

Splat-Nav: Safe Real-Time Robot Navigation in Gaussian Splatting Maps

Timothy Chen^{1*}, Ola Shorinwa^{1*}, Joseph Bruno³, Javier Yu¹, Weijia Zeng², Keiko Nagami¹, Philip Dames³, Mac Schwager¹

Abstract—We present Splat-Nav, a real-time navigation pipeline designed to work with environment representations generated by Gaussian Splatting (GSplat), a popular emerging 3D scene representation from computer vision. Splat-Nav consists of two components: 1) Splat-Plan, a safe planning module, and 2) Splat-Loc, a robust pose estimation module. Splat-Plan builds a safety-by-construction polytope corridor through the map based on mathematically rigorous collision constraints and then constructs a Bézier curve trajectory through this corridor. Splat-Loc provides a robust state estimation module, leveraging the point-cloud representation inherent in GSplat scenes for global pose initialization, in the absence of prior knowledge, and recursive real-time pose localization, given only RGB images. The most compute-intensive procedures in our navigation pipeline, such as the computation of the Bézier trajectories and the pose optimization problem run primarily on the CPU, freeing up GPU resources for GPU-intensive tasks, such as online training of Gaussian Splats. We demonstrate the safety and robustness of our pipeline in both simulation and hardware experiments, where we show online re-planning at 5 Hz and pose estimation at about 25 Hz, an order of magnitude faster than Neural Radiance Field (NeRF)-based navigation methods, thereby enabling real-time navigation. Videos are hosted on our website, and code can be found here.

Index Terms—Vision-Based Navigation, Collision Avoidance, Localization.

I. INTRODUCTION

Autonomous robotic operation requires robots to plan safe paths to reach a desired goal location as well as to localize themselves during navigation to perform closed-loop control to follow the planned path. These two areas of planning and localization are core to mobile robotics and robotic manipulation, and the specific solutions for each problem depend upon the representation used for the map of the environment. Traditionally, these representations have included occupancy grids [1], (watertight) triangular meshes [2], point clouds [3], and Signed Distance Fields (SDFs) [4], all of which provide well-defined geometry. Many planning and localization algorithms have been designed for these representations (see Section II-A). However, these explicit scene representations are

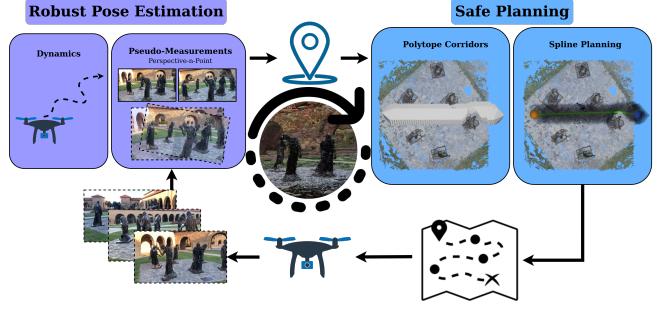


Fig. 1. Our real-time, robust navigation pipeline, Splat-Nav, consists of a safe planning module, Splat-Plan, and robust localization module, Splat-Loc, both powered by the Gaussian Splatting environment representation. We first represent our agent as an ellipsoid, and then rapidly solve for smooth paths using an ellipsoid-to-ellipsoid collision test. We then use the point cloud representation from the Gaussian Splat map to perform both global camera localization from RGB-(D) images using point cloud alignment, and online pose tracking using only RGB images with a PnP image-to-point cloud pose refinement. Additionally, we also learn a semantic channel to enable semantically-conditioned goal locations like “go to the table”.

generally constructed at limited resolutions, to enable real-time operation, leaving out potentially-important scene details that could be valuable in planning and localization problems.

Implicit scene representations such as Neural Radiance Fields (NeRFs) [5] address these fundamental limitations. NeRFs, which were introduced by the computer vision community, represents the environment as a volumetric density and view-dependent color using a multilayer perceptron (MLP) deep neural network. NeRFs offer several potential benefits over traditional scene representations: they provide more photo-realistic renderings, they can be trained using only monocular RGB images, and current implementations of NeRFs can be trained in seconds [6, 7]. Although some NeRF methods can be trained quickly, these methods generally require access to high-performance GPUs, which might not be available in many robotics tasks. We note that some planning and localization algorithms for NeRF maps exist (see Section II-B); however, these methods do not run in real-time, in general. Moreover, the use of MLPs makes scene rendering computationally expensive and makes it difficult to use NeRFs for standard robotics tasks in an interpretable way, especially for tasks that require interacting with the environment.

More recently, Gaussian Splatting (GSplat) [8] has emerged as a viable scene representation compared to NeRFs, representing the environment as a collection of Gaussians associated with a position (mean), spatial extents, color, and opacity. Compared

* The co-first authors contributed equally.

† This work was supported in part by ONR grant N00014-23-1-2354 and DARPA grant HR001120C0107 and NSF grant 2220866. Toyota Research Institute provided funds to support this work. T. Chen was supported by a NASA NSTGRO Fellowship.

¹ Stanford University, Stanford, CA 94305, USA
{chengine, shorinwa, javieryu, knagami, schwager}@stanford.edu

²University of California San Diego, San Diego, CA 92093, USA.
wez195@ucsd.edu

³Temple University, Philadelphia, PA 19122, USA, {brunoj6, pdames}@temple.edu

to NeRFs, GSplat generates more photorealistic renderings, has shorter or comparable training times, and has much faster rendering by taking advantage of differentiable rasterization techniques for 3D Gaussians on the GPU. More importantly for robotics, GSplats, unlike NeRFs, offer a geometrically consistent collision geometry, enabling us to use level sets of these Gaussians to generate an ellipsoidal representation of the scene (see Section III for more details). These interpretable geometric primitives facilitate the development of rigorous motion planning algorithms that are safe, robust, and real-time.

A. Approach

In this paper, we develop a framework called *Splat-Nav*, outlined in Fig. 1, for robot trajectory planning and pose estimation that is designed to work with the GSplat representation. The first component, *Splat-Plan*, generates safe trajectories through the GSplat scene. The primary contribution is an efficient navigation pipeline for Gaussian Splatting Maps, built upon a computationally efficient collision checker (which harnesses GPU parallelization to enable real-time collision checking) and a safe planning framework. The collision checker extends the method in [9] for ellipsoid-to-ellipsoid collision checking by eliminating computationally-intensive root-finding procedures required in the method through the formulation of a generalized eigenvalue problem to yield a significant computational speed-up. We implement the collision-checking algorithm with efficient GPU parallelization using sampling to obtain massive speedups for checking millions of ellipsoids collisions per second. Moreover, we integrate safe planning tools into our proposed planning pipeline through the computation of safe polytopes obtained from the decomposition of the configuration space into occupied and collision-free regions. We compute Bézier spline trajectories for the robot within these safe polytopes for navigation.

