PolyMerge: Compressing 3D Gaussian Splats with Polytope Coverings for Provably Safe Resource-Constrained Navigation

Jihoon Hong, Chih-Yuan Chiu, Sara Fridovich-Keil and Glen Chou

***Abstract*—Obstacle avoidance is essential for safe navigation and motion planning. Recent radiance field reconstruction methods enable object detection and modeling with high fidelity, but remain too memory- and compute-intensive for on-board perception-based path planning. To address these limitations, we propose PolyMerge to convert a large, photorealistic 3D Gaussian Splatting (3DGS) model of a scene into a lightweight representation of convex polytopes whose union provably over-approximates all obstacles in the original 3DGS model. PolyMerge tunes the polytope count to trade off conservativeness and compute cost, and integrates with control barrier functions (CBFs) to plan collision-free paths. We showcase PolyMerge in simulation and hardware experiments on a Crazyflie drone, which uses PolyMerge to compute and follow safe trajectories in real time under severe onboard compute resources, outperforming baselines in speed while guaranteeing safety. For our code and videos, visit https://athlon76.github.io/PolyMerge-website/.**

***Index Terms*—Vision-Based Navigation; Collision Avoidance; Reactive and Sensor-Based Planning**

I. Introduction

To guarantee safe operation, a robot must accurately detect obstacles in its environment. As such, recent advances in radiance field-based scene reconstruction [1]–[4], such as 3D Gaussian splatting (3DGS) methods [4], have attracted increasing attention in the robotics community for their potential to precisely identify the location and geometry of obstacles. However, the high computational and memory demands of most existing scene reconstruction approaches precludes their direct integration into perception pipelines on resource-constrained robot hardware. For example, a 3DGS models often contain hundreds of thousands to millions of Gaussians, occupying megabytes or gigabytes of memory. This far exceeds the memory capacity of mobile platforms, typically in orders of kilobytes for small drones such as Crazyflies. Moreover, existing approaches to improve the computational efficiency of scene reconstructions often focus on retaining visual fidelity for vision tasks [5], [6], but do not ensure geometric over-approximation of occupied space and thus cannot safely be used for motion planning. Among methods designed for robotic perception, despite improvements in computational efficiency [7] there remains a gap in performance requirements that prohibits deployment on resource-constrained robotic hardware such as small drones.

To bridge this gap, we propose PolyMerge, a novel framework for compressing a precomputed scene representation to enable on-device real-time obstacle avoidance and motion planning via a control barrier function (CBF)-based safety filter. Our method can take any photorealistic scene model as input; our experiments are built on 3DGS [4], a state-of-the-art computer vision method that represents a scene as a

Manuscript received: January 11th, 2025; Revised March 28th, 2026; Accepted April 27th, 2026.

This paper was recommended for publication by Editor Ayoung Kim upon evaluation of the Associate Editor and Reviewers' comments.

[1]All authors are with the Georgia Institute of Technology; Jihoon Hong, Chih-Yuan Chiu and Sara Fridovich-Keil are with the School of ECE; Glen Chou is with the Schools of AE and CSP ({jhong392, cyc, sfk, chou} at gatech dot edu).

Digital Object Identifier (DOI): see top of this page.

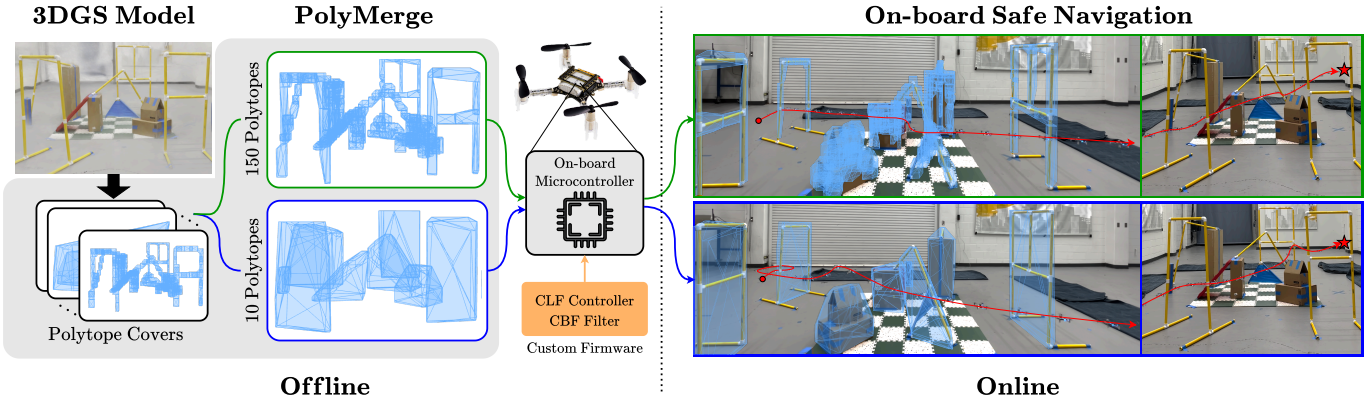


Fig. 1. **Overview. (Left)** PolyMerge converts a 3DGS model into multiple polytope covers, each of which covers all obstacles but with different number of convex hulls and level of conservativeness. **(Middle)** A cover that satisfies the hardware-driven computational constraints can be selected and loaded to the microcontroller of the drone, **(Right)** which then performs CBF filtering of nominal control inputs provided by a CLF-based controller to navigate safely, entirely on-board. Using fewer polytopes reduces hardware cost but may introduce conservativeness in navigation, as can be seen from detours of the drone around hoop obstacles. Red circle and star denotes the start and goal positions of the drone.

collection of ellipsoids. PolyMerge first extracts a volumetric occupancy grid to ensure all obstacles in the input scene are considered occupied. It then computes a covering set of convex polytopes which contains all occupied space. Finally, PolyMerge performs iterative merging by repeatedly replacing pairs of convex polytopes with their convex hull, with the merged polytopes chosen to minimize the volume of unoccupied space within the convex hull. This iterative merging allows PolyMerge to simplify a scene to accommodate arbitrary computational constraints while ensuring obstacle over-approximation with small conservativeness, enabling safe and efficient navigation. Our contributions include:

1) We present PolyMerge, a scene compression algorithm for constructing and merging polytope set covers which computes guaranteed over-approximations of 3D obstacles in a given 3DGS environment model.
2) We show that PolyMerge can modulate the number of geometric primitives used to describe obstacles to satisfy strict resource constraints at the cost of increased conservativeness in the resulting obstacle over-approximation.
3) In simulations and hardware experiments on resource-constrained Crazyflies, we illustrate that PolyMerge outperforms existing baselines [6], [7] (which cannot be deployed at all on Crazyflies or do not guarantee safe navigation) in efficiently generating safe trajectories.

II. Related Work

A. Scene Reconstruction

The design of efficient algorithms to generate high-fidelity 3D scene reconstructions from multi-view 2D images has long been a core focus in both computer vision and robotics. Recent, key advances include the development of implicit neural radiance fields [1], plenoptic voxel grids [2], and state-of-the-art 3D Gaussian Splatting (3DGS) [4] scene models which are the input to our method. While 3DGS captures photorealistic detail of obstacles in a scene, a 3DGS model can contain hundreds of thousands of individual ellipsoids that render it too computationally intensive for direct use in robotic perception and safe navigation. To improve computational efficiency, many methods have been proposed to reduce scene complexity, chiefly by pruning away ellipsoids not essential to visual fidelity [5], [8], or estimating object surfaces via a 2D mesh [6] that is more efficient for downstream computation. While [5], [6], [8] reduce 3DGS model complexity, they do not ensure safety for downstream robotics tasks, as space occupied by ellipsoids in the original 3DGS model may be marked unoccupied in a compressed or mesh-extracted representation.

