Interval Image Abstraction for Verification of Camera-Based Autonomous Systems

Habeeb P, Deepak D'Souza, Kamal Lodaya, Pavithra Prabhakar

***Abstract*—We propose an abstraction-refinement based algorithm for the problem of verifying the safety of a camera-based autonomous system in a synthetic 3D-scene, based on the notion of interval images. An interval image is an abstract data structure that represents a set of images in a 3D-scene. We give a Computer Graphics style rendering algorithm to efficiently compute interval images from a given region. Our proposed abstraction-refinement algorithm leverages recent abstract interpretation tools for neural networks. We have implemented and evaluated the proposed technique on complex 3D-scenes, demonstrating its effectiveness and scalability in comparison with earlier techniques.**

***Index Terms*—Verification, autonomous systems, abstraction, refinement, abstract interpretation.**

I. Introduction

Autonomous vehicles typically make use of deep neural networks (DNNs) to interpret inputs from their perception sensors like cameras and lidars. The problem of reasoning about the safe navigation of such vehicles is both important and challenging. The de facto solution is to test the vehicle in different scenarios in specialized testing tracks or on roads and other spaces. However this can be expensive in terms of time and resources, and offers very limited coverage. Studies [34] estimate that to guarantee a probability of 10^{-9} fatality per hour of driving one would need a prohibitive thirty billion miles of test driving. A promising alternative is to test a *model* of the vehicle (which captures its sensing mechanism and control dynamics) in a *synthetic*, Computer Graphics (CG) style, 3D-environment. Studies [14], [19] have shown that both safe and unsafe trajectories in synthetic environments transfer well to real-life test scenarios, in that a majority of unsafe simulated behaviours could be reproduced as unsafe behaviours in a real track, while most safe simulated behaviours continued to be safe in the real track. Thus, formal analysis using synthetic environments is a reasonable approximation to achieving higher levels of assurance in the safe functioning of autonomous vehicles. Furthermore, there is a move towards building autonomous vehicles that use only camera sensors [38]. Hence reasoning about purely camera-based autonomous vehicles is important.

In [17], the authors consider camera-based autonomous vehicles and go a step beyond testing. They consider the problem of *verifying* the safety of *all* trajectories of (a model of) a camera-based autonomous vehicle from an initial region in a *given* synthetic 3D-scene. Let us call this the "scene safety" problem for autonomous vehicles. The analysis is based on the decomposition of space into image-invariant regions which corresponds to vehicle/camera positions from which the images captured are guaranteed to be identical. Further, by treating the DNN as a black box, the algorithm explores the vehicle's trajectory space, correctly returning a *safe* or an *unsafe* answer. While this approach provides a decidable algorithm for the bounded scene safety problem, in practice, computing the invariant region decomposition for complex scenes turns out to be intractable.

In this paper, we propose an alternative approach to the scene safety verification problem. We introduce the notion of "interval images," and propose a *scalable* abstraction-refinement technique based on this notion. An interval image corresponding to a region of space in a synthetic 3D-scene conservatively represents all the images seen from the region. We should point out here that the notion of interval inputs to DNNs (including images) have been considered before, in the context of reasoning about the robustness of DNNs. However in these applications they represent perturbations of a *single* input or image, while in our setting they are an abstract representation of the set of images seen from a region in a 3D-scene. The challenge in the latter interpretation lies in efficiently computing and refining such abstractions in a precise way.

In conjunction with existing techniques [15], [35], [42] to abstractly interpret a DNN on interval images, we show how interval images can be used to abstractly interpret (and refine when needed) the entire system. These ideas are put together in an abstraction-refinement based verification algorithm for the scene safety problem. The key components of this algorithm are (a) an efficient and reasonably precise technique to compute a sound interval image for a region, generalizing classical CG rendering algorithms, and (b) an efficient technique to refine unsafe abstract paths.

In our experimental evaluation we show that in contrast to the approach in [17], the proposed algorithm is able to both verify and detect unsafe trajectories in complex realistic 3D-scenes containing tens of thousands of edges. Thus, interval image based abstractions provide a new technique to achieve scalable solutions to the scene safety problem.

To summarize, the main contributions of this paper are as follows.

- We introduce a novel interval-based image abstraction technique that over-approximates the set of images from

Habeeb P, and Deepak D'Souza are with the Department of Computer Science And Automation, Indian Institute of Science, Bangalore 560012, Karnataka, India (email: habeebp@iisc.ac.in; deepakd@iisc.ac.in).

Kamal Lodaya was formerly with the The Institute of Mathematical Sciences, Chennai 600113, Tamil Nadu, India (email: kamal@imsc.res.in).

Pavithra Prabhakar is with the Department of Computer Science, Kansas State University, Manhattan, Kansas 66506, USA (e-mail: pprabhakar@ksu.edu).

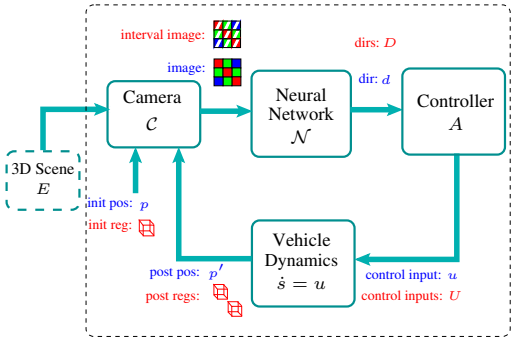
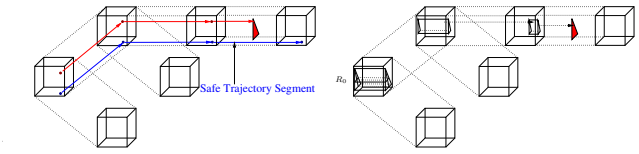


Fig. 1. Camera-Based autonomous system



(a) Trajectories and partial abstraction tree

(b) Spurious collision check

Fig. 2. Overview of our approach

a given region using real camera renderings in 3D environments.

- We develop a safety verification algorithm for camera based linear dynamical systems with neural network controller, which checks safety of all paths from the given initial region to the target region.
- We implemented our approach in a tool and demonstrate the scalability of our approach by applying it to large and complex environments and show that the over-approximation in interval image computation does not significantly impact precision.

II. Overview

In this section we give a high-level overview of the problem and our approach to solving it. We are given a model of a camera-based autonomous vehicle as shown in Fig. 1. The vehicle comprises a single front-facing camera, a neural network that classifies the images captured by the camera, and a controller that converts this classification to a control input to the dynamics of the vehicle. The vehicle runs in a synthetic environment modelled by a CG-style 3D-scene, comprising multiple objects, each represented by a set of triangular facets.

Starting from an initial position p in the environment, the vehicle traces a trajectory as follows. Its camera first captures an image of the environment; this is fed to the neural network which classifies the image and outputs one of a finite set of directions (say go "left", "straight", or "right"). The controller component takes the direction and converts it to a control input u, representing velocities in the x, y, and z-directions, to the vehicle dynamics. The vehicle then moves with this velocity for a fixed sampling period (say 33 ms) to reach a new position p'. This cycle keeps repeating.

We say a trajectory of the vehicle is "safe" if it reaches a specified target region without colliding with an object on its way. Collision with an object occurs if the trajectory intersects one of the triangle faces of the object. Fig. 2(a) depicts two trajectories: a blue one which is safe and a red one which is unsafe. The problem at hand is to check for a given 3D-scene with an initial and target region (specified as cubic regions in the scene), whether all the vehicle's trajectories from the initial region reach the target region safely.

Our proposed approach involves *abstractly* interpreting the system, starting with a *region* of positions. If the blue labels on the edges connecting the components in Fig. 1 represent the *concrete* interpretation of the system, the red labels represent the *abstract* interpretation of the system. We begin by interpreting the camera with the initial region. This gives us a set of possible images taken from different points in the region, which we represent conservatively as an *interval image*. Instead of RGB values associated with pixels in a standard image, an interval image associates *intervals* of RGB values with each pixel. Next the neural network is interpreted with this interval image using existing techniques like [35], [41], to obtain a set of potential directions. Finally, using these directions and the vehicle dynamics, we obtain a set of post regions that overapproximate the set of positions the vehicle can be in after one cycle of the system.

Fig. 2(a) shows a partial "abstraction tree" that we obtain by interpreting the system in this way. The tree is drawn from left to right, with the root at the leftmost end, and the left and right children towards the top and bottom respectively. Each rooted path in the tree, showing successive regions and their sweep along the way, represents (conservatively) trajectories of the system that follow this sequence of directions. If all the leaves of this tree are contained in the target region, and none of its paths intersect a triangle from the scene, we can declare the system to be safe in the given environment. However, if there is path which intersects a triangle, like the top path (shown in red) in Fig. 2(a), it might be spurious in that there is no actual trajectory along that path that intersects the triangle.

To check whether a colliding path is spurious or not, we need to refine the path. We could do this by first "retracting" the triangle along the path to obtain a "source" volume R_0 in the initial cube (see Fig. 2(b)), and then using the technique of [17] to decompose it into "invariant regions" (portions of the volume from where the camera captures *identical* images), and propagating (and in turn decomposing) the regions that follow the path. However this approach fails for complex scenes, due to the prohibitive cost of invariant region decomposition. Instead, we propose a refinement technique based on interval images as follows. We compute the interval image corresponding to R_0 and check the directions given by the neural network on it. If there is no direction along the path, we discard the region (and in this case declare the path to be spurious); if there is only a single direction and this direction is along the path, we compute its post and recursively examine the post region; and if there are multiple directions, one of which is along the path, we subdivide R_0 into smaller regions, and repeat the process on each of them. If the sweep of any of the propagated regions intersects the triangle, we declare the colliding path to

be valid; and if all regions are eventually discarded, we declare the path to be spurious. Fig. 2(b) illustrates this, showing that we find a fragment that reaches the triangle, thereby declaring the collision valid.