The second component, *Splat-Loc*, allows the robot to localize itself within the GSplat map using monocular images from its onboard camera. The primary contribution is the derivation of a real-time pose estimation module, which leverages the real-time rendering speeds of GSplats for fast extraction of point clouds for image-based localization. We introduce a robust image-to-point cloud pose estimation pipeline, built on well-established computer vision tools, harnessing the visual and geometric features of images to compute an estimate of the robot’s pose, without having to rely on more traditional gradient-based methods. In addition, we present our pose estimator within an optimization-based paradigm, enabling the seamless integration of prior knowledge from the planning module, as well as scene-relevant constraints. Moreover, in the absence of a prior guess on the pose of the robot, we provide a global pose estimation module, utilizing point-cloud registration algorithms, provided an RGB-D camera is available. Both modules are many times faster than their existing NeRF-based counterparts, which do not provide guarantees on performance [10].

To showcase these two modules, we provide simulation and hardware results to show that our pipeline: (1) generates safe, but not overly conservative, trajectories through the environment, (2) robustly localizes a robot in the scene, and

(3) can be executed in real-time. Our planner demonstrates safe flight over a variety of synthetic and real-world GSplat maps. We find that our localization is more accurate, faster, and fails less often compared to baselines. We demonstrate re-planning at 5 Hz and online pose estimation at about 25 Hz on a desktop computer, enabling real-time navigation.

B. Contributions

The key contributions of this paper are as follows: 1) We derive an accelerated collision checking algorithm for the potentially millions of ellipsoids that make up GSplat scenes, parallelized on a GPU for real-time collision checking (Proposition 1, Corollary 1, Corollary 2). 2) We generate safe polytope corridors from the ellipsoidal representation of GSplat maps (Proposition 2). 3) We implement a receding horizon trajectory planning algorithm based on the flight corridors (Sec. IV-D). 4) We propose a global pose initialization and a UKF-based filter for state estimation from on-board vision, by adapting existing point cloud alignment and Perspective-n-Point (PnP) algorithms to the GSplat representation (Sec. V). 5) We show significantly improved computational speed and lower tracking errors in comparison to existing state estimation algorithms for NeRFs and GSplat maps (Tables I and II, and Fig. 7). 6) We demonstrate real-time operation in hardware tests on a quadrotor drone (Sec. VI).

II. RELATED WORK

Here, we review the related literature in robot planning and localization with different map representations, which we categorize into three groups: traditional representations (e.g., occupancy grids, meshes, point clouds, and SDFs), NeRFs, and GSplat.

A. Traditional Map Representations

1) *Planning*: There is a long history of planning within robotics. We refer readers to [11] for an excellent explanation of the major algorithms in this realm. Most relevant for this work are the graph-based planners (e.g., A^{*}), which compute a path over a grid representation of the environment; sampling-based planners (e.g., PRM [12], RRT and RRT* [13]), which generate a path by sampling candidate states within the configuration space of the robot; and trajectory optimization-based planners (e.g. CHOMP [14] and Traj-Opt [15]), which take an optimization-based approach to planning. The Open Motion Planning Library (OMPL) [16] provides open-source implementations of many of these algorithms, which we use later for benchmarking. Prior work in [17] utilizes an optimization-based approach in path planning, taking a point cloud representation of the environment and converting this into a set of safe polytopes. The resulting safe polytopes are utilized in computing a safe trajectory, parameterized as a spline from a quadratic program (QP), which can be efficiently solved.

There is also extensive literature on planning based on onboard sensing. Typically, these works present reactive control schemes [18, 19], using the sensed depth directly to perform

collision checking in real-time. These methods typically are myopic, reasoning only locally about the scene. Consequently, such methods often converge to local optima, preventing the robot from reaching its goal, especially in cluttered/intricate environments. An alternate approach is to construct a map of the environment using the depth measurements. Often, a Signed Distance Field (SDF) or its truncated variant (TSDF) is constructed from depth data [20, 21], which is often encoded within a voxel representation. Such a representation is typical in dynamic robotic motion planning, providing fast collision checking and gradients in planning.

2) *Localization*: Prior works in robot localization utilize filtering schemes, such as Extended Kalman Filters (EKFs) [22, 23], Particle Filters (PFs) [24, 25], and other related filters [26, 27], to solve the pose localization problem. These methods generally estimate the pose of the robot from low-dimensional observations (measurements), extracted from the high-dimensional observations collected by the robot’s onboard sensors, such as cameras. This approach often fails to leverage the entire information available in the raw, high-dimensional measurements. Learning-based filtering methods [28, 29] seek to address this limitation using deep-learning to develop end-to-end frameworks for localization, computing a pose estimate directly from raw camera images. Although learning-based approaches can be quite effective given sufficient training data, these methods are often limited to a single robot platform (dynamics model), and thus require separate filters for each robot or environment.

B. NeRF Maps

1) *Planning*: NeRFs implicitly represent the environment as a spatial density field (with color) [5]. Using this representation, NeRF-Nav [10] plans trajectories for differentially flat robots, e.g., quadrotors, that minimize a total collision cost. Further, CATNIPS [30] converts the NeRF into a probabilistic voxel grid and then uses this to generate trajectories parameterized as Bézier curves. The work in [31] uses the predicted depth map at sampled poses to enforce step-wise safety using a control barrier function. The above works are complementary, with [10, 30] serving as high-level planners that encourage non-myopic behavior while [31] can be used as a safety filter for a myopic low-level controller.

2) *Localization*: There is some existing work on tracking the pose of a robot equipped with an on-board camera and IMU through a pre-trained NeRF map. iNeRF [32] does this for single images, and NeRF-Nav [10] and Loc-NeRF [33] both track a trajectory using a sequence of images. Other works consider simultaneous localization and mapping (SLAM) using a NeRF map representation. Existing methods such as [32, 34] all simultaneously optimize the NeRF weights and the robot/camera poses. NeRF-SLAM [35] proposes a combination of an existing visual odometry pipeline for camera trajectory estimation together with online NeRF training for the 3D scene.

C. GSplat Maps

To the best of our knowledge, there are no planning algorithms designed to operate in GSplat environments. There

are a few recent works on SLAM using a GSplat representation of the environment [36, 37, 38]. However, none of these works have a standalone real-time pose localization component, and none address safe trajectory planning or control through the GSplat map, as is our focus in this paper.