B. CBFs and Safe Control from Perception Data

Safety filters are designed to minimally adjust unsafe control inputs to prevent safety violations. While a diverse range of safety filters have been proposed [9], we focus on Control Barrier Function (CBF)-based methods [10], which enforce forward set invariance to guarantee safety. In particular, many recent works present CBF-based controller designs which incorporate perception data for obstacle detection and avoidance [11]–[14]. For instance, [11] and [12] use CBFs to avoid obstacles which are currently visible to the robot's sensors, such as LiDAR or stereo cameras, but do not attempt to fully map the robot's environment. In contrast, [13] and [14] formulate CBFs based on scene-wide obstacle representations constructed using LiDAR-inertial odometry and visual-inertial simultaneous localization and mapping (SLAM), respectively, but do not take advantage of high-fidelity photorealistic scene models such as 3DGS [4]. Other methods perform safe perception-based control using image data [15], [16], but focus on ensuring safety under localization error caused by a learned perception module while assuming simple, known obstacle geometry (e.g., a single spherical obstacle); our method complements these approaches.

Methodologically, since our experiments use the polytope-based obstacle sets representations produced by our PolyMerge algorithm to synthesize CBF-based controllers, our work is also related to [17], which likewise generates CBFs in environments with polytope-shaped obstacles. However, in contrast to [17], our main contribution lies in constructing lightweight, polytope-shaped obstacle representations from 3DGS models, rather than the downstream design of CBF controllers.

C. Safe Motion Planning using Scene Maps

The use of radiance field-based map construction to guide safe motion planning tasks has received increasing attention in the robotics literature in recent years. In particular, prior work has used neural [18], [19] or Gaussian representations [20] to encode obstacle locations within a *collision loss term*; such methods, however, cannot guarantee collision avoidance. In contrast, [21] and [22] likewise use 3DGS-based scene reconstructions to encode obstacle locations and geometries, but enforce collision avoidance guarantees via *hard constraints*, similar to our work. However, whereas [21] and [22] use model predictive control and trajectory-level collision checking, respectively, to enforce collision avoidance, our PolyMerge method uses CBF-based filters to design minimally-invasive safe control inputs and can be run on-board with very limited compute (e.g., on Crazyflie drones). Methodologically, our work is most similar to [7], [23], which likewise formulate a CBF safety filter over a 3DGS scene map, improving reconstruction fidelity compared to odometry- and SLAM-based representations. However, our PolyMerge method attains higher computation and memory efficiency compared to [7], [23], by merging polytope set covers over a 3DGS scene to over-approximate obstacles.

III. Preliminaries

A. 3D Gaussian Splatting (3DGS)

A 3DGS model $\mathcal{M} = \{\mathcal{E}_i\}_{i \in I}$ consists of ellipsoids $\mathcal{E}_i$ derived from a number of Gaussian distributions indexed by I, each specified by a mean vector $c_i \in \mathbb{R}^3$, positive definite covariance matrix $\Sigma_i \in \mathbb{R}^{3\times3}$, opacity value, and set of spherical harmonics coefficients to describe color [4]. Concretely, each ellipsoid $\mathcal{E}_i$ is centered at the corresponding mean vector c_i with kernel given by Σ_i^{-1}, i.e., $\mathcal{E}_i := \{p \in \mathbb{R}^3 : (p - c_i)^\top \Sigma_i^{-1} (p - c_i) \leq 1\}$ for each $i \in I$.

B. Control Barrier Functions (CBFs)

CBFs enable the synthesis of controllers that guarantee robot safety by rendering a provided safe set forward invariant [10]. Formally, consider control-affine system dynamics $\dot{x} = f(x) + g(x)u$ evolving over a state space $\mathcal{X} \subseteq \mathbb{R}^n$, with controls u drawn from a set $\mathcal{U} \subseteq \mathbb{R}^{n_i}$ of admissible controls, where $f : \mathcal{X} \to \mathbb{R}^n$ and $g : \mathcal{X} \to \mathbb{R}^{n \times n_i}$. Given a *safe set* $\mathcal{S} \subseteq \mathcal{X}$ characterized as the zero super-level set of a function $\psi : \mathcal{X} \to \mathbb{R}$, i.e., $\mathcal{S} = \{x \in \mathcal{X} : \psi(x) \geq 0\}$, we wish to ensure that $x(t) \in \mathcal{S}$ for all $t \in [0, T]$. We call ψ a CBF if there exists an extended class-$\mathcal{K}_\infty$ function $\alpha : \mathbb{R} \to \mathbb{R}$ satisfying, $\forall x' \in \mathcal{X}$, $\exists u' \in \mathcal{U}$ [10]:

$$\big[L_f\psi(x') + L_g\psi(x')u'\big] \geq -\alpha\big(\psi(x')\big), \tag{1}$$

where L_f and L_g denote Lie derivatives with respect to f and g, respectively. A controller which generates inputs $u(t) \in \mathcal{U}$ satisfying (at each t) (1) with $x' = x(t)$ and $u' = u(t)$, would guarantee the forward-invariance of $\mathcal{S}$ with respect to $x(t)$. In other words, if $x(0) \in \mathcal{S}$ and at each t, $u(t) \in \mathcal{U}$ satisfies (1) with $x = x(t)$, then $x(t) \in \mathcal{S}$ for all t.

C. CBF in Polytope Scenes

While finding a valid CBF that satisfies (1) is often challenging, a previous work [17] established a systematic method of constructing the CBF when obstacles are represented with convex polytopes under single-integrator dynamics $\dot{x} = u$. Specifically, it assumes a finite collection of obstacles $\mathcal{H} = \{H_k\}_{k \in K}$ with index set K, where each H_k is a convex polytope described by bounding planes Λ_k:

$$H_k = \textstyle\bigcap_{l \in \Lambda_k} \big\{p \in \mathbb{R}^3 \mid n_{kl}^\top (p - w_{kl}) \leq 0\big\}, \tag{2}$$

Above, $n_{kl} \in \mathbb{R}^3$ and $w_{kl} \in \mathbb{R}^3$ denote the normal and offset vectors of the l-th plane of H_k, respectively. Then, the barrier function, defined as

$$\psi(p) = \min_{k \in K} \max_{l \in \Lambda_k} n_{kl}^\top (p - w_{kl}), \tag{3}$$

evaluates to a strictly positive value if and only if p does not belong to any polytope H_k.

Since both the min and max functions contained in $\psi(p)$ are non-differentiable, they are approximated by the smooth soft-min and soft-max functions, respectively defined as $A \mapsto -\frac{1}{\kappa} \log\big(\sum_{a \in A} e^{-\kappa a}\big)$ and $A \mapsto \frac{1}{\kappa} \log\big(\sum_{a \in A} e^{\kappa a}\big)$ $\forall A \subset \mathbb{R}$, which admit Lie derivatives; we use $\kappa = 100$ in our experiments. [17] then illustrates that with a function $\alpha(\psi(x)) := \beta\psi(x)$ for some $\beta > 0$, ψ is a CBF that renders the zero super-level set $\mathcal{S}$ (which by construction does not coincide with the interior of any polytope) forward-invariant for a point agent. [17] also extends (3) to a spherical agent with radius r as $\psi_r(p) = \psi(p) - r$, which inflates all obstacles by r.

IV. Problem Statement

While SaferSplat [7] has explored CBF-based safe navigation directly on a 3DGS scene, it cannot be directly used for on-board safe navigation, where both memory and compute resources are often too limited to process a large set of Gaussians. To circumvent these challenges, we formulate our PolyMerge algorithm that compresses a 3DGS model into a light-weight representation, in turn enabling the construction of a CBF that can be evaluated on-device in real time.