III. Preliminaries

We use $\mathbb{R}$ and $\mathbb{R}_{\geq 0}$ to denote the set of reals and non-negative reals with ∞. We use $\mathbb{B}$ to denote the set of byte-sized non-negative integers in the range 0–255. We will be dealing with closed intervals (of reals or integers) of the form $[l, r]$ with $l \leq r$. For intervals $[l, r]$ and $[l', r']$, we define $[l, r] < [l', r']$ iff $r < l'$ (and similarly for $\leq$), and their union $[l, r] \cup [l', r']$ to be the interval $[\min(l, l'), \max(r, r')]$.

For an $m \times n$ matrix M and $a \in [0 \ldots m-1]$ and $b \in [0 \ldots n-1]$, we denote the (a, b)-th element of M by $M(a, b)$. For $m \times n$ matrices M and N, we write $M \leq N$ to denote the fact that $M(a, b) \leq N(a, b)$ for each $a \in [0 \ldots m-1]$ and $b \in [0 \ldots n-1]$.

Given a triangle $t = (v_0, v_1, v_2)$ in two dimensions, with $v_0 = (x_0, y_0)$, $v_1 = (x_1, y_1)$, and $v_2 = (x_2, y_2)$, real-valued attributes r_0, r_1, r_2 at v_0, v_1, v_2 respectively, and a point $p = (x, y)$ within t, the *barycentric interpolation* of the vertex attributes of t at the point p, denoted $ipol(t, r_0, r_1, r_2, p)$, is defined to be $b_0 r_0 + b_1 r_1 + b_2 r_2$, where

$$b_0 = \frac{area(v_1, v_2, p)}{area(v_0, v_1, v_2)}, b_1 = \frac{area(v_0, v_2, p)}{area(v_0, v_1, v_2)},$$

$$b_2 = \frac{area(v_0, v_1, p)}{area(v_0, v_1, v_2)},$$

are the barycentric coordinates of p. The interpolation operation can be seen to be "convex" in the following sense: If d_0, d_1, and d_2 are $\mathbb{R}_{\geq 0}$-valued attributes of the vertices of triangle t, with $d_0 \in [l_0, u_0]$, $d_1 \in [l_1, u_1]$, and $d_2 \in [l_2, u_2]$, then for any point p in t, we have $ipol(t, d_0, d_1, d_2, p) \in [l, u]$, where $l = ipol(t, l_0, l_1, l_2, p)$ and $u = ipol(t, u_0, u_1, u_2, p)$.

IV. Background

We recall some background material and the scene safety problem for a camera-based autonomous vehicle in a synthetic 3D environmment, introduced in [17].

a) 3D-Scenes: We follow standard CG modelling and rendering of 3D-scenes (see e.g. [31]). A 3D-scene is made up of a set of triangles, with each triangle represented by its three vertices in a 3-dimensional *world coordinate system*. The world space is usually represented by a right-handed coordinate system, where the x, y, and z axes are positive in the right, upward, and backward directions respectively. Additionally, each vertex has an associated colour represented by three numbers between 0 to 255, corresponding to components of red, green, and blue colour channels. Formally, a *3D-scene* is a tuple $E = (V, vcol, T)$, where V is a finite non-empty set of vertices in world space, $vcol : V \to \mathbb{B}^3$ is a map that assigns an RGB value to each vertex, and T is a finite non-empty list of triangles built from vertices in V. We assume that $vcol$ induces maps $R_{vcol}, G_{vcol}, B_{vcol} : V \to \mathbb{B}$ representing, respectively, the red, green and blue colours assigned to a vertex.

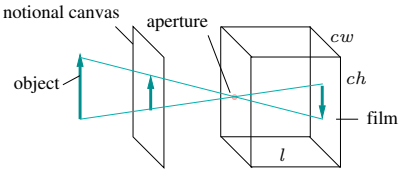
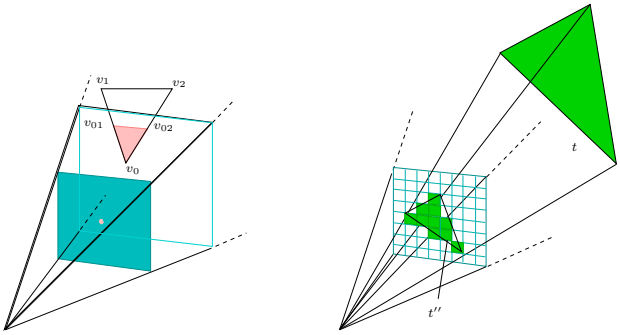


Fig. 3. Camera model



(a) Clipping triangle (v_0, v_1, v_2)

(b) Rendering a triangle on an 8×8 canvas

Fig. 4. Clipping and Rendering

b) Camera Model, Images, and Rendering: We use a simple pinhole-camera based model popularly used in CG, as shown in Fig. 3. A camera model $\mathcal{C}$ is specified by components (l, cw, ch, cwp, chp) where l is the *focal length* of the camera (in m); cw and ch are the *canvas width* and *height*, respectively (in m); and cwp and chp are the canvas *width* and *height* in pixels. We fix a camera model $\mathcal{C} = (l, cw, ch, cwp, chp)$ for the paper.

An image captured by the camera $\mathcal{C}$ is a $chp \times cwp$ grid of pixels with associated RGB values. More precisely, a *$\mathcal{C}$-image* is a triple $I = (R, G, B)$, where R, G and B are $chp \times cwp$ matrices over $\mathbb{B}$. We will denote the components of an image I by R_I, G_I and B_I, respectively.

To describe the way images are rendered, it will be convenient to consider images with depth information that captures the distance to the object contributing to the pixel value. Towards this, we define a *$\mathcal{C}$-depth-image* J to be a tuple (R, G, B, D) where D is an additional $chp \times cwp$ matrix of $\mathbb{R}_{\geq 0}$ values.

Given a 3D-scene E, a camera model $\mathcal{C}$ and a camera position p in world space, we denote the image of E captured by $\mathcal{C}$ from position p, by $img_{\mathcal{C}}(E, p)$. We assume that the camera axis points in a *fixed* direction, namely, the negative z-axis of world space. The image $I = img_{\mathcal{C}}(E, p)$ is obtained as follows:

1) First eliminate all triangles from E that do not intersect with the viewing frustum of the camera. The viewing frustum of the camera is the unbounded rectangular pyramid defined by the camera position p as its origin and the notional canvas of the camera placed at a distance l from p (see Fig. 4(a)).
2) Replace triangles that partially intersect with the viewing frustum, by a new set E' of "clipped" triangles that are fully within the frustum. In Fig. 4(a), the triangle

(v_0, v_1, v_2) is replaced by the triangle (v_0, v_{01}, v_{02}). The colours of the new vertices are obtained by interpolation from the colours of the vertices of the original triangle.

3) For each triangle $t = (v_0, v_1, v_2)$ in E' compute a depth-image J_t as follows:
   - Find the projection t' of t onto the canvas.
   - Consider the "corner-point" triangle $t'' = (v_0'', v_1'', v_2'')$ formed by taking the top-left corners of the pixels containing the vertices of t'.
   - For each pixel (a, b) whose center $c_{a,b}$ falls within the triangle t'':
     - Set $R_{J_t}(a, b) = \mathit{ipol}(t'', R_{vcol}(v_0), R_{vcol}(v_1), R_{vcol}(v_2), c_{a,b})$. And similarly for G and B.
     - Set $D_{J_t}(a, b) = \mathit{ipol}(t'', d_0, d_1, d_2, c_{a,b})$, where d_0, d_1, d_2 are the depths (distances from p along the z-axis) of v_0, v_1, v_2 respectively.

   For pixels (a, b) whose centers lie outside t'', set $R_{J_t}(a, b)$ to be the background colour bg_{red} (and similarly for G and B), and the depth $D_{J_t}(a, b) = \infty$. Fig. 4(b) shows a triangle t, its corner-point projection t'' on the canvas, and the colours assigned to the pixels, assuming the vertices of t are coloured green and the background colour is white.

4) For each pixel (a, b):
   - Find the triangle t such that $D_{J_t}(a, b)$ has the smallest value (break ties in favour of earlier occurrence in the list of triangles T).
   - Set $R_I(a, b) = R_{J_t}(a, b)$ (and similarly for G and B).

c) Camera-based Autonomous Vehicles: Following [17], we model a camera-based autonomous vehicle in a given 3D-scene as a simple closed-loop continuous-time sampled control system (see Fig. 1). At the beginning of a sample period τ, the vehicle, with camera mounted on it in a fixed orientation along the negative z-axis, is at a certain position s in world space. Let $I = \mathit{img}_{\mathcal{C}}(E, s)$. The image is fed to the perception module, a neural network $\mathcal{N}$, which outputs the vehicle direction $dir = f_{\mathcal{N}}(I)$. This is then fed to a controller (represented by a linear transformation A), which provides the control input $u = f_A(dir) = A \cdot dir$ to be fed to the vehicle dynamics $\dot{s} = u$. The vehicle's state at the end of sample period τ is then updated to be $s + \tau \cdot u$. We will represent state update based on the direction due to the control and dynamics by f^A_{VC}, with $f^A_{VC}(s, dir) = s + \tau \cdot f_A(dir)$. Note that the states traversed in that interval $[0, \tau]$ is the convex hull of s and $f^A_{VC}(s, dir)$, which we denote by $\mathit{CHull}(s, f^A_{VC}(s, dir))$.