III. 3D GAUSSIAN SPLATTING

We present a brief introduction to 3D Gaussian Splatting [8], a radiance field method for deriving volumetric scene representations. In contrast to explicit 3D scene representations (e.g., point clouds and meshes) and continuous scene representations (e.g., NeRFs), Gaussian Splatting represents non-empty space in a scene using 3D Gaussian primitives, each of which is parameterized by a mean $\mu \in \mathbb{R}^3$ (defining its position), covariance matrix $\Sigma \in \mathbb{S}_{++}$ (related to its spatial extent), opacity $\alpha \in [0, 1]$, and spherical harmonics (SH) coefficients (defining view-dependent colors). For better numerical optimization, the anisotropic 3D covariance of each Gaussian is written as: $\Sigma = RSS^T R^T$, where $R \in \text{SO}(3)$ is a rotation matrix (parameterized by a quaternion) and S is a diagonal scaling matrix (parameterized by a 3D vector). The number of primitives, along with the coefficients for each primitive, are learned using monocular images (the same as NeRF methods) along with a sparse point-cloud representation of the environment, which is generally computed from structure-from-motion [39].

The anisotropic covariance of the Gaussian primitives along with adaptive density control enable the computation of compact high-quality representations, even in complex scenes, unlike many state-of-the-art point-based rendering methods. Further, 3D Gaussian Splatting obviates the need for volumetric ray-marching required in NeRF methods, enabling high-quality real-time rendering, even from novel views. Moreover, 3D Gaussian Splatting enables relatively fast extraction of a mesh representation of the scene from the Gaussian primitives. In Fig. 2, we visualize the ground-truth mesh, the GSplat mesh, and the associated point cloud extracted from a NeRF of a simulated Stonehenge scene. We note that the resulting GSplat mesh has a smaller Chamfer distance (0.031 with 3M vertices) compared to the NeRF point cloud (0.081 with 4M points) despite having fewer points. We note that the NeRF does not necessarily yield a view-consistent geometry due to volumetric rendering, especially when the point cloud is not post-processed to remove outliers.

To render an image from a given camera pose, the 3D Gaussians are projected onto the image plane using an affine approximation of the projective transformation, given by $\Sigma_{2D} = JW\Sigma W^T J^T$, with Jacobian J and viewing transformation W . Subsequently, the color C of each pixel is computed via α -blending of the splats, given by:

$$C = \sum_{i \in \mathcal{N}} c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j), \quad (1)$$

where c_i represents the color of Gaussian i , computed from the Spherical Harmonics model, \mathcal{N} represents the set of Gaussians whose projection intersects with the pixel. We overload the definition of α_i in the rendering model to represent

the opacity of Gaussian i multiplied by the unnormalized probability density function associated with each Gaussian in the 2D image plane, given by: $G(x) = e^{-\frac{1}{2}(x-\mu)^T \Sigma_{2D}^{-1}(x-\mu)}$, with $\Sigma_{2D} \in \mathbb{S}_{++}$ representing the 2D covariance matrix. We note the rendering model (1) corresponds to the rendering model used in NeRF methods [5], although Gaussian Splatting utilizes differentiable tile-based rasterization, contributing to its high rendering speed.

Remark 1. *The original work only projects 3D Gaussians whose 99% confidence interval intersects the view frustum of a camera, effectively restricting the scene representation to the 99% confidence ellipsoid associated with each Gaussian. Consequently, the union of the 99% confidence ellipsoids represents the entirety of the geometry of the scene learned during the training procedure.*

The Gaussian Splatting representation is trained via stochastic gradient descent with a loss function comprising of the photometric loss between the rendered and ground-truth images and the structural-similarity (SSIM) index loss, while simultaneously adaptively controlling the number of Gaussians in the representation. Generally, Gaussian Splatting requires about the same training time as state-of-the-art NeRF methods, while achieving about the same or better quality. However, Gaussian Splatting achieves real-time rendering speeds [8], unlike NeRFs. See Fig. 2 for an example showing a NeRF and GSplat map of the same scene. In the subsequent sections, we present the core contributions of our work in deriving an efficient navigation pipeline for robots, describing how we leverage 3D Gaussian Splatting as the underlying scene representation, which we augment with semantic embeddings for goal specification in natural-language. Specifically, we present our planning and pose estimation pipelines that enable safe navigation in Gaussian Splatting environments.



Fig. 2. Visualization of a point cloud from a NeRF and a mesh from a Gaussian Splat in the synthetic scene Stonehenge. The Chamfer Distance between the NeRF and ground-truth is 0.081 (with 4M points). The Chamfer distance between the GSplat and ground-truth is 0.031 (with 3M vertices). The collision geometry (especially the foreground) of the GSplat is better and can be extracted instantaneously from the model parameters compared to the costly rendering procedure from many viewpoints to create a point cloud from the NeRF.

IV. PLANNING IN GAUSSIAN SPLATTING MAPS

Now, we present our planning pipeline based on Gaussian Splatting, called Splat-Plan. The pipeline consists of generating safe polytopic corridors (inspired by [17]) that discretize the free space of a GSplat Map, for navigation from an initial configuration to a goal configuration, rigorously built on theory derived from tests for intersection between ellipsoids. The

method is fast enough to provide real-time operation, provides safety guarantees extending to any scene with a pre-trained GSplat representation, and is not overly-conservative. We stress that, as with any safety guarantee on a map, our ultimate safety rests on the completeness of the map. If the map does not reflect the presence of an obstacle, our method may collide with the obstacle—we cannot avoid what we cannot see. In practice, we observe that GSplat maps provide fast and efficient representations of the underlying ground truth geometry, as validated in our hardware experiments.

Before presenting the planning problem, we make the following assumptions on the representations of the robot \mathcal{R} and the map \mathcal{G} considered in this work. We assume that the robot is represented by a union of the ellipsoids in the non-empty set $\{\mathcal{E}_{\mathcal{R},i}\}_{i=1}^d$, where d denotes the cardinality of the set, i.e., $\mathcal{R} \subseteq \cup_{i=1}^d \mathcal{E}_{\mathcal{R},i}$. For simplicity, we consider a singleton set $\mathcal{E}_{\mathcal{R}}$, noting that the subsequent discussion applies directly to the non-singleton case by running the collision check for all robot ellipsoids. One can also convert a mesh or point cloud of a robot to an ellipsoid by finding the minimal bounding ellipsoid (or sphere).