For a 3DGS model $\mathcal{M}$, let the set of unsafe states defined by the model $\mathcal{X}_{\text{unsafe}}^{\text{GS}}$ be

$$\mathcal{X}_{\text{unsafe}}^{\text{GS}} = \bigcup_{\mathcal{E}_i \in \mathcal{M}} \mathcal{E}_i, \tag{4}$$

which includes all object surfaces implicitly defined by the ellipsoids $\mathcal{E}_i$. We consider $\mathcal{X}_{\text{unsafe}}^{\text{GS}}$ and $\mathcal{X}_{\text{safe}}^{\text{GS}} := \big(\mathcal{X}_{\text{unsafe}}^{\text{GS}}\big)^c$, as the ground truth unsafe and safe sets, respectively. We also define $\text{Vol}(\cdot)$ to return the volume of given subsets of $\mathbb{R}^3$.

We now describe the problem of 3DGS compression for on-board safe motion planning. Suppose we are given a 3DGS model $\mathcal{M}$, and a budget B over the number of polytopes we can process on-device. We aim to obtain a collection $\mathcal{H}^\star$ of convex polytopes which solves (5), as given below:

$$\min_{\mathcal{H}} \quad L(\mathcal{H}) := \text{Vol}\left(\bigcup_{H_k \in \mathcal{H}} H_k \Big\backslash \mathcal{X}_{\text{unsafe}}^{\text{GS}}\right), \tag{5a}$$

$$\text{s.t.} \quad \mathcal{X}_{\text{unsafe}}^{\text{GS}} \subseteq \textstyle\bigcup_{H_k \in \mathcal{H}} H_k, \tag{5b}$$

$$|\mathcal{H}| \leq B, \tag{5c}$$

where $|\cdot|$ denotes set cardinality. (5b) guarantees that $\mathcal{H}$ over-approximates the true unsafe set and thus ensures safety, while (5c) limits the number of polytopes in $\mathcal{H}$ to respect computational and storage constraints imposed by hardware. Under (5b) and (5c), we aim to minimize (5a), which measures how conservative $\mathcal{H}$ is compared to $\mathcal{M}$ in describing obstacles. A set of convex polytopes $\mathcal{H}$ which yields a large objective value $L(\mathcal{H})$ would designate an excessively large subset of $\mathcal{X}_{\text{safe}}^{\text{GS}}$ as impassable, and thus induce overly conservative motion plans downstream.

V. Methods

Since the optimization problem described in Sec. IV is difficult to solve exactly, we present an algorithm, PolyMerge, to compute an approximation $\tilde{\mathcal{H}}$ of $\mathcal{H}^\star$ given a 3DGS model $\mathcal{M}$ (Fig. 2). PolyMerge proceeds in three steps: (a) Deriving a voxel-based occupancy grid $\mathcal{V}^+$ from the cloud of 3DGS ellipsoids (Sec. V-A); (b) Constructing a collection of convex sets $\overline{\mathcal{H}}$ which covers[1] the occupied voxels (Sec. V-B); (c) Iteratively and greedily merging local groups of occupied voxels and taking their convex hulls to produce compressed collections of polytopes (Sec. V-C). As a result, we obtain provable overapproximations of obstacle sets with varying degrees of conservativeness, as shown in Fig. 2.

A. Occupancy Voxelization

First, we extract a voxel-based occupancy grid $\mathcal{V}^+$ from the 3DGS model $\mathcal{M}$, which already achieves some compression because many ellipsoids often overlap and can be covered by a small number of occupied voxels. We define a voxel $v \subset \mathbb{R}^3$ with center $p_v \in \mathbb{R}^3$ and grid size g by:

$$v := \left\{p' \in \mathbb{R}^3 \mid \|p' - p_v\|_\infty \leq \frac{g}{2}\right\}, \tag{6}$$

[1]Given a set S, a collection of sets $\mathcal{T}$ is called a *cover* of S if, for each point $x \in S$, there exists some set $T \in \mathcal{T}$ such that $x \in T$.

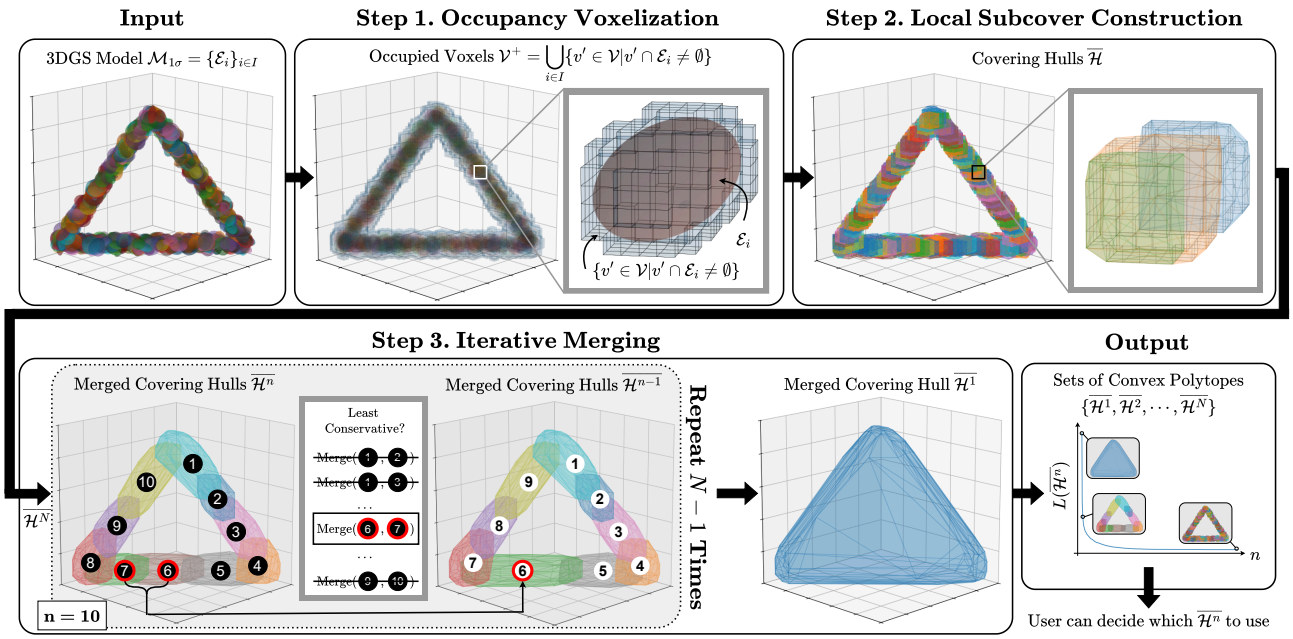


Fig. 2. **Compression Algorithm**: Given a 3DGS model, PolyMerge extracts occupancy information into a voxel grid (Sec. V-A), constructs a local sub-cover of the occupied voxels (Sec. V-B), and iteratively merges the cover elements to incrementally reduce the number of convex hulls (Sec. V-C).

and let $\mathcal{V}$ denote the set of all voxels in $\mathbb{R}^3$ whose 8 corners all have coordinates that are multiples of g.