Definition 1. *A* (camera-based) autonomous vehicle V *of dimension* (k, l) *is a tuple of the form* $V = (\mathcal{C}, \mathcal{N}, A)$ *where*

- $\mathcal{C} = (\mathit{fl}, \mathit{cw}, \mathit{ch}, \mathit{cwp}, \mathit{chp})$ *is a camera model, with* $k = \mathit{cwp} \cdot \mathit{chp} \cdot 3$*,*
- $\mathcal{N}$ *is a neural network with input and output layers of dimension* k *and* l *resp., with* $f_{\mathcal{N}}$ *representing its input-output function,*
- A *is a* $3 \times l$ *matrix which models the controller, with* f^A_{VC} *being the induced map from directions to state updates.*

A *trajectory* of vehicle V in a given 3D-scene E starting from $\mathit{Init} \subseteq \mathbb{R}^3$ is a sequence of states $\sigma = s_0, s_1, \ldots$, where $s_0 \in \mathit{Init}$ and for each $i \in \mathbb{N}$, $s_{i+1} = f^A_{VC}(s_i, f_{\mathcal{N}}(\mathit{img}_{\mathcal{C}}(E, s_i)))$. A trajectory σ of V in a scene E is *safe* w.r.t. a target region $\mathit{Tgt} \subseteq \mathbb{R}^3$ if there exists $i \in \mathbb{N}$ such that $s_i \in \mathit{Tgt}$, and for all $0 \leq j < i$, $\mathit{CHull}(s_j, s_{j+1})$ does not intersect any triangle in E.

For a vehicle V, a convex set of states M, and a direction dir, we define

$$\mathit{Post}(M, V, dir) = \{f^A_{VC}(s, dir) \mid s \in M\} \text{ and}$$
$$\overline{\mathit{Post}}(M, V, dir) = \mathit{CHull}(M, \mathit{Post}(M, V, dir)).$$

d) Invariant Regions: An invariant region in a given 3D-scene, w.r.t. a camera model $\mathcal{C}$ [17], is a 3D-region in world space where images captured by the camera are indistinguishable: i.e. all images from points in this region have *identical* RGB values for each pixel. As shown in [17], invariant regions can be represented as logical constraints on the coordinates of the camera viewing point p.

e) Scene Safety Problem: The *scene safety* verification problem for a camera-based autonomous vehicle in a given 3D-scene, is the following: Given an autonomous vehicle V, a 3D-scene E, an initial region Init, and a target region Tgt; are *all* trajectories of V in E starting from Init safe w.r.t. Tgt? Following [17], we assume that the initial region Init is a convex polyhedral region, the target region Tgt is specified as the region beyond an unbounded plane parallel to the xy-plane, and that in each sample period the vehicle makes a minimum progress in the negative z-direction. We also assume that the neural network $\mathcal{N}$ classifies images into a finite set of directions.

V. Interval Images

In this section we introduce our interval image abstraction. An interval image stores an interval of RGB values for each pixel on the canvas. It thus represents a *set* of images, obtained by essentially taking the cartesian product of the intervals.

More formally, let $\mathcal{C}$ be a camera model. Then a *$\mathcal{C}$-interval image* is a tuple $\mathcal{I} = (\mathcal{R}, \mathcal{G}, \mathcal{B})$ where $\mathcal{R}, \mathcal{G}$ and $\mathcal{B}$ are $\mathit{chp} \times \mathit{cwp}$ matrices whose entries are $\mathbb{B}$-intervals. An interval image $\mathcal{I} = (\mathcal{R}, \mathcal{G}, \mathcal{B})$ represents a set of "concrete" $\mathit{chp} \times \mathit{cwp}$ images $\gamma(\mathcal{I}) = \{I \mid R_I \in \mathcal{R},\ G_I \in \mathcal{G},\ B_I \in \mathcal{B}\}$. Here we use the notation $R_I \in \mathcal{R}$ to mean that for each (a, b), $R_I(a, b) \in \mathcal{R}(a, b)$. Conversely, given a non-empty set X of images, we can define the (canonical) interval image abstraction of X, denoted $\alpha_0(X)$, to be the interval image $\mathcal{I} = (\mathcal{R}, \mathcal{G}, \mathcal{B})$ where for each (a, b), $\mathcal{R}(a, b) = [l, u]$ where $l = \min\{R_I(a, b) \mid I \in X\}$ and $u = \max\{R_I(a, b) \mid I \in X\}$ (and similarly for $\mathcal{G}$ and $\mathcal{B}$). It is easy to see that for any non-empty set of images X, $X \subseteq \gamma(\alpha_0(X))$. We will consider other interval image abstractions α that associate an interval image $\alpha(X)$ with a set of non-empty images X. We say an interval image abstraction α is *sound*, if for every non-empty set of images X, we have $X \subseteq \gamma(\alpha(X))$. The terminology and notations used here are based on theory of abstract interpretation for programs [7]. .

For a non-empty set of interval images X we define their *union* to be the interval image $\mathcal{I}$, where for each pixel (a, b),

$R_{\mathcal{I}}(a,b)$ is obtained by taking the union of the intervals $R_{\mathcal{I}'}(a,b)$ for each $\mathcal{I}' \in X$ (and similarly for $G_{\mathcal{I}}$ and $B_{\mathcal{I}}$).

Once again, it will be useful to define the interval analogue of depth-images, for the purpose of computing interval images. An *interval depth-image* is a tuple $\mathcal{J} = (\mathcal{R}, \mathcal{G}, \mathcal{B}, \mathcal{D})$, where $\mathcal{R}$, $\mathcal{G}$, and $\mathcal{B}$ are as in interval images, and $\mathcal{D}$ is $chp \times cwp$ matrix of $\mathbb{R}_{\geq 0}$ intervals. Such an interval depth-image $\mathcal{J}$ represents a set of depth-images $\gamma(\mathcal{J})$, defined in the expected way, where in particular, for J to be in $\gamma(\mathcal{J})$ we require $D_J(a,b) \in \mathcal{D}(a,b)$ for each pixel (a,b). We define the union of a set of interval depth-images in the expected manner.

A. Computing Interval Images

We now describe our technique to efficiently compute an interval image corresponding to a given convex region *reg* in an environment E. We consider a couple of techniques leading up to our proposed technique. The techniques we use are sound (in that the concretization of the interval image computed contains all the images seen from *reg*), but vary in precision (i.e. how close the concretization is to the actual set of images seen from *reg*).

The first idea is to compute the set X of all possible images from the region *reg*, and then take their canonical interval image abstraction (i.e. $\alpha_o(X)$). While this is the most precise we can get, it is not very scalable. As demonstrated in [17], computing all possible images in a region is infeasible when the number of triangles in the environment is large.

The second idea is to first compute interval images $\mathcal{I}_t$ corresponding to each *triangle* t in E. The interval image $\mathcal{I}_t$ is computed as $\alpha_0(X_t)$ where X_t is the set of images of triangle t seen from *reg*. The set X_t can be computed using invariant regions w.r.t. t using the technique of [17]. From each of these invariant regions, the triangle t has a unique image, which lets us compute X_t. The interval image $\mathcal{I}$ corresponding to the region is now computed by taking the interval union of the interval images $\mathcal{I}_t$ for each triangle t in the scene E.

The interval image computed this way is not very precise, as it takes the union of pixel intervals *without* considering the *depth* of points on the triangle contributing the pixel. In rendering, if two triangles map to the same pixel, the pixel's colour is determined by the depth of the corresponding point on the triangles. The nearest triangle "wins," and only *its* colours contribute to the pixel's colour. A depth-based union provides a more precise interval image that mirrors the classical rendering process.

With this in mind we propose a technique to compute an interval image, as described below. We call the interval image thus computed, $IntervalImage(E, reg)$.

1) For each triangle $t = (v_0, v_1, v_2)$ in E, with $v_i = (x_i, y_i, z_i)$, compute an interval depth-image $\mathcal{J}_t$ as follows. Let $I_1, \ldots, I_k$ be the images of t seen from *reg*, and let $r_1, \ldots, r_k$ be the corresponding invariant regions in *reg*. Let t_l'' be the projected corner-point triangle corresponding to the image I_l of t. In general t_l'' could be a polygon (due to possible clipping), but for simplicity let us call it a triangle. We note that the relative position of t within the camera viewing frustum (i.e. whether it is fully within the frustum or an edge intersects a plane of the frustum), and the projected corner-point triangle, does not change if we move the camera position anywhere within the invariant region r_l.
   a) For each image I_l, $l \in \{1, \ldots, k\}$, compute interval depth-image $\mathcal{J}_l$ as:
      i) If t is fully within the frustum from points in r_l, let
      $$\begin{aligned} l_0 &= \inf\{z_p - z_0 \mid z_p \in r_l\}, \text{ and} \\ u_0 &= \sup\{z_p - z_0 \mid z_p \in r_l\}, \end{aligned}$$
      and similarly define l_1, u_1 corresponding to v_1, and l_2, u_2 corresponding to v_2. For each pixel (a,b) such that $c_{a,b}$ lies within t_l'', set
      $$\begin{aligned} R_{\mathcal{J}_l}(a,b) &= (R_{vcol}(v_0), R_{vcol}(v_0)) \\ &\quad \text{(and similarly for } G \text{ and } B), \\ D_{\mathcal{J}_l}(a,b) &= (l,u), \text{ where} \\ &\quad l = ipol(t, l_0, l_1, l_2, c_{a,b}) \text{ and} \\ &\quad u = ipol(t, u_0, u_1, u_2, c_{a,b}). \end{aligned}$$
      For all other pixels, set $R_{\mathcal{J}_l}(a,b) = [bg_{red}, bg_{red}]$ (and similary for G and B) and $D_{\mathcal{J}_l}(a,b) = [\infty, \infty]$.
      ii) If t is not fully within the frustum, say v_0 is inside while v_1 and v_2 are outside the frustum, as shown in Fig. 4(a). Here $t_l'' = (v_0, v_{01}, v_{02})$ is the projected corner-point triangle corresponding to t. Let V_{01} and V_{02} be the sets of points of intersection of edges (v_0, v_1) and (v_0, v_2) with the top plane of the frustum from different positions in r_l. Let
      $$\begin{aligned} l_{01} &= \inf\{z_p - z_{01} \mid z_p \in r_l, (x_{01}, y_{01}, z_{01}) \in V_{01}\}, \\ u_{01} &= \sup\{z_p - z_{01} \mid z_p \in r_l, (x_{01}, y_{01}, z_{01}) \in V_{01}\}, \end{aligned}$$
      and similarly define l_{02}, u_{02} corresponding to v_{02}, and l_0, u_0 corresponding to v_0. For each pixel (a,b) s.t. $c_{a,b}$ lies within t_l'', set
      $$\begin{aligned} R_{\mathcal{J}_l}(a,b) &= (R_{vcol}(v_0), R_{vcol}(v_0)) \\ &\quad \text{(and similarly for } G \text{ and } B), \\ D_{\mathcal{J}_l}(a,b) &= (l,u), \text{where} \\ &\quad l = ipol(t, l_0, l_{01}, l_{02}, c_{a,b}) \text{ and} \\ &\quad u = ipol(t, u_0, u_{01}, u_{02}, c_{a,b}). \end{aligned}$$
      For all other pixels, set $R_{\mathcal{J}_l}(a,b) = [bg_{red}, bg_{red}]$ (and similary for G and B) and $D_{\mathcal{J}_l}(a,b) = [\infty, \infty]$.
   b) Set $\mathcal{J}_t$ to be the union of the interval depth-images $\mathcal{J}_l$ for $l \in \{1, \ldots, k\}$.
2) Compute the interval image $\mathcal{I}$ by taking the "depth-union" of the depth-interval images $\mathcal{J}_t$, as follows. For each pixel (a,b):
   a) Let $u_{a,b}$ be the *minimum* value of the right interval bound of $D_{\mathcal{J}_t}(a,b)$ over triangles t in E.
   b) Let $S = \{t \in E \mid D_{\mathcal{J}_t}(a,b) = (l,u) \text{ with } l \leq u_{a,b}\}$.
   c) Set $R_{\mathcal{I}}(a,b) = \bigcup_{t \in S} R_{\mathcal{J}_t}(a,b)$ (and similarly for G and B).