We represent non-empty space in the environment with $\gamma\%$ confidence ellipsoids obtained from the GSplat map, as discussed in Remark 1, given by:

$$\mathcal{E}_j = \{x \in \mathbb{R}^3 \mid (x - \mu_j)^T \Sigma_j^{-1} (x - \mu_j) \leq \chi_3^2(\gamma)\}, \quad (2)$$

where $\mu_j \in \mathbb{R}^3$ denotes the mean of Gaussian j , $\Sigma_j \in \mathbb{S}_{++}$ denotes its covariance matrix, and $\chi_3^2(\gamma)$ denotes the γ th percentile of the chi-square distribution with three degrees of freedom. The union of these ellipsoids, given by $\mathcal{G} = \{\mathcal{E}_j\}_{j=1}^N$, defines the occupied space in the environment. To simplify notation, we express the ellipsoid in (2) in standard form: $\mathcal{E}_j = \{x \in \mathbb{R}^3 \mid (x - \mu_j)^T \Sigma_j^{-1} (x - \mu_j) \leq 1\}$, where we overload notation with $\Sigma_j := \chi_3^2(\gamma) \Sigma_j$. Based on Remark 1, we set $\gamma = 0.99$ to be safe with respect to the entirety of the supervised scene.

Remark 2 (Online Gaussian Splatting). *Our planning algorithm requires a GSplat map. This map can be constructed a priori or online using real-time SLAM methods for radiance fields [38, 37, 36], which is a very new and active area of research.*

Remark 3 (Handling Uncertainty of the Scene Representation). *We can vary the value of γ (from that used during the training procedure) based on the quality of the GSplat Map and uncertainty in different regions of the GSplat Map. In general, larger values of γ inflate the volume of the confidence ellipsoids associated with each Gaussian, resulting in greater safety margins and more conservative planning. The converse holds if smaller values of γ are selected. Moreover, for simplicity, we utilized a uniform value of γ . However, the value of γ can vary among the ellipsoids, allowing the planner to account for varying levels of uncertainty in different regions of the GSplat Map. Likewise, the volume of the ellipsoid representing the robot can be increased/decreased to account for uncertainty in the pose of the robot.*

Remark 4 (Dynamic Scenes). *We limit our discussion to planning in static scenes. However, we note that our method readily applies to planning in dynamic scenes, under the assumption that a dynamic Gaussian Splatting scene representation can be constructed. We discuss more about planning in dynamic scenes in Section VIII.*

A. Problem Statement

Given a bounding ellipsoid \mathcal{E}_R for the robot and a GSplat Map \mathcal{G} , we seek to find a feasible path for a robot to navigate from an initial configuration X to a specified goal configuration Y , such that there are no collisions, i.e., $\mathcal{E}_R \cap \mathcal{E}_j = \emptyset, \forall \mathcal{E}_j \in \mathcal{G}$.

B. Collision Detection

We leverage the ellipsoidal representations of the robot and the environment to derive an efficient collision-checking algorithm, based on [9], where we take advantage of GPU parallelization for faster computation. We build upon [9] rather than on other existing ellipsoid-to-ellipsoid intersection tests, because of its amenability to significant GPU parallelization. We do not utilize the GJK algorithm [40], since we do not require knowledge of the distance between the two ellipsoids. For completeness, we restate the collision-checking method from [9, Proposition 2].

Theorem 1. *Given two ellipsoids $\mathcal{E}_a, \mathcal{E}_b$ (with means μ_a, μ_b and covariances Σ_a, Σ_b) and the concave function $K : (0, 1) \rightarrow \mathbb{R}$,*

$$K(s) = (\mu_b - \mu_a)^T \left[\frac{1}{1-s} \Sigma_a + \frac{1}{s} \Sigma_b \right]^{-1} (\mu_b - \mu_a),$$

$\mathcal{E}_a \cap \mathcal{E}_b = \emptyset$ if and only if there exists $s \in (0, 1)$ such that $K(s) > 1$.

Note that while K is concave in s (i.e., $-K$ is convex in s), it is convex with respect to the means and variances. Theorem 1 is a complete test that will always indicate whether two ellipsoids are in collision or not. We note, however, that solving the feasibility problem in Theorem 1 can be challenging, particularly in large-scale problems, where the feasibility problem has to be solved for many pairs of ellipsoids with an associated matrix inversion procedure in each problem. In general, Gaussian Splatting environments consists of hundreds of thousands to tens of millions of Gaussians [8]. Consequently, we eliminate the matrix inversion by operating in a shared basis for both Σ_A and Σ_B , for faster collision-checking, detailed in the following Proposition.

Proposition 1. *By solving the generalized eigenvalue problem for Σ_a and Σ_b , we obtain generalized eigenvalues λ_i and the corresponding matrix of generalized eigenvectors ϕ . Letting $v = \phi^T(\mu_a - \mu_b)$, Theorem 1 can then be simplified to*

$$K(s) = v^T \text{diag} \left(\frac{s(1-s)}{1+s(\lambda_i-1)} \right) v.$$

Proof. We present the proof in Appendix A. \square

In the following corollary, we discuss a sampling technique more suitable for faster collision-checking.

Corollary 1. *Executing the collision test in Theorem 1 via sampling for $s \in (0, 1)$ instead of analytically solving Theorem 1 will always yield safe behavior, never yielding that $\mathcal{E}_a \cap \mathcal{E}_b = \emptyset$ when the ellipsoids are intersecting.*

Proof. We provide the proof in Appendix B. \square

The intersection test from Proposition 1 can now be parallelized by sampling many points between 0 and 1 and finding if any yield $K(s) > 1$. In particular, our planner leverages GPU parallelization across batches of ellipsoids, leading to massive computation speedups.

Remark 5 (Comparison of Ellipsoid-to-Ellipsoid Intersection Tests). *We note that there exists many methods for detecting if a pair of ellipsoids intersect. In addition to Proposition 1, we derive an alternative intersection test for ellipsoids from Theorem 1, based entirely on matrix multiplications, without requiring the solution of an eigenvalue problem, which we present in Appendix C. We examine the computation time required by these methods, across scenes with varying numbers of ellipsoids. The results indicate that our proposed test in Proposition 1 outperforms relevant existing methods with respect to the computation time, and scales efficiently to dense scenes.*

Corollary 2. *Given a Gaussian Splatting representation with $\Sigma_j = RSS^T R^T$, let $SST = \text{diag}(\lambda_i)$ and let $w = R^T(\mu_R - \mu_j)$. If we choose to parameterize our robot body as a sphere with covariance $\Sigma_R = \kappa I$, then the intersection test can be simplified to:*

$$K(s) = w^T \text{diag} \left(\frac{s(1-s)}{\kappa + s(\lambda_i - \kappa)} \right) w.$$

Note that this test can be faster to query than in Proposition 1 as there is no need to perform an eigenvalue decomposition. For non-spherical robots, one can approximate it as a sphere by setting κ to be the maximum eigenvalue of Σ_R (i.e., $\kappa = \lambda_{\max}(\Sigma_R)$).