We aim to construct a sparse voxel occupancy grid $\mathcal{V}^+ \subseteq \mathcal{V}$ from the 3DGS model $\mathcal{M}$ which maintains only voxels that overlap with some ellipsoid $\mathcal{E}_i$ in $\mathcal{M}$, i.e.,:

$$\mathcal{V}^+ := \{v \in \mathcal{V} \mid \exists i \in I \text{ s.t. } v \cap \mathcal{E}_i \neq \phi\}. \tag{7}$$

To construct $\mathcal{V}^+$, it suffices to evaluate the following *occupancy map* $O : \mathcal{V} \to \{0, 1\}$ over all $v \in \mathcal{V}$:

$$O(v) = \mathbb{1}\big[\exists i \in I \text{ s.t. } v \cap \mathcal{E}_i \neq \phi\big], \tag{8}$$

since $\mathcal{V}^+ = \{v \in \mathcal{V} \mid O(v) = 1\}$. In turn, for a given voxel v and an ellipsoid $\mathcal{E}_i$, we can determine if $v \cap \mathcal{E}_i \neq \phi$ by evaluating whether $\min_{p\in v}(p - c_i)^\top \Sigma_i^{-1}(p - c_i) \leq 1$, which we can compute with a small quadratic program. However, naïvely evaluating whether v intersects every possible $\mathcal{E}_i$ is computationally heavy, especially when many ellipsoids and/or voxels are used in the scene description, as is the case when the scene spans a large area, or the grid resolution is high. To reduce computational cost, we leverage the fact that each voxel often intersects with only a small number of ellipsoids, which allows us to avoid solving a majority of the optimization problems. Concretely, we construct axis-aligned bounding boxes $\mathcal{B}_i$ containing each $\mathcal{E}_i$, and, for each voxel v, identify a subset of ellipsoids which may overlap with v, as indexed by $I_v := \{i \mid v \cap \mathcal{B}_i \neq \emptyset\}$. We then evaluate:

$$O(v) = \mathbb{1}\Big[\min_{i\in I_v} \min_{p\in v}(p - c_i)^\top \Sigma_i^{-1}(p - c_i) \leq 1\Big]. \tag{9}$$

After this step, we have a sparse set of occupied voxels $\mathcal{V}^+$ covering all obstacles in the original 3DGS scene. Note that, if an input scene were provided in any format other than 3DGS, PolyMerge would first extract an occupancy grid $\mathcal{V}^+$ and then the rest of the method would proceed as usual.

B. Local Subcover Construction

Although the sparse voxel grid $\mathcal{V}^+$ constructed in Sec. V-A covers the obstacle sets, the cardinality of $\mathcal{V}^+$ may be too high to satisfy the hardware limitations underlying the constraint (5c). To address this issue, we use $\mathcal{V}^+$ to construct a collection $\overline{\mathcal{H}}$ of convex sets, with lower cardinality compared to $\mathcal{V}^+$, which provably covers but does not excessively over-approximate the unsafe space $\mathcal{X}^{\text{GS}}_{\text{unsafe}}$ (in light of the objective (5a) and constraint (5b)).

Concretely, for each $v \in \mathcal{V}^+$, we first define the cluster $\mathcal{C}_v \subseteq \mathcal{V}^+$ of voxels within a bg neighborhood of v:

$$\mathcal{C}_v = \{v' \in \mathcal{V}^+ \mid \|p_{v'} - p_v\|_\infty \leq bg\}, \tag{10}$$

where $b \in \mathbb{N}$ is a design choice and g is the voxel side length as in (6). Let $\mathcal{C} := \{\mathcal{C}_v \mid v \in \mathcal{V}^+\}$ be the set of all such voxel clusters, which covers $\mathcal{V}^+$ by construction. To facilitate the construction of a cover for $\mathcal{X}^{\text{GS}}_{\text{unsafe}}$ with lower cardinality than $\mathcal{V}^+$, we apply the weighted set cover algorithm [24] (Remark 1) to extract a sub-cover $\overline{\mathcal{C}}$ of $\mathcal{V}^+$ from $\mathcal{C}$, i.e., a sub-collection $\overline{\mathcal{C}} \subseteq \mathcal{C}$ which covers $\mathcal{V}^+$. Finally, we define $\overline{\mathcal{H}}$ to be the collection of convex hulls of voxel clusters in $\overline{\mathcal{C}}$, i.e., we set $\overline{\mathcal{H}} := \{\text{Hull}(\overline{\mathcal{C}}_v) \mid \overline{\mathcal{C}}_v \in \overline{\mathcal{C}}\}$. By construction, $\overline{\mathcal{H}}$ covers $\mathcal{V}^+$ (thus satisfying (5b)) while including minimal volume outside of $\mathcal{V}^+$, up to the approximation error of the weighted set cover algorithm.

Remark 1: (Application of the Weighted Set Cover Algorithm [24].) Let a set S, a cover $\mathcal{T}$ of S consisting of subsets of S, and a weight function $W : \mathcal{T} \to \mathbb{R}$ be given. The weighted set cover algorithm of [24] produces a sub-cover $\overline{\mathcal{T}}$ which approximates the optimal subcover with respect to the original cover $\mathcal{T}$ and the weight function W, given by:

$$\mathcal{T}^\star := \arg\min_{\mathcal{T}'} \left\{ \sum_{\mathcal{T}'_i \in \mathcal{T}'} W(\mathcal{T}'_i) \mid \mathcal{T}' \subset \mathcal{T}, S \subseteq \bigcup_{\mathcal{T}'_i \in \mathcal{T}'} \mathcal{T}'_i \right\},$$

which is NP-complete to compute [24]. In the context of Sec. V-B, to compute the sub-cover $\overline{\mathcal{C}}$, we take $S = \mathcal{V}^+$ and $\mathcal{T} = \mathcal{C}$, and define the weight function $W : \mathcal{C} \to \mathbb{R}$ as:

$$W(\mathcal{C}_v) := \text{Vol}\Big(\text{Hull}(\mathcal{C}_v) \Big\backslash \bigcup \{v' \mid v' \in \mathcal{V}^+\}\Big). \tag{11}$$

Note that W approximates the volume of unoccupied space $\mathcal{V} \backslash \mathcal{V}^+$ included in the hull of $\mathcal{C}_v$, which quantifies the extent of obstacle set over-approximation.

C. Iterative Merging

The final step of PolyMerge is to refine $\overline{\mathcal{C}}$ so that $\overline{\mathcal{H}} := \{\text{Hull}(\overline{\mathcal{C}}_v) \mid \overline{\mathcal{C}}_v \in \overline{\mathcal{C}}\}$ satisfies the hardware constraints (5c). We iteratively merge pairs of elements $\overline{\mathcal{C}}_v$ of the sub-cover $\overline{\mathcal{C}}$ until (5c) is satisfied. This process facilitates an explicit visualization of the relation between the number of polytopes and the objective function L (see (5a)), which allows users to navigate the tradeoff between hardware capacity constraints and obstacle over-approximation conservativeness.

Let $N := |\overline{\mathcal{C}}|$, and let the sub-cover and convex hulls before merging be respectively denoted by:

$$\overline{\mathcal{C}^N} := \overline{\mathcal{C}} = \{\overline{\mathcal{C}_i^N}\}_{i \in I^0}, \quad \overline{\mathcal{H}^N} := \overline{\mathcal{H}} = \{\overline{\mathcal{H}_i^N}\}_{i \in I^0}, \tag{12}$$

whose elements are now indexed from a new index set I^0, instead of using a voxel v. We reuse the weight W in (11) and select a pair $(i^\star, j^\star)$ with minimum weight, as follows:

$$(i^\star, j^\star) = \underset{i,j \in I^0}{\arg\min}\, W(\overline{\mathcal{C}_i^N} \cup \overline{\mathcal{C}_j^N}). \tag{13}$$

Then, we merge the voxel clusters indexed $i^\star$ and $j^\star$, i.e., $\overline{\mathcal{C}_{i^\star}^N}$ and $\overline{\mathcal{C}_{j^\star}^N}$, into the following new sub-cover (see also "Step 3" in Fig. 2):

$$\overline{\mathcal{C}^{N-1}} := \Big(\overline{\mathcal{C}^N} \setminus \{\overline{\mathcal{C}_{i^\star}^N}, \overline{\mathcal{C}_{j^\star}^N}\}\Big) \cup \{\overline{\mathcal{C}_{i^\star}^N} \cup \overline{\mathcal{C}_{j^\star}^N}\} \tag{14}$$

indexed by I^1, and a new set of convex hulls

$$\overline{\mathcal{H}_i^{N-1}} := \{\text{Hull}(\overline{\mathcal{C}_i^{N-1}})\}_{i \in I^1}. \tag{15}$$

We repeat the above process to acquire $\overline{\mathcal{C}^n}$ and $\overline{\mathcal{H}^n}$ for $n = N-2, N-3, \cdots, 1$, with $|\overline{\mathcal{H}^n}|$ decreasing by one per iteration.