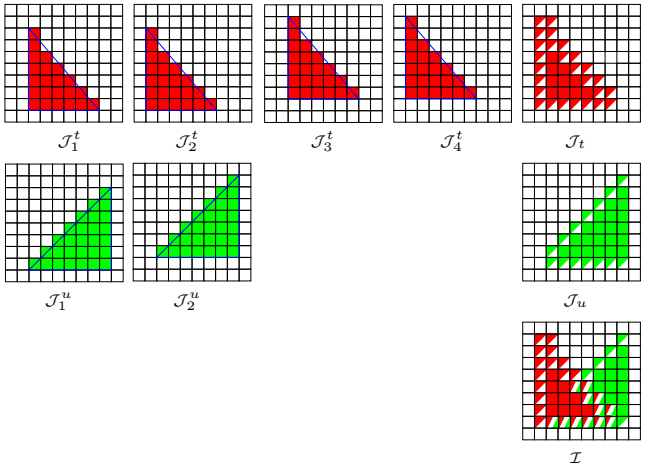
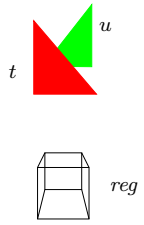


Fig. 5. Illustrating computation of interval image $\mathcal{I}$



To illustrate our interval image computation, consider a 3D-scene comprising just two triangles: a red one t and a green one u; and a region reg from where we need to compute the interval image, as shown in the figure alongside. Let us say our camera has a canvas of 10×10 pixels, with pixels $(0, 0)$ and $(9, 9)$ located in the top-left and bottom-right corners respectively. The idea is basically to compute the interval depth-images for t and u separately, and then take their depth-union.

To compute the interval depth-image for triangle t, we first compute all the images of t that can be seen from reg. Let us say there are four such images, shown in Fig. 5 as $\mathcal{J}_1^t, \mathcal{J}_2^t, \mathcal{J}_3^t$ and $\mathcal{J}_4^t$. This means that reg can be decomposed into four invariant regions r_1, r_2, r_3, and r_4, representing the volumes in reg from where the camera captures exactly these respective images. However, the depth associated with each pixel (say in $\mathcal{J}_1$), which is essentially the distance from the center of the pixel to t in the z-direction, may *vary* depending on different positions in the region r_1. Hence we represent the images as interval depth-images with an interval of depths associated with each pixel. Let us focus on the pixels $(6, 3)$ and $(8, 3)$. Table I shows the values of the different interval depth-images for these two pixels. We assume here that the background colour is white, with RGB value (255,255,255).

We now take the union of these interval depth-images, to obtain the interval depth-image $\mathcal{J}_t$ corresponding to triangle t. The image $\mathcal{J}_t$ is depicted in Fig. 5 by showing each pixel split into multiple colours corresponding to the colour intervals associated with the pixel. Thus, pixel $(6, 3)$ gets the RGB intervals $[255, 255]$, $[0, 0]$, $[0, 0]$ since the pixel is coloured red in all the four images. It also gets the depth-interval $[10, 10.5]$ as the union of the depths associated with the pixel across the four images. One the other hand, the pixel $(8, 3)$ gets the RGB intervals $[255, 255]$, $[0, 255]$, $[0, 255]$, since the pixel takes colour red in some images and white (background) in others. Its depth interval is $[10, \infty]$ for a similar reason. In a similar way, assuming two images $\mathcal{J}_1^u$ and $\mathcal{J}_2^u$ for triangle u, as shown in Fig. 5, we obtain the interval depth-image $\mathcal{J}_u$ for it.

Finally, we take the depth-based union of these two images, to obtain the interval image $\mathcal{I}$ as shown in Fig. 5. If we consider pixel $(6, 3)$, the depth interval from $\mathcal{J}_t$ is $[10, 10.5]$, and the depth interval from $\mathcal{J}_u$ is $[20, \infty]$. Therefore, in the final interval image $\mathcal{I}$, this pixel has only the colour intervals from $\mathcal{J}_t$. On the other hand, for pixel $(8, 3)$, the depth intervals are $[10, \infty]$ and $[20, \infty]$ in $\mathcal{J}_t$ and $\mathcal{J}_u$ respectively. Therefore, in the final interval image this pixel contains the union of the colour intervals from both interval images, displayed as red, white, and green in the figure.

The following claim asserts the soundness of our interval image construction:

Theorem 1. *Procedure IntervalImage is sound (i.e. for every environment E and region reg, if X is the set of images seen from the region reg in the environment E, and $\mathcal{I}$ is the interval image $IntervalImage(E, reg)$, then $X \subseteq \gamma(\mathcal{I})$.*

Proof. Let I be an image seen from a point p in reg, and let r be its invariant region in reg. Consider an arbitrary pixel (a, b). Its colour in I (i.e. $R_I(a, b)$, $G_I(a, b)$ and $B_I(a, b)$) must be contributed by a triangle $t = (v_0, v_1, v_2)$ in E, and its interpolated depth, say d, at $c_{a,b}$ must be the least among all triangles in E.

Subclaim: We now claim that $R_I(a, b) \in R_{\mathcal{J}_t}(a, b)$ (and similarly for B and G), and $d \in D_{\mathcal{J}_t}(a, b)$. For convenience, let us assume t was fully contained in the camera viewing frustum at p. The colour $R_I(a, b)$ would definitely belong to $R_{\mathcal{J}_I}(a, b)$, and hence also to $R_{\mathcal{J}_t}(a, b)$. For the depth claim, let d_0, d_1, d_2 be the depths of the vertices of t from p. Then d_0, d_1, d_2 would belong to the depth intervals computed for v_0, v_1, v_2 respectively. By the convexity property of the interpolation function, it follows that d will belong to the interpolated intervals for (a, b) in $\mathcal{J}_t$. (End of subclaim.)

Now the only way $D_{\mathcal{J}_t}(a, b)$ would not be included in $D_{\mathcal{I}}(a, b)$ is if there was some other triangle t' such that the interval $D_{\mathcal{J}_{t'}}(a, b)$ *ends* before $D_{\mathcal{J}_t}(a, b)$ begins. But triangle t' must have a depth $d' \geq d$ w.r.t. the pixel (a, b) from p, and since $d' \in D_{\mathcal{J}_{t'}}(a, b)$, this cannot happen. This completes the proof of the claim. □

B. Interpretation of DNNs on Interval Images

A neural network can be viewed as a mathematical function that takes an input vector, such as an image, and produces a set of output values. The network learns to compute this function by adjusting the weights and biases of its neurons during the training process. In the context of our system model, the neural network takes an image seen from the vehicle's current position as input, and produces an output direction for the vehicle to follow. Each pixel in the input image is represented by three input nodes in the neural network, one for each colour channel (red, green, and blue). The output layer of the network produces a single value for each output node. Each output node corresponds to a particular direction. The output node with the maximum value is taken to be the output direction.

