CNN correctly predicting the waypoint near the floor areas (left image in Fig. 11(a)), indicating that the CNN somehow assumed the original wall to be traversable.

For our second example, we observed that if the CNN were fed with images containing obstacles with a small apparent height (such as glass gates), it would often ignore the obstacle as if it were a traversable area. A simulation at one of these failure states exposed a glass door, as shown in fig. 10(c). The network conveniently ignores the glass door as it can see the ground area through it.

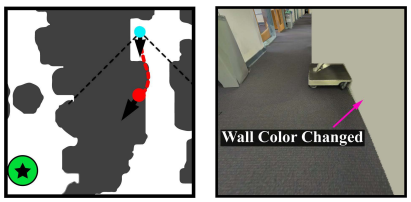

Fig. 11. **(a)** On changing the color of the wall (right image), the CNN predicts a more feasible waypoint (left).

Comparison with forward-simulation methods. Our method took ∼6.5 days to compute the BRAT, out of which 6 days were spent on image rendering at grid points and the CNN query. Even though the computation time is non-trivial, it is still significantly lower than that for a forward simulation-based procedure, which is estimated to be ∼11K days (extrapolated based on 10 trajectories). We recorded an average simulation time of ~ 40 seconds per trajectory over a statespace discretized into $\sim 2.4 \times 10^7$ datapoints. Most of this computation time is again due to the repeated queries of the simulator, forward passes through the CNN, and post-processing the CNN predictions while simulating a trajectory, which can be extremely slow.

Moreover, the BRAT slices provide an intuitive clustering of the failure states, which can be helpful for targeted data collection to improve the CNN.

VII. Discussion and Future Work

We introduce a framework for automatically discovering the closed-loop failures of vision-based controllers. Our work combines simulation-based approaches with HJ reachability analysis to systematically and tractably find these failures. We demonstrate the efficacy of our approach on two distinct applications – a 5-dimensional wheeled robot and a 3-dimensional aircraft using RGB-image-based neural network controllers.

Our work suggests a number of interesting future directions. First, the grid-based approach to computing reachable sets suffers from the curse of dimensionality. In the future, it would be interesting to consider alternative sampling-based reachability methods, such as DeepReach [27], that are shown to scale well to high-dimensional systems. We will also like to explore using the obtained failures in improving the performance of the vision-based controller, e.g., through incremental training on the obtained scenarios. Finally, even though the failure discovery is automatic, the analysis of the failures is primarily performed manually in the current work. Automating the failure analysis, e.g., through generative and clustering methods, will be a promising future direction.

References

[1] A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep drone racing: From simulation to reality with domain randomization," *T-RO*, vol. 36, no. 1, pp. 1–14, 2019.

[2] A. Wang, T. Kurutach, K. Liu, P. Abbeel, and A. Tamar, "Learning robotic manipulation through visual planning and acting," *arXiv preprint arXiv:1905.04411*, 2019.

[3] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *JFR*, vol. 37, no. 3, pp. 362–386, 2020.

[4] S. M. Katz, A. L. Corso, C. A. Strong, and M. J. Kochenderfer, "Verification of image-based neural network controllers using generative models," in *DASC*. IEEE, 2021, pp. 1–10.

[5] V. Tjeng, K. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," *arXiv preprint arXiv:1711.07356*, 2017.

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

[7] R. A. Brown, E. Schmerling, N. Azizan, and M. Pavone, "A unified view of sdp-based neural network verification through completely positive programming," in *AISTATS*, vol. 151. PMLR, 2022, pp. 9334–9355.

[8] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *ICCAV*. Springer, 2017, pp. 3–29.

[9] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *SOSP*, 2017, pp. 1–18.

[10] C. Huang, J. Fan, W. Li, X. Chen, and Q. Zhu, "Reachnn: Reachability analysis of neural-network controlled systems," *ACM TECS*, vol. 18, no. 5s, pp. 1–22, 2019.

[11] W. Xiang, H.-D. Tran, and T. T. Johnson, "Output reachable set estimation and verification for multilayer neural networks," *IEEE TNNLS*, vol. 29, no. 11, pp. 5777–5783, 2018.

[12] K. D. Julian and M. J. Kochenderfer, "Guaranteeing safety for neural network-based aircraft collision avoidance systems," in *DASC*, 2019.

[13] C. Hsieh, K. Joshi, S. Misailovic, and S. Mitra, "Verifying controllers with convolutional neural network-based perception: a case for intelligible, safe, and precise abstractions," *arXiv preprint arXiv:2111.05534*, 2021.

[14] U. Santa Cruz and Y. Shoukry, "Nnlander-verif: A neural network formal verification framework for vision-based autonomous aircraft landing," in *NASA Formal Methods*. Springer, 2022, pp. 213–230.

[15] "Laminar Research: X-Plane 11 (2019)," https://www.x-plane.com/.

[16] "Matterport," https://matterport.com/.

[17] I. Armeni, A. Sax, A. R. Zamir, and S. Savarese, "Joint 2D-3D-Semantic Data for Indoor Scene Understanding," *ArXiv e-prints*, Feb. 2017.

[18] F. Indaheng, E. Kim, K. Viswanadha, J. Shenoy, J. Kim, D. J. Fremont, and S. A. Seshia, "A scenario-based platform for testing autonomous vehicle behavior prediction models in simulation," *arXiv preprint arXiv:2110.14870*, 2021.

[19] D. J. Fremont, E. Kim, Y. V. Pant, S. A. Seshia, A. Acharya, X. Bruso, P. Wells, S. Lemke, Q. Lu, and S. Mehta, "Formal scenario-based testing of autonomous vehicles: From simulation to the real world," in *ITSC*. IEEE, 2020, pp. 1–8.

[20] T. Dreossi, S. Ghosh, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Systematic testing of convolutional neural networks for autonomous driving," *arXiv preprint arXiv:1708.03309*, 2017.

[21] S. Ghosh, Y. V. Pant, H. Ravanbakhsh, and S. A. Seshia, "Counterexample-guided synthesis of perception models and control," in *ACC*. IEEE, 2021, pp. 3447–3454.

[22] L. Yang and N. Ozay, "Synthesis-guided adversarial scenario generation for gray-box feedback control systems with sensing imperfections," *ACM TECS*, vol. 20, no. 5s, pp. 1–25, 2021.

[23] I. Mitchell, A. Bayen, and C. J. Tomlin, "A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games," *TAC*, vol. 50, no. 7, pp. 947–957, 2005.

[24] I. Mitchell and C. J. Tomlin, "Level set methods for computation in hybrid systems," in *HSCC*. Springer, 2002, pp. 310–323.

[25] I. M. Mitchell *et al.*, "A toolbox of level set methods," *UBC Department of Computer Science Technical Report TR-2007-11*, p. 31, 2007.

[26] S. Bansal, V. Tolani, S. Gupta, J. Malik, and C. Tomlin, "Combining optimal control and learning for visual navigation in novel environments," in *CoRL*. PMLR, 2020, pp. 420–429.

[27] S. Bansal and C. J. Tomlin, "Deepreach: A deep learning approach to high-dimensional reachability," in *ICRA*. IEEE, 2021, pp. 1817–1824.