The memory and compute complexities of each iteration as described above grow quadratically with the sub-cover size, so the merging process can become costly when the initial sub-cover size is large. To avoid incurring this quadratic cost, we construct an axis-aligned bounding box $\mathcal{B}_i^n$ for each sub-cover element $\overline{\mathcal{C}_i^n}$, and evaluate $W(\overline{\mathcal{C}_i^n} \cup \overline{\mathcal{C}_j^n})$ only when $\mathcal{B}_i^n \cap \mathcal{B}_j^n \neq \emptyset$. We thus only consider merging sub-cover element pairs $(\overline{\mathcal{C}_i^n}, \overline{\mathcal{C}_j^n})$ that are in each others' vicinity, since, for all other pairs, $\text{Hull}(\overline{\mathcal{C}_i^n} \cup \overline{\mathcal{C}_j^n})$ is likely to contain safe set regions, and thus the weight $W(\overline{\mathcal{C}_i^n}, \overline{\mathcal{C}_j^n})$ is likely large.

D. Guarantees of Obstacle Set Overapproximation

Since the steps of our PolyMerge method generate progressively conservative overapproximations of the unsafe set $\mathcal{X}_{\text{unsafe}}^{\text{GS}}$, the input 3DGS model $\mathcal{M}$ is provably covered by each of the polytope covers $\overline{\mathcal{H}^n}$. For a mathematically rigorous argument, see Thm. 1 in the appendix.

VI. Experiments

We evaluate PolyMerge through both simulation and hardware experiments. We describe details of the simulation (Sec. VI-A) and hardware (Sec. VI-B) experiments, followed by baselines that we compare against and metrics that we report (Sec. VI-C), and our experiment results (Sec. VI-D).

A. Simulation

We perform simulations using the mip-NeRF360 dataset [25], which contains 9 scenes, 5 outdoor and 4 indoor, and is widely used to evaluate 3D scene reconstruction methods. For each scene, we apply PolyMerge to $\mathcal{M}$ to acquire $\{\overline{\mathcal{H}^n}\}_{n=1}^N$. After compression, we sample 100 random start and goal state pairs $\{(x_{\text{start}}^m, x_{\text{goal}}^m)\}_{m=1}^{100}$ from the scene such that x_{goal}^m is at least a certain distance away from x_{start}^m, and all start and goal states belong to $\mathcal{X}_{\text{safe}}^{\text{GS}}$.

Then, we simulate navigation from x_{start}^m to x_{goal}^m in $\overline{\mathcal{H}^n}$ for various choices of n with a discrete time-step $\Delta t = 0.01$. For simplicity, we assume a spherical 3D agent of radius r with single-integrator dynamics $x_{t+\Delta t} = x_t + u_{\text{cbf},t}\Delta t$, with the control input $u_{\text{cbf},t}$ computed as follows. First, we compute a nominal control input u_{nom} via:

$$u_{\text{nom}} := \underset{u}{\arg\min}\ \tfrac{1}{2}\|u\|_2^2 \tag{16a}$$

$$\text{s.t. } L_f V(x) + L_g V(x) u \leq -cV(x), \tag{16b}$$

where $V(x) := \frac{1}{2}\|x - x_{\text{goal}}\|_2^2$ is a control Lyapunov function (CLF) [10], [26], for some constant $c > 0$, directing the agent towards the goal state. Second, since the nominal controller is unaware of the obstacles in the scene, to prevent collision u_{nom} is filtered by the CBF ψ_r defined in Sec. III-C, which encodes $\overline{\mathcal{H}^n}$ as the set of polytopes representing obstacles:

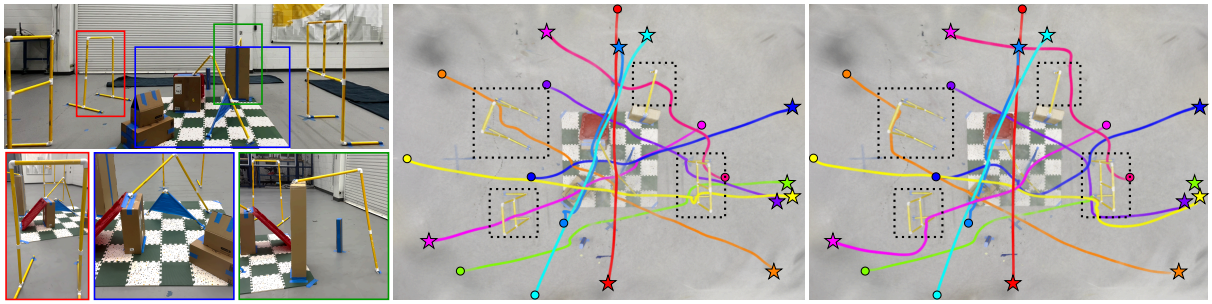
$$u_{\text{cbf}} := \underset{u}{\arg\min}\ \tfrac{1}{2}\|u - u_{\text{nom}}\|_2^2 \tag{17a}$$

$$\text{s.t. } L_f \psi_r(x) + L_g \psi_r(x) u \geq -\beta \psi_r(x). \tag{17b}$$

Finally, we update the agent state using u_{cbf}, i.e., $x_{t+\Delta t} = x_t + u_{\text{cbf}}\Delta t$. From x_{start}, we unroll the dynamics following this state update until either (a) the agent is sufficiently close to x_{goal}, or (b) the agent stays near a position with little or no state update for a certain number of time-steps, which occasionally occurs due to the myopic nature of the nominal controller and safety filter. We repeat this simulation for the same set of start and goal pairs across varying values of n.

B. Hardware Experiments

We also perform hardware experiments on a Crazyflie2.1+ drone in a custom indoor scene (Fig. 3). Prior to flight, we train a 3DGS model offline from video footage of the scene, and acquire $\{\overline{\mathcal{H}^n}\}_{n=1}^N$ similarly as in our simulation experiments. After occupancy voxelization, we filter out voxels that overlap with less than 30 unique ellipsoids. This step was included to remove "floaters", which are Gaussians floating around in free, unoccupied space, artifacts often found in 3DGS models [27]. This filtering could be skipped entirely if the 3DGS model were free of floaters, which is an active area of research [28], [29]. Then, we modify the drone firmware to run the CLF



(a) Hardware Experiment Scene (b) Flight Trajectories for $\overline{\mathcal{H}^{150}}$ (c) Flight Trajectories for $\overline{\mathcal{H}^{10}}$

Fig. 3. **Hardware Experiments. (a)** *Top*: The scene in which we perform drone experiments. *Bottom*: 3 different parts of the scene from different views. **(b-c)** Visualization of 10 different trajectories of the drone, when the obstacles are represented using (b) 150 and (c) 10 convex hulls, respectively (see Fig. 1). Circles represent the start positions, and stars denote the goal positions. Dotted boxes highlight the key differences between the two plots, where the drone successfully passes through the hoop in (b) while taking a detour around the hoop (orange plan) or even failing to navigate further (green plan) in (c).

TABLE I

MEMORY COMPARISON. EACH ROW SHOWS THE NUMBER OF ELLIPSOIDS OR CONVEX HULLS NEEDED TO REPRESENT OBSTACLES, AND THE FILE SIZE IN MEMORY. THE NUMBER OF HULLS ARE SET TO BE EQUAL FOR SUGAR + COACD AND POLYMERGE FOR FAIR COMPARISON.