In our setting, we aim to interpret a neural network on an interval image. An interval image represents all possible

TABLE I
VALUES FOR PIXELS $(6, 3)$ AND $(8, 3)$ IN EXAMPLE INTERVAL IMAGES

(6,3)	$\mathcal{J}_1^t$	$\mathcal{J}_2^t$	$\mathcal{J}_3^t$	$\mathcal{J}_4^t$	$\mathcal{J}_t$	$\mathcal{J}_1^u$	$\mathcal{J}_2^u$	$\mathcal{J}_u$	$\mathcal{I}$
R	[255,255]	[255,255]	[255,255]	[255,255]	[255,255]	[255,255]	[0,0]	[0,255]	[255,255]
G	[0,0]	[0,0]	[0,0]	[0,0]	[0,0]	[255,255]	[255,255]	[255,255]	[0,0]
B	[0,0]	[0,0]	[0,0]	[0,0]	[0,0]	[255,255]	[0,0]	[0,255]	[0,0]
D	[10,10.5]	[10,10.3]	[10,10.5]	[10,10.3]	[10,10.5]	$[\infty, \infty]$	[20,20.5]	$[20,\infty]$	
(8,3)	$\mathcal{J}_1^t$	$\mathcal{J}_2^t$	$\mathcal{J}_3^t$	$\mathcal{J}_4^t$	$\mathcal{J}_t$	$\mathcal{J}_1^u$	$\mathcal{J}_2^u$	$\mathcal{J}_u$	$\mathcal{I}$
R	[255,255]	[255,255]	[255,255]	[255,255]	[255,255]	[0,0]	[255,255]	[0,255]	[0,255]
G	[0,0]	[0,0]	[255,255]	[255,255]	[0,255]	[255,255]	[255,255]	[255,255]	[0,255]
B	[0,0]	[0,0]	[255,255]	[255,255]	[0,255]	[0,0]	[255,255]	[0,255]	[0,255]
D	[10,10.5]	[10,10.3]	$[\infty, \infty]$	$[\infty, \infty]$	$[10,\infty]$	[20,20.5]	$[\infty, \infty]$	$[20,\infty]$	

images for a given region. Our objective is to obtain all the neural network outputs that might arise from a concrete input image induced by the interval image. To achieve this, we utilize the alpha-beta-CROWN tool [42], [41], which is a neural network verifier based on an efficient linear bound propagation framework and branch and bound techniques. This tool is a complete verifier for neural networks, providing "Yes" or "No" answers for whether there exists an input contained in the input interval on which the output nodes of the given neural network satisfy a given relation. In our setting, we query the tool for each possible output of the neural network to determine if there exists any input in the given input interval image that could produce the specific output. The collection of all such outputs constitutes the set of outputs of the given input interval image.

In the next section we show how to put together interval images and the abstract interpretation of a DNN on them, towards a solution to the scene safety problem.

VI. ABSTRACTION-REFINEMENT ALGORITHM

This section presents our decision procedure based on interval images for the scene safety problem for an autonomous vehicle V operating in a 3D-scene E, with initial region $Init$ and target region Tgt.

We first compute an interval image of the initial region using our interval image computation technique explained in the previous section. Next, we interpret the neural network on this interval image using alpha-beta-CROWN, to obtain the possible outputs for the interval image. We then use the vehicle dynamics to compute the post region based on the neural network outputs. We propagate a region fully in each direction provided by the neural network and check for collisions along the path from the region to the post region. If there is no collision and the post region is not fully in the target region, the process repeats. In case of a collision, we check whether the collision is valid or not using our collision refinement procedure, as defined in algorithm 2. If the collision is valid, we stop and return "Unsafe". If the algorithm has no more regions remaining to explore, it stops and returns "Safe".

The algorithm $CheckSafetyII$ given in Algorithm 1 outlines the safety check procedure. It takes as input the vehicle dynamics V, an environment E, an initial region $Init$, and a target region Tgt, returning "Safe" if all trajectories of V starting from $Init$ in E are safe, and "Unsafe" otherwise. The algorithm essentially builds an exploration tree $\mathcal{T}$ in an incremental manner. It begins by initializing the tree with a single node containing the initial region $Init$. It then executes

Algorithm 1 $CheckSafetyII(V, E, Init, Tgt)$

Require: Vehicle V, environment E, initial region $Init$, target region Tgt.
Ensure: Returns "Safe" iff all trajectories of V in E are safe, else "Unsafe"
1: Initialize tree $\mathcal{T}$ with root node containing $Init$.
2: **while** Exists a leaf node M in $\mathcal{T}$ which is not contained in Tgt **do**
3: $\quad\mathcal{I} = IntervalImage(E, M)$
4: $\quad\mathcal{D} =$ alpha-beta-CROWN$(\mathcal{I})$
5: $\quad$**for** each $dir \in \mathcal{D}$ **do**
6: $\quad\quad$Let $N = Post(M, V, dir)$
7: $\quad\quad$Add node N and edge (M, dir, N) to $\mathcal{T}$
8: $\quad$**for** each $dir \in \mathcal{D}$ **do**
9: $\quad\quad$Let $N = Post(M, V, dir)$
10: $\quad\quad$**for** each triangle $t \in E$ **do**
11: $\quad\quad\quad$**if** $CheckCollision(M, N, t)$ **then**
12: $\quad\quad\quad\quad R = \overline{Post}(t, V, dir) \cap N$
13: $\quad\quad\quad\quad$**if** $CheckReach(\mathcal{T}, R, N)$ **then return** "Unsafe"
14: **return** "Safe"

lines 2 to 13 continuously until there is no leaf node in $\mathcal{T}$ that is not contained in Tgt.

In each step the algorithm takes a leaf node M from $\mathcal{T}$ which is not contained in Tgt and computes its interval image $\mathcal{I}$ using the $IntervalImage(E, M)$ procedure. It then applies the alpha-beta-CROWN tool on $\mathcal{I}$ to obtain the set of possible output directions $\mathcal{D}$ for the vehicle. Next, for each direction $dir \in \mathcal{D}$, the algorithm computes the post region N of M using the function $Post(M, V, dir)$. This function computes the post region N of M by transforming M in the direction dir, based on the vehicle V's dynamics. The newly computed region N is added to $\mathcal{T}$, along with an edge from M to N in direction dir. Then, for each triangle $t \in E$, we check for collisions with the path segment M, N using the function $CheckCollision(M, N, t)$. This function computes a convex hull of the regions M and N, and checks for intersection with t.

If the collision check returns true for a particular triangle t, it could be spurious on account of overapproximations due to both the interval image computation and the abstract interpretation of the neural network. Therefore, we need to check whether this is a valid collision or not. We initially compute the set of points R in N that can be reached from $t \cap CHull(M, N)$ by following the direction dir by intersecting $\overline{Post}(t, V, dir)$ with N. Then algorithm calls the function $CheckReach(\mathcal{T}, R, N)$, described in Algorithm 2, to verify the validity of the collision. If $CheckReach$ returns true, the algorithm returns "Unsafe".

The algorithm returns "Safe" if all collisions (if any) are found to be spurious.

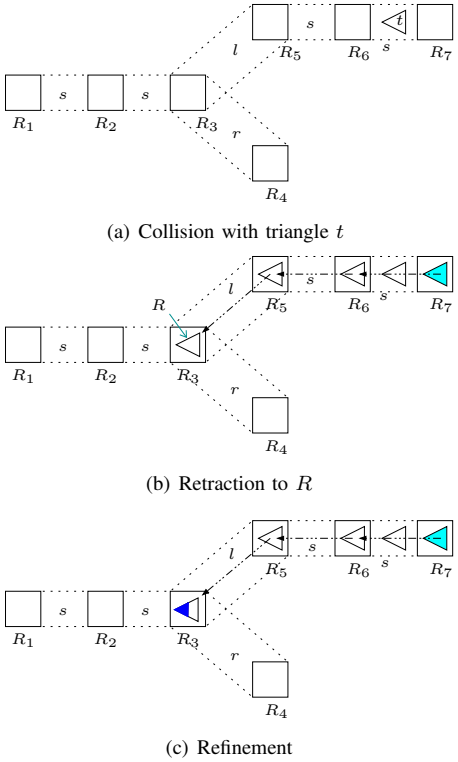
The *CheckReach* procedure (see Algorithm 2) utilizes a helper function *Decompose*. Given a region R and a direction dir, the *Decompose* procedure computes a decomposition of R into sets of disjoint subregions $\mathcal{U}$ and $\mathcal{V}$, such that $\mathcal{U}$ has regions R' for which the abstract-DNN output is exactly $\{dir\}$, and $\mathcal{V}$ has regions R'' for which the abstract-DNN output does not contain dir. One way to compute such a $(\mathcal{U}, \mathcal{V})$ is to partition R into invariant regions using the technique in [17], and collect invariant regions whose corresponding images have DNN output d in $\mathcal{U}$, and the rest into $\mathcal{V}$. However this approach suffers from scalability issues. Instead, we first try to refine R and obtain a decomposition using interval images and the abstract interpretation of the DNN, as follows. We first compute an axis-parallel cube-hull of R, dividing it into n^3 uniform cubes (for a suitable choice of n), and retain the non-empty intersections of the cubes with R. Then compute the interval images from these regions and compute the abstract-DNN outputs on the images. Collect all the regions with abstract-DNN outputs $\{dir\}$ into $\mathcal{U}$, discard those regions with abstract-DNN outputs disjoint from $\{dir\}$, and keep the remaining into another set of regions $\mathcal{V}$. Now repeat the process with each region in $\mathcal{V}$, collecting those with output direction dir into $\mathcal{U}$ and discarding those disjoint from direction dir. After k iterations (for a suitable k) if $\mathcal{V}$ is still non-empty, we fall back on the invariant region decomposition of the regions in $\mathcal{V}$ to collect those with direction $\{dir\}$ into $\mathcal{U}$.