Although performing the collision test in Corollary 2 is fast, it is difficult to naively scale to test all ellipsoids in the scene while maintaining real-time performance in a planner. As a result, we rely on the cost-effectiveness of K-D tree queries to test ellipsoids just in the vicinity of a robot. To find a small set of ellipsoids that contains all possible ellipsoid-robot collisions conditioned on the robot's position, we refer readers to Appendix D.

C. Computing Safe Polytopes

We like to again emphasize that having convex primitives (ellipsoids) as an environment representation facilitates the development of interpretable algorithms for planning. This is especially true in the construction of safe trajectories within convex safe polytopes, which define obstacle-free regions of the robot's configuration space. We build upon prior work on convex decomposition of configuration spaces such as [41, 17]. In this work, we leverage the ellipsoidal primitives to create polytopes that define the safe regions of space through the use of supporting hyperplanes. The ellipsoidal representation

of the environment obtained from GSplat enables the direct computation of these convex obstacle-free regions without the need for a convex optimization procedure. Moreover, while some existing methods [17] require the specification of an initial, safe path to create the polytopes, our proposed method can generate safe polytopes even from an unsafe initial path. Furthermore, our method is fast enough to run in real time. In the following proposition, we describe the generation of safe polytopes for a given robot.

Proposition 2. *Given a test point x^* (i.e., robot position μ_R) and a collision test set \mathcal{G}^* , for every ellipsoid in \mathcal{G}^* , a supporting hyperplane to the j -th ellipsoid (considering the extent of the robot body) derived from Proposition 1, for any $\epsilon > 0$, is given by*

$$\Delta_j^T Q_j x \geq (1 + \epsilon)k_j + \Delta_j^T Q_j \mu_j,$$

where $\Delta_j = x^* - \mu_j$, $Q_j = \phi \text{ diag} \left(\frac{\bar{s}(1-\bar{s})}{1+\bar{s}(\lambda_i-1)} \right) \phi^T$, and $k_j^2 = K(\bar{s}) = \Delta_j^T Q_j \Delta_j > 0$, for $\bar{s} \in (0, 1)$. We note that we can define \bar{s} in Q_j as the maximizer of $K(s)$, with $\bar{s} := \arg \max_{s \in (0,1)} K(s)$. Further, note that Q_j is always symmetric and positive-definite. By stacking the hyperplane constraints (a_j, b_j) , we arrive at a polytope $Ax \geq b$ that is guaranteed to be safe, where $a_j = \Delta_j^T Q_j$ and b_j is given by the right-hand-side of the inequality.

Proof. We provide the proof in Appendix E. \square

This allows us to use the polytope generation with our parallelized sampling method in a safe way. Additionally, we note that the polytopes apply to general ellipsoidal robot bodies conditioned on knowing the ellipsoid, as we never relied on the structure of Q_j beyond it being symmetric and positive-definite. In other words, the robot does not have to be spherical in Proposition 2. We believe that these findings related to ellipsoid-based environment representations may have an impact beyond the scope of safety, such as in graphics.

D. Generating Safe Paths

We present Splat-Plan, derived using the safe flight corridor method from [17], leveraging Proposition 2 to generate safe polytopes. We first create a 3D binary occupancy grid using a point cloud of the Gaussians. To do this, we sample the Fibonacci sphere and apply rotation, scaling, and translation to the points per Gaussian. From this grid, we use A* to generate a candidate path. Each point on the A* path almost always has sufficient free space in its neighborhood, and will not be stuck in corners since the path is fully connected from start to goal. However, there may be cases where the union of polytopes queried on the A* is disconnected, leading to an infeasible trajectory, which may happen when the A* path goes through walls of ellipsoids. To remedy this, we execute A* multiple times, running our collision test from Corollary 2 for intersections and updating our occupancy grid accordingly. In this way, the final A* path is guaranteed to converge to a safe path given a sufficiently small grid discretization.

Since the polytopes generated from safe points necessarily contain those points, our method sufficiently guarantees the

generation of a connected polytopic corridor in the limit of the grid resolution. Given a connected set of L polytopes (where polytope ℓ is parametrized by matrix A^ℓ and vector b^ℓ) from the initial configuration x_0 of the robot to its desired configuration x_f , we compute a set of L Bézier curves (parametrized by $M+1$ control points c_i^ℓ and Bernstein basis $\beta_i(t)$ with normalized time $t \in [0, 1]$) representing the trajectory of the robot using the path-length minimization problem:

$$\min_{c_i^\ell, x^\ell(t)} \sum_{\ell=1}^L \sum_{i=0}^{M-1} \|c_{i+1}^\ell - c_i^\ell\|_2^2 \quad (3a)$$

subject to

$$\text{Safety: } A^\ell c_i^\ell \leq b^\ell, \ell = 1, \dots, L; i = 0, \dots, M \quad (3b)$$

$$\text{Configuration: } x^0(0) = x_0 \quad (3c)$$

$$x^L(1) = x_f$$

$$\text{Continuity: } x^\ell(1) = x^{\ell+1}(0), \ell = 1, \dots, L-1 \quad (3d)$$

$$\text{Bézier curve: } x^\ell(t) = \sum_{i=0}^M \beta_i(t) c_i^\ell, \ell = 1, \dots, L-1 \quad (3e)$$

$$\text{Dynamics: } x^\ell(t+1) = f(x^\ell(t), u). \quad (3f)$$

Without the dynamics constraints (3f), the optimization problem reduces to a quadratic program that can be solved in real-time, producing a trajectory that can be tracked by differentially-flat robots. Moreover, due to the convex hull property of Bézier curves, constraining the control points to lie in the polytopes ensures that all points along the curves will lie in the corridor and hence guarantees safety in the continuum.

The generated sequence of polytopes using Proposition 2 may contain many redundant halfspaces, and there may be many polytopes in the corridor, so we find a smaller set of connected polytopes. We first solve for a smaller halfspace representation. Then, we take a model predictive control (MPC) perspective, planning a trajectory over a receding time horizon and replanning before the robot reaches the end of this partial trajectory. Using the same set of polytopes (which are quick to compute), our MPC problem uses the next N polytopes and plans a path to the A* waypoint in the N -th polytope in the future (Fig. 3). This greatly reduces the number of constraints in the trajectory optimization problem, allowing us to achieve real-time performance. Note that by taking N to be the length of the A* path, we recover the solution to the full-length trajectory.