		Bicycle	Flowers	Garden	Stump	Treehill	Room	Counter	Kitchen	Bonsai
SaferSplat [7]	Ellipsoids	1.50M	1.54M	1.61M	1.47M	1.44M	1.06M	1.43M	1.37M	1.40M
	Size (MB)	671.1	692.0	718.7	659.8	644.2	473.6	640.3	612.5	626.7
SuGaR [6] + CoACD [30]	Hulls	3,296	2,624	1,006	2,458	2,681	454	1,367	901	2,080
	Half-planes in Hulls	189K	149K	58K	150K	150K	26K	76K	53K	119K
	Size (MB)	13.5	10.6	4.2	10.7	10.7	1.8	5.6	3.8	8.5
Ours	Hulls	3,296	2,624	1,006	2,458	2,681	454	1,367	901	2,080
	Half-planes in Hulls	258K	209K	80K	207K	217K	33K	82K	69K	172K
	Size (MB)	60.7	49.8	20.1	67.3	58.8	8.5	10.6	17.4	39.4

nominal controller and the CBF-based safety filter on-board during flight. Specifically, the controller computes u_{nom} via (16) with the drone's state estimates, which are produced by its extended Kalman filter (EKF) based on the real-time position information broadcast from a Vicon motion-capture system. The controller reads off $\overline{\mathcal{H}^n}$ stored in the flash memory of the drone, applies the safety filter to u_{nom} following (17), and sets velocity way-points. While on-board state estimation is also desirable, we leave this to future work and assume exact position information is available in our experiments.

We fly the drone with this custom controller for 10 different $(x_{\text{start}}, x_{\text{goal}})$ pairs, with start states on the ground (i.e. $z = 0$), and goal states chosen such that obstacles lie along the drone's trajectory. We repeat this experiment for $n \in \{10, 30, 50, 100, 150\}$ and configure the drone to run the controller at different time-step frequencies for different values of n, as the compute cost of the CBF filter increases with n.

C. Metrics and Baselines

In simulation, we compare our PolyMerge against two baselines for safe navigation on top of a 3DGS model. The first baseline is SaferSplat [7], which uses a 3DGS model $\mathcal{M}$ for CBF-based safe navigation. The second baseline consists of SuGaR [6], a state-of-the-art technique for extracting a mesh from a 3DGS scene, coupled with CoACD [30], a method that converts a mesh into a set of convex polytopes. SuGaR tends to generate 3DGS models with a smaller number of Gaussians, since it introduces a regularization term to align Gaussians to object surfaces. For fair comparison, we run all simulations starting from the same 3DGS model trained by SuGaR.

To evaluate the reduction in hardware cost enabled by PolyMerge relative to our baselines, we report the memory footprint and computation speed of each method. For memory, we report the total number of ellipsoids and polytopes and the file size in bytes, while for computation speed, we report simulation runtime. In addition, we quantify the conservativeness of each representation, as measured by the total volume of space occupied by the polytopes, which we approximate through Monte Carlo sampling. Finally, we report other relevant statistics of the trajectories, including trajectory lengths and goal-reaching success rates.

D. Results

Table I shows the memory footprint of storing ellipsoids needed for SaferSplat [7], convex hulls acquired from the Gaussians using SuGaR [6] and CoACD [30], and finally the convex hulls acquired from PolyMerge in each scene in mip-NeRF360 [25]. The number of convex hulls for SuGaR + CoACD is chosen as the maximum number attainable by tuning the parameters of CoACD when applying it to the mesh constructed by SuGaR, and we use the same number of convex hulls for PolyMerge to ensure fair comparison. Across all scenes, we observe an up to 10x reduction in memory footprint when convex polytopes, rather than ellipsoids are stored, since 3DGS models contain a large numbers of Gaussians. In addition, although the same number of convex hulls are used for both polytope-based methods, PolyMerge occupies more memory than SuGaR + CoACD because its polytopes are comprised of more half-planes. Nevertheless, PolyMerge provides an over-approximation of the obstacles covering all voxels in $\mathcal{V}^+$ while SuGaR + CoACD covers only a strict subset of obstacles, which does not guarantee

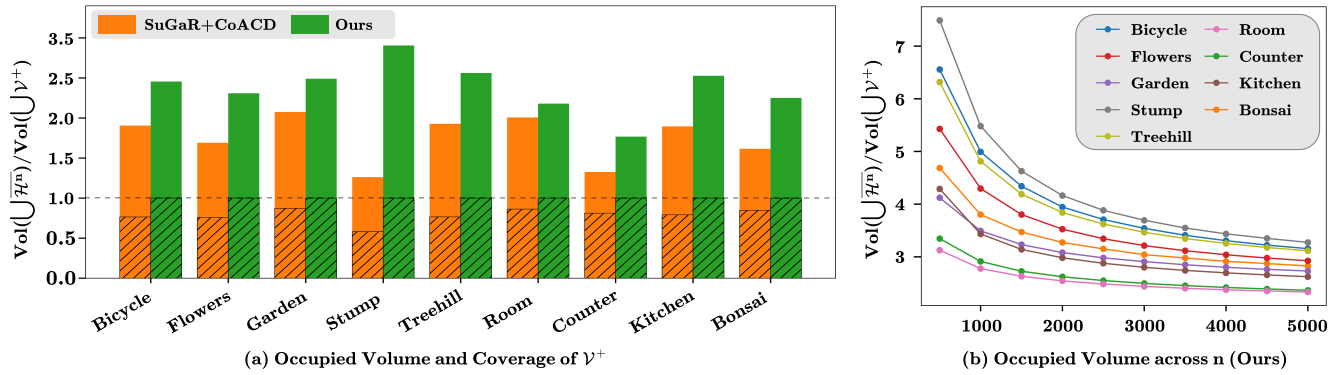


Fig. 4. **Conservativeness of Convex Hulls.** (a) The volume of the union of convex hulls, normalized by the volume of the union of occupied voxels. Hatched bars show the volume of the intersection of the two. Hatched bars reaching 1.0 mean all occupied voxels are included in the union of convex hulls, while those below 1.0 mean the convex hulls do not entirely cover all occupied voxels; SuGaR + CoACD fails to cover all occupied voxels. Portions of the bar without hatches are volumes of obstacle-free region included in the hulls, which represent conservativeness. (b) The trade-off between the number of convex hulls n and the conservativeness of the convex hull sub-cover. Using fewer convex hulls results in the hulls covering a larger obstacle-free region.