Algorithm 2 *CheckReach*$(\mathcal{T}, R, N)$

Require: Partial Abstract Tree $\mathcal{T}$, region R in node N of $\mathcal{T}$.
Ensure: Returns *True* iff R is reachable by a concrete execution along path to N.

```
Let p be the path from root to N in T.
if (No branching ancestor of N in T) then return True
Let M be youngest branching ancestor of N in T.
Let R' be the retraction of R along p to M, and dir be direction from M along p.
Let I = IntervalImage(E, R) and D = alpha-beta-CROWN(I)
if (D = {dir}) then
    U = {R'}
else
    U = Decompose(R', dir)
if U is empty then
    return False
else
    return ⋁_{R''∈U} CheckReach(T, R'', M)
```

We now explain the main steps of our algorithm via a schematic example illustrated in Fig. 6. Let our initial region be R_1. Let us say the abstract-DNN output for the interval image for region R_1 is determined to be "straight". We compute the post region corresponding to R_1 in the direction "straight" as R_2. Similarly, the abstract-DNN output for region R_2 is also found to be "straight", leading to the region R_3. Let us say region R_3 has two abstract-DNN outputs, namely "right" and "left", with corresponding post regions being R_4 and R_5. Let us say we prioritize region R_5 for further exploration, reaching the region R_6 with a "straight" abstract-DNN output. Now a collision with a triangle is detected in the path segment R_6-R_7, as depicted in Fig. 6(a).



(a) Collision with triangle t

(b) Retraction to R

(c) Refinement

Fig. 6. Illustrating the *CheckReach* Procedure.

To determine the validity of the collision, we first project the obstacle triangle onto R_7, represented by the cyan-colored triangular region in Fig. 6(b), and call *CheckReach* on it. The procedure first checks whether R_7 has a branching ancestor. If it doesn't, *CheckReach* returns *true*, and we declare the collision along the path segment R_6-R_7 as valid because the collided object is reachable from the initial region. But since R_7 does have a branching ancestor (namely R_3), we retract the collision region from R_7 up to R_3. Let us call the retracted region R, as shown in Fig. 6(b). Next we compute an interval image $\mathcal{I}$ for R, and let us say the abstract-DNN outputs on $\mathcal{I}$ are "left" and "right". Then we call $Decompose(R, left)$ to decompose the region R along the "left" direction. Let us say the decomposition gives us a single region, shown coloured blue in Fig. 6(c). We recursively check reachability of this blue region using *CheckReach*. However, now since the region has no branching ancestors in the tree, *CheckReach* returns *true*, and we declare the collision as valid, and the algorithm returns "Unsafe".

We can now state:

Theorem 2. *The CheckSafetyII algorithm is sound and complete: it returns "Safe" iff all trajectories of V in E starting from Init are safe.*

Proof. Let $\mathcal{T}'$ be the complete "abstraction tree" corresponding to $V, E, Init, Tgt$ and the sound DNN abstract interpreter alpha-beta-CROWN, whose set of nodes correspond to the

set of regions obtained by starting with the initial region $Init$ and taking its closure under 1-step posts for each of the directions given by alpha-beta-CROWN on the interval images corresponding to each region. A region R is added to the tree only if its parent is *not* fully contained in the target region Tgt. An edge (M, N) in $\mathcal{T}'$ is labelled by a direction d if N was obtained by taking the post of M in the direction d.

Given the assumptions of the problem, $\mathcal{T}'$ must be a finite tree. Further, every execution π of V in E, must lie along a path p in $\mathcal{T}'$ (given by the sequence of directions along π). This follows by the soundness of both the interval image computation and alpha-beta-CROWN. The $CheckSafetyII$ algo essentially explores this tree $\mathcal{T}'$, looking for collisions with obstacles along each edge.

We first assume the termination and correctness of the subroutine $CheckReach$, and argue termination and correctness of $CheckSafetyII$. Termination of $CheckSafetyII$ immediately follows from the fact that $\mathcal{T}'$ is finite and that $CheckReach$ always terminates.

For correctness in one direction, suppose the given system has an unsafe trajectory π, and t is the first triangle in E that it collides with. Let p be the $\mathcal{T}'$-path corresponding to π. Then π must lie along this path in $\mathcal{T}'$, with the collision with t happening from node M to N along direction d. Then the algo will eventually check for collisions from M to N with t, and take its projection R to N, and check for reachability of R using $CheckReach$. Since R is reachable (via a execution that extends π), $CheckReach$ must return $True$, and our algo will return "Unsafe".

Conversely, if the algo returns "Unsafe", it must have encountered such nodes M and N, a direction d, a triangle t in E with projection R in N, and $CheckReach$ says that R is reachable along the path to N. Again, by correctness of $CheckReach$, there must be an execution (along p) that reaches R, and hence must collide with t.

We now argue the correctness of $CheckReach$. We assume the termination and correctness of the procedure $Decompose$, which follows from the correctness of our interval image computation (Thm. 1), the DNN abstract interpreter, and the invariant region computation of [17]. Suppose $CheckReach$ is called on a partial abstract tree $\mathcal{T}$ and a region R in a node N of $\mathcal{T}$. Let p be the sequence of directions along the root to N path in $\mathcal{T}$. We argue by induction on the depth of N in $\mathcal{T}$, that the procedure terminates and outputs $True$ iff R is reachable via an execution along p. For the base case, N is the root node (whose associated region is the initial region I). In this case $CheckReach$ will return $True$ since N has no branching ancestors in $\mathcal{T}$. This is correct since the whole of N (and hence R) is indeed reachable.

For the inductive step let N be at a level $k+1$ from the root. The procedure clearly terminates on a call at this level, by the termination of $Decompose$ and the inductive assumption. Let us say R is reachable along p via an execution π. If N has no branching ancestors the algorithm will return $True$ and we are done. Else, let M be the youngest branching ancestor of N, and let $\mathcal{U}$ be returned by $Decompose$ on the retraction R' of R to M. Then clearly there must be a region S in $\mathcal{U}$ which is reachable (via π). Since S is in node M which is at a level k or less in $\mathcal{T}$, by the induction hypothesis $CheckReach(\mathcal{T}, S, M)$ must return $True$. Hence the call to $CheckReach(\mathcal{T}, R, N)$ will also return $True$, and we are done. Conversely, suppose the procedure returns $True$ when called with $\mathcal{T}$, R and N. Then either N has no branching ancestors (in which case R is indeed reachable), or it has a youngest branching ancestor M with a reachable (by induction) region S belonging to $\mathcal{U}$ in the decomposition of the retraction R' of R to M. But this implies that R is also reachable (since all points in S reach R). □

VII. Implementation and Experimental Evaluation

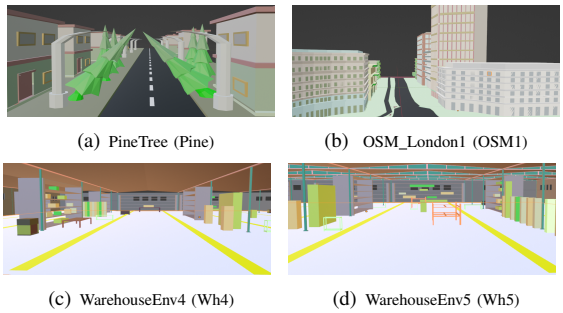
We have implemented our safety checking algorithm in a tool called CamVerif. The tool takes as input the components of an autonomous vehicle (with the neural network in Open Neural Network eXchange format), a 3D-scene in Universal Scene Description (USD) format, and the initial and target regions as linear constraints on the x, y, z coordinates. We use the Z3 solver [28] for partitioning a region into invariant regions, and to find solutions to constraints and check satisfiability. The Parma Polyhedral Library (PPL) [3] is used to generate and processes constraints representing invariant regions. We use the alpha-beta-CROWN tool [42], [41] to interpret the neural network on interval images. The image rendering module is written in C++, while the rest of the implementation is in Python. The code, including rendering and interval image computation for a given region in the environment, is available at https://github.com/camverif1/camverif_0.001.

We use two case studies involving purely camera-based controllers, and environments generated using Blender [36]. The first is a road-following quadcopter [30], [17]. The quadcopter's camera has a focal length of 35 mm, a canvas width and height of 0.9872 in and 0.735 in, respectively, and pixel dimensions of 49×49. The neural network comprises six layers: three convolutional layers, two fully connected layers, and an argmax layer.

The second case study is a line-following warehouse robot that navigates by tracking a yellow marking line on the floor. Unlike the quadcopter, this vehicle is meant to run in a *fixed* premises, though minor variations in terms of placement of objects are possible. We designed a synthetic 3D-model of a warehouse and trained a neural network controller on manually labelled CG images in this scene. The neural network had five layers: two convolutional layers, two fully connected layers, and an argmax layer. The camera model and vehicle dynamics were similar to the quadcopter system.

We created several realistic 3D-environments as shown in Fig. 7(a) and Fig. 7(b), each containing thousands of edges. For example, we used Blender-OSM to create the 'OSM_London1' environment shown in Fig.7(b), obtained from OpenStreetMap. Factory floor environments are depicted in Fig. 7(c) and Fig. 7(d), showing two tracks for to-and-fro movement.

We compared the performance of our tool with that of [17] on all these environments, on a machine with an Intel(R) Core(TM) i7-8700 3.20 GHz CPU and 64GB RAM. Table III shows the bottom left corners of the initial regions. The initial



(a) PineTree (Pine) (b) OSM_London1 (OSM1)

(c) WarehouseEnv4 (Wh4) (d) WarehouseEnv5 (Wh5)

Fig. 7. Images from Different Environments.

region I_1^* is a 3cm cube. The results are summarised in Table II. The first column in the table is the name of the environment. The '#Edge' column indicates the number of edges in the environment. The column titled 'I' gives the initial regions used. The distance to the target region from the initial region in meters is given in the 'Tgt' column. The 'S?' column indicates whether the environment is safe (S) or unsafe (U) with respect to the given initial region and the target region. A '-' in this column indicates that we ran these experiments for 20 hours without finding a valid collision nor proving safety. The time taken by AirVerif[17] and our tool CAMVERIF is shown in the next two columns titled 'Time'. An 'X' in a cell in this column indicates that the tool hangs (usually because of a solver call that does not return) on this input configuration. The column titled '#SC' gives the number of spurious collisions detected and the '#Ref' column gives the number of times we refined the region before concluding that the collision was spurious, respectively. The column titled '#N' gives the number of nodes in the abstract tree, which essentially indicates the number of times we computed the interval image in the granularity of the initial region. The '#P' column indicates how many child nodes of the above tree we were able to prune out (in our case study, each node can have three child nodes) with the use of interval images.