V. POSE ESTIMATION

In this section, we present our pose estimation module, Splat-Loc, for localizing a robot in a Gaussian Splatting representation of its environment. First, we consider the case when an initial guess of the robot's pose is available, before presenting approaches for dealing with cases where no prior information on its pose is available.

A. Problem Formulation

We perform pose estimation on the SE(3) manifold, which represents rigid transformations in 3D space. A pose in SE(3) is parameterized by a rotation matrix $R \in \text{SO}(3)$ and



Fig. 3. Left: GSplat render of the scene. Middle: Generated safe polytope corridor colored with a gradient, with each successive polytope being a slightly different color. The corridor is visualized in a “meshed” version of the ellipsoids from GSplat. Right: Finite horizon planned path (inflated to represent geometry of agent) through the corridor.

a translation vector $\rho \in \mathbb{R}^3$. Our approach leverages the lie algebra $\mathfrak{se}(3)$ associated with the $\text{SE}(3)$ matrix lie group. $\mathfrak{se}(3)$ consists of a vectorspace of matrices in $\mathbb{R}^{4 \times 4}$ spanned by a basis of six generators [42]. We parameterize each element of $\mathfrak{se}(3)$ by $\xi \in \mathbb{R}^6$, where $\xi = [r^T, p^T]^T$, $p, r \in \mathbb{R}^3$, with p representing the translational component and r the rotational component. We assume that the state of the robot is given by $x \in \mathbb{R}^{12}$, comprising of the lie algebra ξ and the linear and angular velocities, $\nu \in \mathbb{R}^3$ and $\omega \in \mathbb{R}^3$, respectively, with $x = [\xi^T, \nu^T, \omega^T]^T$. We assume that the discrete-time dynamic model $f : \mathbb{R}^n \mapsto \mathbb{R}^n$ of the robot is of the form: $x_t = f(x_{t-1}, u_{t-1}, w_{t-1})$, where $w_t \in \mathbb{R}^n$ denotes Gaussian white noise at time t with covariance $Q_t \in \mathbb{R}^{n \times n}$ and $u_t \in \mathbb{R}^m$ denotes the control inputs of the robot.

The measurement model $h : \mathbb{R}^n \times \mathbb{R}^k \mapsto \mathbb{R}^m$ of the robot can be expressed in the form: $y_t = h(x_t, v_t)$, where $y_t \in \mathbb{R}^m$ denotes the measurement and $v_t \in \mathbb{R}^n$ denotes Gaussian white noise at time t with covariance $R_t \in \mathbb{R}^{n \times n}$. In general, the measurements y_t obtained by the robot can include the robot’s pose, linear velocities, and its body rates. In this work, we consider an RGB (or RGB-D) camera-based measurement model that uses keypoints from an image $I_t \in \mathbb{R}^{H \times W \times 3}$ to measure elements from the robot’s state. Let $\psi : \mathbb{R}^{H \times W \times 3} \rightarrow \mathbb{R}^k$ denote the function that goes from a raw camera image to a set of keypoints with descriptors $\{k_{t,i}\}_i = \psi(I_t)$. Based on the state of the robot x_t , we can also render an image $\hat{I}_t(x_t)$ from the GSplat map and use ψ to obtain another set of keypoints $\{\hat{k}_{t,j}\}_j = \psi(\hat{I}_t(x_t))$. Note, each keypoint \hat{k}_t has an associated 3D position from the GSplat map. The measurement function h can then use matches between the sets of keypoints $\{k_{t,i}\}_i$ and $\{\hat{k}_{t,j}\}_j$ to, for example, estimate the pose by solving the Perspective-n-Point (PnP) problem or estimate the velocities using optical flow (see Section V-D for more detail). We note that our discussion extends directly to other measurement models.

B. MAP Pose Estimate

We formulate a constrained *maximum a-posteriori* (MAP) optimization problem to compute an estimate of the robot’s pose that satisfies the safety constraints (defined by the polytope) at each timestep. We assume we have a prior estimate of the pose at time $t - 1$ represented by a Gaussian distribution

$\mathcal{N}(\mu_{t-1|t-1}, P_{t-1|t-1})$ and know the current safe polytope containing the robot, which is parameterized by (A_t, b_t) , and the polytope at the previous timestep (A_{t-1}, b_{t-1}) . Given the dynamics and measurement models, we can compute the new estimate of the robot’s pose that maximizes the posterior distribution of the robot’s pose (given the set of measurements) from:

$$\begin{aligned} \min_{x_t, x_{t-1}} & \|x_t - f(x_{t-1}, u_{t-1})\|_{Q_{t-1}^{-1}}^2 + \|\psi(I_t) - h(x_t)\|_{R_t^{-1}}^2 \\ & + \|x_{t-1} - \mu_{t-1|t-1}\|_{P_{t-1|t-1}^{-1}}^2, \end{aligned} \quad (4a)$$

subject to

Prior Knowledge at time t :

$$\left\| A_{t,(i)}^T \right\|_{\bar{P}_{t|t-1}}^2 + A_{t,(i)} x_t \leq b_{t,(i)}, \quad \forall i, \quad (4b)$$

Prior Knowledge at time $t - 1$:

$$\left\| A_{t-1,(i)}^T \right\|_{\bar{P}_{t-1|t-1}}^2 + A_{t-1,(i)} x_{t-1} \leq b_{t-1,(i)}, \quad \forall i, \quad (4c)$$

where $A_{t,(i)}$ denotes the i th row of A_t . In the last-two constraints in (4), we encode that the $\gamma\%$ -confidence ellipsoid of the estimated state at time t lies within the associated safe polytope, where: $\bar{P}_{t|t-1} = \chi_3^2(\gamma) (F_t P_{t|t-1} F_t^T + Q_{t-1})$ represents the predicted covariance at time t , analogous to the Extended Kalman Filter (EKF) Predict Procedure, with F_t representing the Jacobian of the dynamics model at time t , and $\bar{P}_{t-1|t-1} = \chi_3^2(\gamma) P_{t-1|t-1}$. We utilize $\bar{P}_{t|t-1}$ in place of $\bar{P}_{t|t}$ since $\bar{P}_{t|t}$ is not known a-priori. The MAP problem in (4) yields the solution: $\mathbf{x}_t^* = [\mu_{t|t}^T, \mu_{t-1|t}^T]^T$, where the estimated pose of the robot at time t is given by the posterior distribution $\mathcal{N}(\mu_{t|t}, P_{t|t})$, with $P_{t|t}$ computed from the inverse Hessian of (4a).