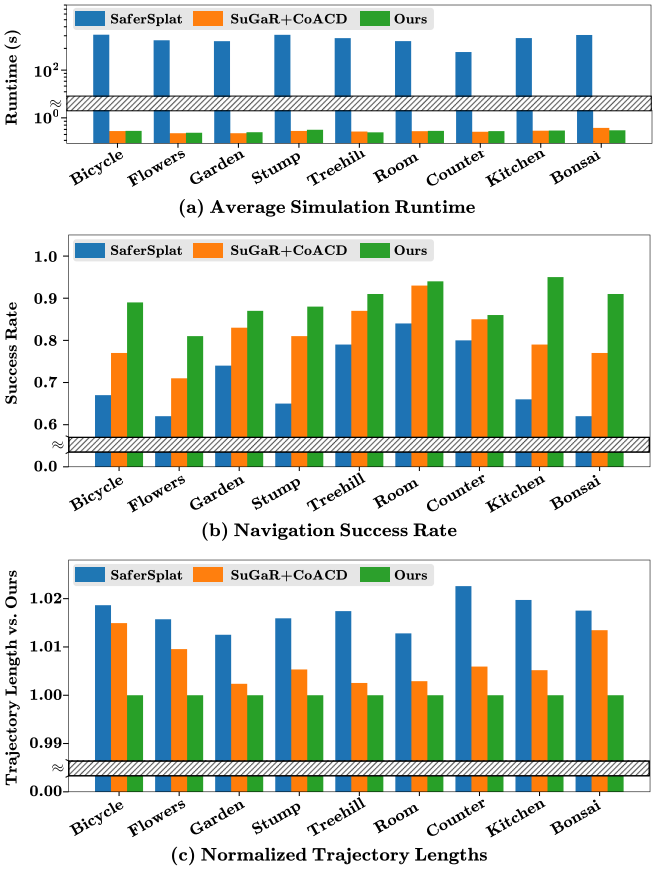


Fig. 5. **Simulation Results.** (a) *Average simulation runtime over 100 trajectories on an RTX A6000 GPU*: Methods using polytopes (SuGaR + CoACD, ours) reduce computation relative to SaferSplat, which uses dense ellipsoid clouds. (b) *Fraction of trajectories that reach within* 1% *distance to* x_{goal}: Our method has highest success rate. (c) *Length of successful trajectories across methods, divided by that of ours and averaged over trajectories*: Our method empirically achieves a shorter trajectory length compared to baselines. (Only trajectories for which all 3 methods succeed are used.)

safety (Fig. 4a). Moreover, PolyMerge allows users to reduce conservativeness by using more convex hulls (via the trade-off curves in Fig. 4b); further improvements can be attained by using a smaller voxel size g or local cover size b.

The smaller number of polytopes compared to ellipsoids also leads to improved compute speeds. On a NVIDIA RTX A6000 GPU, we observe ≈100x speed-up in average simulation runtime in polytope-based representations over ellipsoids (see Fig. 5a). We also empirically find that the success rate over 100 simulations increases across all scenes under PolyMerge compared to the baselines (Fig. 5b), and that goal-reaching is achieved more effectively with shorter trajectories (Fig. 5c). We hypothesize that the over-approximation of the obstacles introduced by the polytopes in PolyMerge may have helped regularize the unsafe set defined by the ellipsoids, which makes the agent less susceptible to "traps" from which the agent can struggle or fail to escape.

Our hardware experiment results are given in Fig. 3 and Table II. We fly the Crazyflie with a CLF-based nominal controller and a CBF filter on-board every 1, 2, 2, 3, and 4 stabilizer loops under $\overline{\mathcal{H}^n}$, for $n \in \{10, 30, 50, 100, 150\}$ respectively. Since the stabilizer loop of a Crazyflie 2.1+ drone runs at 500Hz [31] when an EKF is used for pose estimation, our controller runs at 500, 250, 250, 167, and 125Hz, for each number n of polytopes. These speeds are set at roughly the maximum feasible for the drone's CPU.

The resulting trajectories using 150 and 10 convex hulls, displayed in Fig. 3b-c, show that the drone is able to navigate through the hoop obstacles when $n = 150$, but flies around them when the convex hulls are merged and eventually cover the interior of the hoops. Table II reports additional quantitative results, which include (i) success rate, measured by the fraction of start-goal pairs for which the drone reached within 5 cm of the goal without getting stuck, and (ii) the minimum distance to the set of 150 polytopes representing ground truth obstacles, after visually verifying the tight localization of the polytopes around objects. Since the drone usually flies closer to obstacles when using more polytopes due to less over-approximation, we see a decreasing trend in the second row.

VII. Conclusion, Limitations, and Future Work

We present PolyMerge, a method for over-approximating obstacles in a 3DGS scene using a desired number of convex polytopes for safe CBF-based navigation. PolyMerge paves the way for integrating state-of-the-art photorealistic scene modeling into real-time onboard perception and collision-free motion planning on highly resource-constrained robotic platforms. It guarantees safety by ensuring obstacles in a

given scene model are over-approximated, gracefully trading off computational complexity with the degree of over-approximation or conservativeness. Future work will address limitations of PolyMerge, including (i) extension to dynamic scenes with moving obstacles, (ii) incorporating on-board localization, for instance via SLAM or odometry, and (iii) enabling the scene map to update and expand as the robot moves beyond an initial scene model.

TABLE II
DRONE TRAJECTORY STATISTICS IN HARDWARE EXPERIMENTS.

	Number of Convex Hulls				
	10	30	50	100	150
Success Rate	0.7	0.8	1.0	0.8	0.9
Min Obstacle Distance	0.268	0.253	0.098	0.117	0.097

REFERENCES

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis," in *ECCV*, 2020.

[2] S. Fridovich-Keil, A. Yu, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa, "Plenoxels: Radiance Fields without Neural Networks," in *CVPR*, 2022.

[3] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant Neural Graphics Primitives with a Multiresolution Hash Encoding," *ACM Trans. Graph.*, vol. 41, no. 4, pp. 102:1–102:15, Jul. 2022.

[4] B. Kerbl, G. Kopanas, T. Leimkuehler, and G. Drettakis, "3D Gaussian Splatting for Real-Time Radiance Field Rendering," *ACM Trans. Graph.*, vol. 42, no. 4, Jul. 2023.

[5] A. Hanson, A. Tu, V. Singla, M. Jayawardhana, M. Zwicker, and T. Goldstein, "PUP 3D-GS: Principled Uncertainty Pruning for 3D Gaussian Splatting," in *CVPR*, June 2025, pp. 5949–5958.

[6] A. Guédon and V. Lepetit, "SuGaR: Surface-Aligned Gaussian Splatting for Efficient 3D Mesh Reconstruction and High-Quality Mesh Rendering," *CVPR*, 2024.

[7] T. Chen, A. Swann, J. Yu, O. Shorinwa, R. Murai, M. Kennedy, and M. Schwager, "A Control Barrier Function for Safe Navigation with Online Gaussian Splatting Maps," in *ICRA*, 2025.

[8] M. S. Ali, M. Qamar, S.-H. Bae, and E. Tartaglione, "Trimming the Fat: Efficient Compression of 3D Gaussian Splats through Pruning," in *BMVC*. BMVA, 2024.

[9] K.-C. Hsu, H. Hu, and J. F. Fisac, "The Safety Filter: A Unified View of Safety-Critical Control in Autonomous Systems," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 7, pp. 47–72, 2024.

[10] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control Barrier Functions: Theory and Applications," in *2019 18th European Control Conference (ECC)*, 2019, pp. 3420–3431.

[11] M. De Sa, P. Kotaru, and K. Sreenath, "Point Cloud-Based Control Barrier Function Regression for Safe and Efficient Vision-Based Control," in *ICRA*, 2024, pp. 366–372.

[12] K. Long, C. Qian, J. Cortés, and N. Atanasov, "Learning Barrier Functions With Memory for Robust Safe Navigation," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4931–4938, 2021.

[13] H. U. Unlu, D. Chaikalis, V. Gonçalves, and A. Tzes, "Control Barrier Functions and LiDAR-Inertial Odometry for Safe Drone Navigation in GNSS-Denied Environments," in *Motion Planning for Dynamic Agents*, Z. A. Ali and A. Israr, Eds., London, 2023, ch. 6.

[14] S. Zhou, S. Papatheodorou, S. Leutenegger, and A. P. Schoellig, "Control-Barrier-Aided Teleoperation with Visual-Inertial SLAM for Safe MAV Navigation in Complex Environments," *arXiv*, 2024.

[15] C. Dawson, B. Lowenkamp, D. Goff, and C. Fan, "Learning safe, generalizable perception-based hybrid control with certificates," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1904–1911, 2022.

[16] G. Chou, N. Ozay, and D. Berenson, "Safe Output Feedback Motion Planning from Images via Learned Perception Modules and Contraction Theory," in *Algorithmic Foundations of Robotics XV*, 2022.