Some immediate take-aways from the table are the following. Of the 15 environments in which both approaches terminate, AirVerif takes significantly less time than CAMVERIF in 5, CAMVERIF does significantly better in another 4, while in the remaining 6 the peformances of the two approaches are comparable (within 20% of each other). Secondly, AirVerif does not terminate on 11 of the larger environments (700+ edges), while CAMVERIF completes its verification run on all of them. For some of the larger environments, CAMVERIF does take substantial time, but notably never gets stuck. Given enough time, CAMVERIF will be able to prove the safety of all environments considered in the experiments.

We present the following insights which explain these results. The AirVerif algorithm essentially relies on a decomposition of each reachable region into invariant regions to compute successors of a region in the tree. On the other hand, CAMVERIF uses the interval image corresponding to a region to compute its successors in the tree. Our first observation is that, in general, computing the invariant region decomposition for a region is far more expensive than computing an interval image for it. As a case in point, AirVerif took more than 9h to generate a partial set of 22 invariant regions for the initial region of the Buildings ('Build') environment, while CAMVERIF generated the interval image for the initial region in 3m and completed the entire analysis in 48m 22s. This expense impacts the exploration of the tree carried out by AirVerif, causing it to time-out on large environments. Secondly, while the invariant region based tree is very precise (fewer false positives), a coarser granularity than invariant regions may often suffice. In the example environment above, CAMVERIF was able to prove safety with the initial granularity of 1cm^3 itself. CAMVERIF refines the granularity used on a need basis, refining only to check spuriousness of a collision. This explains the advantage that CAMVERIF has in environments like Buildings ('Build'). Finally, the two approaches use different abstraction techniques: AirVerif uses a convex-hull based abstraction, while CAMVERIF uses a cubic region abstraction. This may explain the incomparable results on smaller environments.

We can quantify the precision of the interval image abstraction used as follows. We can see that the interval image abstraction is fairly precise, in that it leads to several (immediate) directions being pruned from the top-level abstract tree. For example, in 'env1' the algorithm explored 12 nodes, and of the 36 possible directions (recall that in our case studies each node has 3 possible directions/children), 24 were pruned away despite the interval abstraction. Averaging across all the environments considered, this gain was a significant 35.63%. In most cases, the initial level of abstraction was sufficient to decide the safety. However in some of the environments (particularly where the target distance was high) our abstraction needed to be refined to eliminate spurious collisions. The '#Ref' column gives the number of pieces we needed to refine the initial region into, to be able to rule out spurious collisions (or find actual collisions). Recall that once a collision is detected, we project the collision region and generate the interval image for that region; for many of the unsafe environments, this was the only refinement that took place. For the safe environments, the number of pieces is as much as 240 (for 'env2' with I_1 as initial region). This illustrates that in some cases we need to refine down to this level to overcome the imprecision in the initial abstraction. We note that this refinement takes place purely on a need basis. Overall, the algorithm's performance is influenced by several factors, including the number of edges in the environment, the size of the initial region, the need for refinement, the number of nodes in the abstract tree, and the number of child nodes.

VIII. RELATED WORK

We focus on work related to testing and verification of environment-closed camera/lidar-based NN-controlled systems.

Testing and Simulation. Several works consider the problem of testing or simulation-based analysis of camera-based systems in a given 3D-scene, for properties like temporal logic and STL based specifications [29], [40], [10], or fuzz testing [10]. These works use custom-made scene description languages (like SDL) or generic scene-modelling tools like Scenic [13], and analysis tools like S-TaLiRo [2] and dReach [24]. Simulation

TABLE II
EXPERIMENTAL RESULTS

Env	#Edge	I	Tgt (m)	S?	AIRVERIF Time (s)	CAMVERIF Time (s)	#SC	#Ref	#N	#P
env1	36	I_1	10.5	S	178	204	0	0	12	24
env2	39	I_1	10.5	S	126	11340	3	240	84	124
env2	39	I_1^*	10.5	U	X	3780	1	10	391	161
env2	39	I_2	10.5	U	50	72	0	2	3	5
env3	39	I_1	10.5	S	95	218	0	0	12	24
env4	66	I_1	10.5	S	240	287	0	0	12	24
env5	186	I_1	10.5	S	1096	527	0	0	12	24
env6	336	I_1	10.5	S	1038	721	0	0	12	24
env7	636	I_1	10.5	S	1658	17100	0	0	145	121
env8	786	I_1	10.5	S	X	4440	0	0	153	126
env9	100	I_1	10.5	S	225	362	0	0	12	24
env12	313	I_1	10.5	S	2409	7024	0	0	548	220
env13	313	I_1	10.5	U	32	58	0	2	3	6
env14	51	I_1	10.5	U	50	59	0	2	3	6
env15	57	I_1	10.5	U	57	59	0	2	3	6
env16	78	I_1	10.5	U	63	84	0	2	3	6
env17	171	I_1	10.5	U	170	109	0	2	3	6
Build	5187	I_1	6.5	S	X	2902	0	0	12	23
Street	7779	I_3	6.5	S	X	11100	0	0	44	53
Trees	9987	I_3	6.5	S	X	2334	0	0	9	17
Pine	11091	I_3	6.5	S	X	3840	0	0	9	18
OSM1	68400	I_4	10.5	S	X	296700	0	0	403	0
OSM2	95850	I_4	10.5	S	X	275100	0	0	397	5
Wh1	8844	I_5	10.5	S	X	4260	0	0	73	45
Wh2	8334	I_5	10.5	S	X	35580	0	0	185	74
Wh3	23556	I_5	10.5	S	X	60120	0	0	214	90
Wh4	29496	I_5	10.5	S	X	77940	0	0	216	84
Wh5	50376	I_5	10.5	S	X	128220	0	0	218	81
Street	7779	I_3	25.5	-	X	72000	0	0	550	221
Trees	9987	I_3	25.5	-	X	72000	4	275	198	15
Pine	11091	I_3	25.5	-	X	72000	2	513	117	4
Wh4	29496	I_6	25.5	-	X	72000	2	77	137	64
Wh5	50376	I_6	25.5	-	X	72000	2	30	140	61

TABLE III
INITIAL REGIONS

Init	Bot Left Corner
I_1	0.1, 4.45, 194.5
I_2	-0.95, 4.45, 194.5
I_3	0.1, 4.45, 175.5
I_4	0.1, 4.5, 121.5
I_5	167, 1.4, 194.5
I_6	130, 1.4, 71.5

tools like AirSim [33] and LGSVL [25] simulate the flight of autonomous drones and other vehicles in a given synthetic 3D-scene. VIVAS [16] is a framework utilizing model checking techniques to generate diverse driving scenarios on an abstract model representing the behavior of an autonomous driving system. These scenarios are designed to cover a given set of criteria related to the system's functionality and potential failure modes. The generated scenarios are analyzed through simulation to identify failures. Paracosm [26] systematically generates test scenarios for autonomous driving simulations, allowing users to programmatically define scenarios with road layouts, weather, and dynamic traffic behavior. The test scenario generator maximizes coverage of various behaviors for finding problematic cases. All these tools and frameworks are good for visualizing system behaviour and generating systematic test cases for analysis. However, they are not capable of providing any verification guarantees.

Verification of lidar-based systems. Sun et al [37] consider the problem of verifying the safe trajectory of a lidar-based robot in a given 2D-environment. They use an abstraction-refinement based approach to prove safety. Ivanov et al [20] consider a lidar-based NN-controlled vehicle in a 2D race track. They propose a compositional technique by verifying each segment of the track separately, using hybrid automata models and the Verisig tool [21] to verify safety. Both these works consider only 2D-scenes and do not model camera sensors.

Verification of camera-based systems. Several works abstract the camera component by using, for example, Generative Adversarial Networks (GANs) that are trained to generate images based on the position of the vehicle [23], or "approximate" [18] and probabilistic (via a confusion matrix) [32] abstractions in place of the camera+NN perception component. None of these techniques are able to give exact guarantees about the original camera-based system. In [8], [9] the authors propose a framework to verify a camera-based NN-controlled autonomous landing system, by discretizing the state-space. In contrast to our technique, they use a simplified camera model with only black and white pixels, use a simple geometric-shaped model of the runway, and carry out only conservative verification (an unsafe verdict does not necessarily mean the system is unsafe). Finally, the work in [17] is closely related and gives a decision similar to ours by carrying out an exact exploration of the state space based on image-invariant regions. In contrast, our approach is based on abstraction-refinement using interval images, leverages abstract interpretation of the NN in a white-box manner, and scales to more complex 3D-scenes.

Several other works focus on testing or verifying closed-loop behaviors of dynamical systems with neural network controllers, excluding the perception component. These include tools such as CORA [1], JuliaReach [5], and NNV [39], which competed in the AINNCS category of the ARCH-COMP competition [27]. Additionally, there are tools like α-β-CROWN [42], [41], NNV [39], nnenum [4], Marabou [22], NeuralSAT [11], and FastBATLLNN [12], which focus on open-loop specification of neural networks and participated in the VNN-COMP [6].

IX. CONCLUSION

We have presented an abstraction-refinement technique to solve the reach-avoid problem of camera-based autonomous systems. The technique is based on the notion of interval images which yields a new computational technique to attack the problem and appears to be effective in practice. Some interesting future directions include incorporating the orientation of the vehicle in the interval image computation, allowing dynamically evolving scenes, and combining multiple sensor inputs.