Remark 6. We note that the minimizer of (4) represents an estimate of the mode of the posterior distribution. Here, we approximate the posterior distribution over (x_t, x_{t-1}) by a Normal distribution, whose mean corresponds to the solution of (4) and whose covariance is given by the inverse of the Hessian of (4a) evaluated at the optimal solution. Notably, this approximation is exact when the dynamics and measurement models are linear-Gaussian in the unconstrained

case. Moreover, given the posterior distribution, we can transform the estimated pose of the robot from $\mathfrak{se}(3)$ to $\text{SE}(3)$ via the exponential map.

Remark 7 (Smoothing of Prior Pose Estimates). *From (4), we obtain a smoothed estimate of the robot's pose at time $t - 1$, given by $\mathcal{N}(\mu_{t-1|t}, P_{t-1|t})$. By incorporating measurements up to time t , the smoothed estimate improves upon the estimate of the robot's pose at the preceding timestep.*

Given a set of correspondences between keypoints in successive images, we can interpret the second term in the objective function in (4) as the reprojection error, resulting in a nonlinear least-squares problem, which could be challenging to solve in real-time. In the subsequent discussion, we present a faster pose estimator to reduce the computation time for solving (4), at the expense of exactness.

C. Handling Highly Nonlinear Dynamics

For robots with arbitrary nonlinear dynamics models, solving the MAP optimization problem in (4) often proves challenging. To circumvent this challenge, we present a two-phase pose estimation procedure. In the first phase, we utilize an Unscented Kalman Filter (UKF) [43] to compute an estimate of the robot's pose consisting of a *predict* procedure, dependent on the robot's dynamics model, followed by an *update* procedure, which performs a correction on the estimated pose given the robot's observations (i.e., camera images). In the second phase, we solve a constrained optimization problem to satisfy an approximation of the problem in (4). We summarize the procedures in Algorithm 1.

Algorithm 1: Splat-Loc

```

Input: Prior  $\mathcal{D}_{t-1} = (\mu_{t-1|t-1}, P_{t-1|t-1})$ , Image  $I_t$ ;
Output: Posterior  $(\mu_{t|t}, P_{t|t})$ ;
// Prediction Phase
 $(\mu_{t|t-1}, P_{t|t-1}) \leftarrow \text{UKF\_Prediction}(\mathcal{D}_{t-1})$  ;
// Compute  $y_t$  from  $I_t$ .
 $y_t \leftarrow \text{Procedure (7)}$ ;
// Update Phase
 $(\hat{\mu}_{t|t}, \hat{P}_{t|t-1}) \leftarrow \text{Procedure (5)}$ ;
// Constrained Estimation
 $(\mu_{t|t}, P_{t|t}) \leftarrow \text{Procedure (6)}$ ;

```

Given a prior estimate of the robot's pose $(\mu_{t-1|t-1}, P_{t-1|t-1})$, we execute the Prediction Procedure of the UKF, which involves computing the sigmapoints associated with the prior, propagating these sigmapoints forward using the robot's dynamic models, and subsequently, computing the predicted distribution of the robot's pose $(\mu_{t|t-1}, P_{t|t-1})$ from the propagated sigmapoints. When the robot obtains new measurements, we compute the expected measurement $\mu_{y,t}$, along with the covariance matrices $P_{yy,t}$ and $P_{xy,t}$ from the sigmapoints associated with the prediction distribution. We fuse these new measurements via the UKF Update Procedure to compute the posterior distribution of the robot's pose from:

$$\hat{\mu}_{t|t} = \mu_{t|t-1} + K_t \Phi(y_t, \mu_{y,t}), \quad (5)$$

where K_t denotes the Kalman Gain, with mean $\hat{\mu}_{t|t}$ and covariance $\hat{P}_{t|t}$, approximating the posterior distribution, where Φ computes the *innovation*, i.e., the difference between the observed measurement and the expected measurement.

We note that the resulting estimated mean of the robot's pose may not reside within a safe polytope. As a result, we project the distribution into the associated safe polytope by solving the constrained convex optimization problem:

$$\begin{aligned} & \underset{\mu_{t|t} \in \mathbb{R}^n, P_{t|t} \in \mathbb{S}_{++}}{\text{minimize}} && -\log(|P_{t|t}|) + \|\mu_{t|t} - \hat{\mu}_{t|t}\|_{P_{t|t}^{-1}}^2 \\ & && + \text{trace}(\hat{P}_{t|t}^{-1} P_{t|t}) \end{aligned} \quad (6a)$$

$$\text{subject to } \chi_3^2(\gamma) \left\| A_{t,(i)}^T \right\|_{P_{t|t}}^2 + A_{t,(i)} \mu_{t|t} \leq b_{t,(i)}, \forall i. \quad (6b)$$

With the objective function in (6a), we seek to minimize the Kullback-Leibler (KL) divergence between the projected distribution $\mathcal{N}(\mu_{t|t}, P_{t|t})$ and the distribution of the estimated pose computed from the UKF $\mathcal{N}(\hat{\mu}_{t|t}, \hat{P}_{t|t})$. Furthermore, we introduce the constraint in (6b) to enforce that the $\gamma\%$ -confidence ellipsoid associated with the normal distribution $\mathcal{N}(\mu_{t|t}, P_{t|t})$ lies within the safe polytope.

D. Modifications to Observation Model

Note that the pose estimation procedure in (4) requires the observations y_t collected by the robot, which could be potentially high-dimensional. To further improve the computation time of the pose estimator, we design a procedure operating on the potentially high-dimensional observations (e.g., high-resolution images), transforming these observations into a more compact pseudo-measurement in $\text{SE}(3)$. In addition, we assume that the measurement function h extracts the $\mathfrak{se}(3)$ elements in its first input $\mu_{t|t-1}^{(i)}$, transforming them to $\text{SE}(3)$. Now, we present our procedure for computing the pseudo-measurement from the RGB image I_t .

Given an estimate of the robot's pose $\mu_{t|t-1}$, we first render a set of 2D images of the GSplat map within a local neighborhood of $\mu_{t|t-1}$. We then identify keypoints and the associated descriptors for points in these map images, as well as the robot's RGB image using the encoder-decoder-based method SuperPoint [44]. Next, we match features between both images using the transformer-based method LightGlue [45].¹ We again utilize RANSAC to remove outliers from the set of matched features to yield the final correspondence set \mathcal{C} . Finally, we take the map points from \mathcal{C} and project them into the 3D scene using the perspective projection model, the camera intrinsics, and the rendered depth from GSplat.