[17] T. G. Molnar, "Navigating Polytopes with Safety: A Control Barrier Function Approach," in *IEEE Conference on Control Technology and Applications (CCTA)*, 2025.

[18] M. Adamkiewicz, T. Chen, A. Caccavale, R. Gardner, P. Culbertson, J. Bohg, and M. Schwager, "Vision-Only Robot Navigation in a Neural Radiance World," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4606–4613, 2022.

[19] M. Kurenkov, A. Potapov, A. Savinykh, E. Yudin, E. Kruzhkov, P. Karpyshev, and D. Tsetserukou, "NFOMP: Neural Field for Optimal Motion Planner of Differential Drive Robots With Nonholonomic Constraints," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10 991–10 998, 2022.

[20] Q. Chen, N. Gao, S. Huang, J. Low, T. Chen, J. Sun, and M. Schwager, "GRAD-NAV++: Vision-Language Model Enabled Visual Drone Navigation With Gaussian Radiance Fields and Differentiable Dynamics," *IEEE Robotics and Automation Letters*, vol. 11, no. 2, 2026.

[21] J. Low, M. Adang, J. Yu, K. Nagami, and M. Schwager, "SOUS VIDE: Cooking Visual Drone Navigation Policies in a Gaussian Splatting Vacuum," *IEEE Robotics and Automation Letters*, vol. 10, no. 5, 2025.

[22] T. Chen, O. Shorinwa, J. Bruno, A. Swann, J. Yu, W. Zeng, K. Nagami, P. Dames, and M. Schwager, "Splat-Nav: Safe Real-Time Robot Navigation in Gaussian Splatting Maps," *IEEE Transactions on Robotics*, vol. 41, pp. 2765–2784, 2025.

[23] M. Tong, C. Dawson, and C. Fan, "Enforcing Safety for Vision-based Controllers via Control Barrier Functions and Neural Radiance Fields," in *IEEE International Conference on Robotics and Automation*, 2023.

[24] V. Chvatal, "A Greedy Heuristic for the Set-covering Problem," *Mathematics of operations research*, vol. 4, no. 3, pp. 233–235, 1979.

[25] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, "Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields," *CVPR*, 2022.

[26] M. Z. Romdlony and B. Jayawardhana, "Uniting Control Lyapunov and Control Barrier Functions," in *53rd IEEE Conference on Decision and Control*, 2014, pp. 2293–2298.

[27] B. Fei, J. Xu, R. Zhang, Q. Zhou, W. Yang, and Y. He, "3d gaussian splatting as new era: A survey," *IEEE Transactions on Visualization and Computer Graphics*, 2024.

[28] H. Xiong, S. Muttukuru, H. Xiao, R. Upadhyay, P. Chari, Y. Zhao, and A. Kadambi, "Sparsegs: sparse view synthesis using 3d gaussian splatting," in *2025 International Conference on 3D Vision (3DV)*. IEEE, 2025, pp. 1032–1041.

[29] L. Wang, Q. Ren, K. Liao, H. Wang, Z. Chen, and Y. Tang, "StableGS: A Floater-free Framework for 3D Gaussian Splatting," *arXiv preprint arXiv:2503.18458*, 2025.

[30] X. Wei, M. Liu, Z. Ling, and H. Su, "Approximate Convex Decomposition for 3D Meshes with Collision-aware Concavity and Tree Search," *ACM Trans. Graph.*, vol. 41, no. 4, Jul. 2022.

[31] Crazyflie 2.1+, "Crazyflie 2.1+ Documentation," https://www.bitcraze.io/products/crazyflie-2-1-plus/, 2025.

APPENDIX

Theorem 1: For each $n \in \{1, \cdots, N\}$, $\overline{\mathcal{H}^n}$ covers $\mathcal{X}^{\text{GS}}_{\text{unsafe}}$.

Proof: First, we prove that $\mathcal{V}^+$ covers $\mathcal{X}^{\text{GS}}_{\text{unsafe}} := \bigcup_{\mathcal{E}_i \in \mathcal{M}} \mathcal{E}_i$. Fix $p \in \mathcal{X}^{\text{GS}}_{\text{unsafe}}$ arbitrarily. Since the set of all voxels $\mathcal{V}$ covers $\mathbb{R}^3$, there exists a voxel v_p which contains p, and thus intersects $\mathcal{E}_i$. Then, by definition of $\mathcal{V}^+$ in Eq. (7), we have $v_p \in \mathcal{V}^+$; thus, $\mathcal{V}^+$ covers $\mathcal{X}^{\text{GS}}_{\text{unsafe}}$.

Next, we prove that $\overline{\mathcal{H}} := \{\text{Hull}(\overline{\mathcal{C}}_v) \mid \overline{\mathcal{C}}_v \in \overline{\mathcal{C}}\}$ covers $\mathcal{X}^{\text{GS}}_{\text{unsafe}}$. Again, fix $p \in \mathcal{X}^{\text{GS}}_{\text{unsafe}}$. Since $\mathcal{V}^+$ covers $\mathcal{X}^{\text{GS}}_{\text{unsafe}}$, there exists a voxel $v_p \in \mathcal{V}^+$ containing p. Since $\overline{\mathcal{C}}$ covers $\mathcal{V}^+$, there exists a $\overline{\mathcal{C}}_v \in \overline{\mathcal{C}}$ such that $v_p \in \overline{\mathcal{C}}_v$, so $v_p \subseteq \text{Hull}(\overline{\mathcal{C}}_v)$. Thus, $p \in v_p \subseteq \text{Hull}(\overline{\mathcal{C}}_v)$, so $\overline{\mathcal{H}}$ covers $\mathcal{X}^{\text{GS}}_{\text{unsafe}}$.

Since $\overline{\mathcal{H}^N} := \overline{\mathcal{H}}$, to complete the proof, it suffices to prove that if $\overline{\mathcal{H}^{n+1}}$ covers $\mathcal{X}^{\text{GS}}_{\text{unsafe}}$ for some $n \in \{1, \cdots, N-1\}$, then $\overline{\mathcal{H}^n}$ covers $\mathcal{X}^{\text{GS}}_{\text{unsafe}}$. Fix $p \in \mathcal{X}^{\text{GS}}_{\text{unsafe}}$ arbitrarily, and let a voxel cluster $\overline{\mathcal{C}_i^{n+1}} \in \overline{\mathcal{C}^{n+1}}$, for some $i \in I^{N-n-1}$, be given such that $p \in \text{Hull}(\overline{\mathcal{C}_i^{n+1}})$. During the process of constructing $\overline{\mathcal{H}^n}$ from $\overline{\mathcal{H}^{n+1}}$, either $\overline{\mathcal{C}_i^{n+1}}$ will be selected to merge (with $\overline{\mathcal{C}_j^{n+1}}$, for some $j \in I^{N-n-1} \backslash \{i\}$), or it will not. In the former case, we have $p \in \text{Hull}(\overline{\mathcal{C}_i^{n+1}}) \subseteq \text{Hull}(\overline{\mathcal{C}_i^{n+1}} \cup \overline{\mathcal{C}_j^{n+1}})$, with $\overline{\mathcal{C}_i^{n+1}} \cup \overline{\mathcal{C}_j^{n+1}} \in \overline{\mathcal{C}^n}$. In the latter case, we have $p \in \text{Hull}(\overline{\mathcal{C}_i^{n+1}})$ with $\overline{\mathcal{C}_i^{n+1}} \in \overline{\mathcal{C}^n}$. Thus, in both cases, there exists a cover set in $\overline{\mathcal{C}^n}$ which contains p, so $\overline{\mathcal{C}^n}$ covers $\mathcal{X}^{\text{GS}}_{\text{unsafe}}$. ■