REFERENCES

[1] Althoff, M.: An introduction to CORA 2015. In: Frehse, G., Althoff, M. (eds.) 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems, ARCH@CPSWeek 2014, Berlin, Germany, April 14, 2014 / ARCH@CPSWeek 2015, Seattle, WA, USA, April 13, 2015. EPiC Series in Computing, vol. 34, pp. 120–151. EasyChair (2015). https://doi.org/10.29007/ZBKV, https://doi.org/10.29007/zbkv

[2] Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems. In: Proc. 17th Intl. Conf. Tools and Alg. Constr. Anal. Systems (TACAS 2011), Saarbrücken. pp. 254–257. Springer (2011)

[3] Bagnara, R., Hill, P.M., Zaffanella, E.: The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. Sci. Comput. Program. **72**(1–2), 3–21 (2008)

[4] Bak, S.: nnenum: Verification of relu neural networks with optimized abstraction refinement. In: NASA formal methods symposium. pp. 19–36. Springer (2021)

[5] Bogomolov, S., Forets, M., Frehse, G., Potomkin, K., Schilling, C.: Juliareach: a toolbox for set-based reachability. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control. pp. 39–44 (2019)

[6] Brix, C., Bak, S., Liu, C., Johnson, T.T.: The fourth international verification of neural networks competition (vnn-comp 2023): Summary and results. arXiv preprint arXiv:2312.16760 (2023)

[7] Cousot, P., Cousot, R.: Abstract interpretation and application to logic programs. J. Log. Program. **13**(2&3), 103–179 (1992). https://doi.org/10.1016/0743-1066(92)90030-7, https://doi.org/10.1016/0743-1066(92)90030-7

[8] Cruz, U.S., Shoukry, Y.: NNLander-VeriF: A Neural Network Formal Verification Framework for Vision-Based Autonomous Aircraft Landing. In: Proc. Intl. Symp. NASA Formal Methods (NFM 2022), Pasadena, USA, 2022. LNCS, vol. 13260, pp. 213–230. Springer (2022)

[9] Cruz, U.S., Shoukry, Y.: Certified vision-based state estimation for autonomous landing systems using reachability analysis. CoRR **abs/2309.05167** (2023)

[10] Dreossi, T., Fremont, D.J., Ghosh, S., Kim, E., Ravanbakhsh, H., Vazquez-Chanlatte, M., Seshia, S.A.: VerifAI: A Toolkit for the Formal Design and Analysis of Artificial Intelligence-Based Systems. In: Proc. 31st Intl. Conf. on Computer Aided Verification (CAV 2019), New York City, USA. pp. 432–442 (2019)

[11] Duong, H., Nguyen, T., Dwyer, M.: A dpll (t) framework for verifying deep neural networks. arXiv preprint arXiv:2307.10266 (2023)

[12] Ferlez, J., Khedr, H., Shoukry, Y.: Fast batllnn: fast box analysis of two-level lattice neural networks. In: Proceedings of the 25th ACM International Conference on Hybrid Systems: Computation and Control. pp. 1–11 (2022)

[13] Fremont, D.J., Dreossi, T., Ghosh, S., Yue, X., Sangiovanni-Vincentelli, A.L., Seshia, S.A.: Scenic: a language for scenario specification and scene generation. In: Proc. 40th Conf. Programming Language Design and Implementation (PLDI 2019), Phoenix, USA, 2019. pp. 63–78. ACM (2019)

[14] Fremont, D.J., Kim, E., Pant, Y.V., Seshia, S.A., Acharya, A., Bruso, X., Wells, P., Lemke, S., Lu, Q., Mehta, S.: Formal scenario-based testing of autonomous vehicles: From simulation to the real world. In: 23rd Intl. Conf. Intelligent Transportation Systems (ITSC 2020), Rhodes, Greece. pp. 1–8. IEEE (2020)

[15] Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.T.: AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In: Proc. IEEE Symp. Security and Privacy (SP 2018), San Francisco, USA. pp. 3–18 (2018)

[16] Goyal, S., Griggio, A., Kimblad, J., Tonetta, S.: Automatic Generation of Scenarios for System-level Simulation-based Verification of Autonomous Driving Systems. In: Proc. 5th Intl. Workshop on Formal Methods for Autonomous Systems (FMAS@iFM 2023), Leiden, The Netherlands. EPTCS, vol. 395, pp. 113–129 (2023)

[17] Habeeb, P., Deka, N., D'Souza, D., Lodaya, K., Prabhakar, P.: Verification of camera-based autonomous systems. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems **42**(10), 3450–3463 (2023)

[18] Hsieh, C., Li, Y., Sun, D., Joshi, K., Misailovic, S., Mitra, S.: Verifying controllers with vision-based perception using safe approximate abstractions. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems **41**(11), 4205–4216 (2022)

[19] Ivanov, R., Carpenter, T.J., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Case study: verifying the safety of an autonomous racing car with a neural network controller. In: Proc. 23rd International Conference on Hybrid Systems: Computation and Control (HSCC 2020). pp. 1–7 (2020)

[20] Ivanov, R., Jothimurugan, K., Hsu, S., Vaidya, S., Alur, R., Bastani, O.: Compositional learning and verification of neural network controllers. ACM Trans. Embed. Comput. Syst. **20**(5s), 92:1–92:26 (2021)

[21] Ivanov, R., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verisig: verifying safety properties of hybrid systems with neural network controllers. In: Proc. 22nd ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2019), Montreal, Canada, 2019. pp. 169–178 (2019)

[22] Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., et al.: The marabou framework for verification and analysis of deep neural networks. In: Computer Aided Verification: 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I 31. pp. 443–452. Springer (2019)

[23] Katz, S.M., Corso, A.L., Strong, C.A., Kochenderfer, M.J.: Verification of image-based neural network controllers using generative models. In: Proc. IEEE/AIAA 40th Digital Avionics Systems Conference (DASC). pp. 1–10 (2021)

[24] Kong, S., Gao, S., Chen, W., Clarke, E.M.: dReach: δ-Reachability Analysis for Hybrid Systems. In: Proc. 21st Intl. Conf. Tools and Alg. Constr. Anal. Systems (TACAS 2015), London. pp. 200–205. Springer (2015)

[25] LG Electronics America R&D Lab: SVL Simulator. https://www.svlsimulator.com/, last accessed: 2022-06-07

[26] Majumdar, R., Mathur, A.S., Pirron, M., Stegner, L., Zufferey, D.: Paracosm: A Test Framework for Autonomous Driving Simulations. In: Proc. 24th Intl. Conf. Fundamental Approaches to Software Engineering (FASE 2021), Luxembourg, 2021. Lecture Notes in Computer Science, vol. 12649, pp. 172–195. Springer (2021)

[27] Manzanas Lopez, D., Althoff, M., Forets, M., Johnson, T.T., Ladner, T., Schilling, C.: Arch-comp23 category report: artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants. In: EPiC Series in Computing (2023)

[28] de Moura, L.M., Bjørner, N.: Z3: An Efficient SMT Solver. In: Proc. 14th Intl. Conf. Tools and Alg. Constr. Anal. Systems (TACAS 2008). pp. 337–340. Springer (2008)

[29] O'Kelly, M., Abbas, H., Mangharam, R.: Computer-Aided Design for Safe Autonomous Vehicles. Tech. rep., U. Pennsylvania (May 2017), https://repository.upenn.edu/mlab_papers/99

[30] Prakash, P., Murti, C., Nath, J.S., Bhattacharyya, C.: Optimizing DNN Architectures for High Speed Autonomous Navigation in GPS Denied Environments on Edge Devices. In: Proc. 16th Pac. Rim Intl. Conf. on Artificial Intelligence (PRICAI 2019), Fiji. pp. 468–481 (2019)

[31] Prunier, J.C.: Scratchapixel: An Overview of the Rasterization Algorithm. https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation, last accessed: 2020-09-28

[32] Păsăreanu, C.S., Mangal, R., Gopinath, D., Yaman, S.G., Imrie, C., Calinescu, R., Yu, H.: Closed-loop analysis of vision-based autonomous systems: A case study. In: Proc. 35th Intl. Conf. Computer Aided Verification (CAV 2023), Paris, France, 2023. LNCS, vol. 13964, pp. 289–303. Springer (2023)

[33] Shah, S., Dey, D., Lovett, C., Kapoor, A.: AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In: Res. 11th Intl. Conf. on Field and Service Robotics (FSR 2017), Zurich, Switzerland. pp. 621–635. Springer (2017)

[34] Shalev-Shwartz, S., Shammah, S., Shashua, A.: On a formal model of safe and scalable self-driving cars. CoRR **abs/1708.06374** (2017), http://arxiv.org/abs/1708.06374

[35] Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: An abstract domain for certifying neural networks. PACMPL **3**(POPL), 41:1–41:30 (2019)

[36] Blender 3D Creation Suite: https://www.blender.org/, last accessed: 2022-03-08

[37] Sun, X., Khedr, H., Shoukry, Y.: Formal verification of neural network controlled autonomous systems. In: Proc. 22nd ACM Intl. Conf. on Hybrid Systems: Computation and Control (HSCC 2019), Montreal, Canada. pp. 147–156 (2019)

[38] Tesla: Tesla vision update: Replacing ultrasonic sensors with tesla vision. https://www.tesla.com/support/transitioning-tesla-vision (October 5, 2023), accessed: 22/01/2024

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

[40] Tuncali, C.E., Faniekos, G.E., Ito, H., Kapinski, J.: Sim-ATAV: Simulation-Based Adversarial Testing Framework for Autonomous Vehicles. In: Proc. 21st Intl. Conf. on Hybrid Systems: Computation and Control (HSCC 2018), Porto, Portugal. pp. 283–284 (2018)

[41] Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C.J., Kolter, J.Z.: Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. Advances in Neural Information Processing Systems **34** (2021)

[42] Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., Hsieh, C.J.: Fast and Complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In: Proc. Intl. Conf. Learning Representations (ICLR) (2021)