We can then estimate the pose of the robot using the Perspective-n-Point (PnP) problem, given by:

$$y_t = \arg \min_{\mathcal{P} \in \text{SE}(3)} \sum_{(c_w, c_o) \in \mathcal{C}} \delta(\Pi(\mathcal{P}, c_w), c_o), \quad (7)$$

where $y_t \in \text{SE}(3)$ denotes the pseudo-measurement describing the camera extrinsic parameters, \mathcal{C} denotes the set of

¹Other methods for feature detection and matching can be utilized, such as Scale-Invariant Feature Transform (SIFT) and Oriented FAST and Rotated BRIEF (ORB) [46].

correspondences between points in the environment c_w and points in the observed RGB image c_o , $\Pi : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ represents the perspective projection model of point p using the camera extrinsics \mathcal{P} , and $\delta : \mathbb{R}^2 \times \mathbb{R}^2 \rightarrow \mathbb{R}$ denotes the reprojection error. One common choice for δ is the ℓ_2 -squared-norm, which we utilize in this work. The ℓ_2 -squared-norm yields a nonlinear least-squares optimization problem that can be solved using variants of the Levenberg-Marquardt and Gauss-Newton optimization algorithms. We note that various methods have been developed to solve the PnP problem such as the EPnP, DLS, and SQPnP methods [47].

Given y_t and $\mu_{y,t}$, via $\Phi : \text{SE}(3) \times \mathfrak{se}(3) \mapsto \mathfrak{se}(3)$, we first map $\mu_{y,t}$ to $\text{SE}(3)$ and compute the measurement error in $\text{SE}(3)$. We compute the rotation component of the error, given by $R_\ell = R_{y(t)}^T R_{\mu_{y,t}}$ where $R_{y(t)}$ denotes the rotation described by y_t and $R_{\mu_{y,t}}$ denotes the rotation described by $\mu_{y,t}$. Likewise, we compute the translation error from the difference between the translation vectors described by y_t and $\mu_{y,t}$. Lastly, we map the innovation in $\text{SE}(3)$ to $\mathfrak{se}(3)$ using the logarithmic map [42]. We note that the rotation component of the innovation in $\mathfrak{se}(3)$ corresponds to the angle-axis representation of the relative rotation between $R_{y(t)}$ and $R_{\mu_{y,t}}$.

Remark 8 (Lightweight Pose Estimator). *We note that the pseudo-measurement y_t , computed directly from the RGB image I_t captured by the robot’s onboard camera, is often of sufficiently-high accuracy. Hence, y_t can be used directly as an estimate of the robot’s pose in many navigation problems, especially in non-highly dynamic problems with feature-rich scenes. However, we note such an approach could be brittle in problems where the robot moves at high speeds in featureless regions. In the experiments, we assess the accuracy of this approach, comparing it to prior work in point-cloud registration and pose estimation in Gaussian Splatting environments. Even without the UKF, we can still obtain an estimate of the covariance associated with y_t from the Hessian of (7) at its minimizer.*

E. Global Initialization

The above pose estimation requires an estimate of the robot’s pose (such as its neighborhood), which may not be available in many practical settings. When a good initial guess of the robot’s pose is unavailable, we execute a global pose estimation procedure. Our approach assumes that the robot has an onboard RGB-D camera with known camera intrinsics. We first use the RGB-D image and camera intrinsics to generate a point cloud (in the camera frame). In addition, we generate a point cloud of the scene from the GSplat using the ellipsoid means $\{\mu_j\}_{j=1}^N$, enabling the formulation of a point-cloud registration problem:

$$\mathcal{P}^* = \arg \min_{\mathcal{P} \in \text{SE}(3)} \sum_{(p,q) \in \mathcal{C}} \|p - Tq\|_2^2, \quad (8)$$

where the transformation matrix $T \in \text{SE}(3)$ comprises of a translation component $\rho \in \mathbb{R}^3$ and a rotation matrix $R \in \text{SO}(3)$ and \mathcal{C} denotes the set of correspondences, associating the point p in the map cloud to a point q in the point cloud from the

camera. Given a known set of correspondences, we can compute the optimal solution of (8) using Umeyama’s method [48].

In practice, we do not have prior knowledge of the set of correspondences \mathcal{C} between the two point clouds. To address this challenge, we apply standard techniques in feature-based global point-cloud registration. We begin by computing 33-dimensional Fast Point Feature Histograms (FPFH) descriptors [49] for each point in the point-cloud, encoding the local geometric properties of each point. Prior work has shown that visual attributes can play an important role in improving the convergence speed of point-cloud registration algorithms [50], something that FPFH does not do. To solve this, we augment the FPFH feature descriptor of a given point with its RGB color. We then identify putative sets of correspondences using a nearest-neighbor query based on the augmented FPFH descriptors, before utilizing RANSAC to iteratively identify and remove outliers in \mathcal{C} . The RANSAC convergence criterion is based on the distance between the aligned point clouds and the length of a pair of edges defined by the set of correspondences.

VI. EXPERIMENTS

We demonstrate the effectiveness of our navigation pipeline for GSplat Maps, examining its performance in real-world scenes on hardware and in simulation. We limit our evaluations to precomputed Gaussian Splatting environments, noting that most existing SLAM methods for Gaussian Splatting do not run in real-time. For example, the Gaussian Splatting SLAM methods in [38] and [37] run at a frequency of 3.2 FPS and 1.1 FPS, respectively, which is not fast enough for many real-time applications. We are further limited by the unavailability of open-source code implementations of some of these methods.

A. Experimental Setup

We evaluate Splat-Nav, including the planning and pose estimation components, across a range of simulated and hardware trials. We perform ablative studies, comparing our algorithms to existing methods.

1) Test Environments: We test Splat-Nav in four different environments: **Stonehenge**, a fully-synthetic scene; two real-world scenes **Statues** and **Flightroom**; and a **Mini-office** scene.

To generate the GSplat maps, we need collections of RGB images. For **Stonehenge**, we render RGB images of the mesh at specified camera poses. For **Statues** and **Flightroom**, we record a video of the scene using an iPhone camera. For the **Mini-office** scene we record a video using the camera onboard the drone. For all scenes, we utilize Nerfstudio [7] to train the Semantic Gaussian Splat, using its default parameters (which includes estimating the camera poses for each image frame from structure-from-motion via COLMAP [39]). We note that Nerfstudio adopts the NeRF conventions in scaling the scene to fit within the confines of a two-unit-length cube centered at the origin, with the poses of the camera residing within a $[-1, 1]^3$ -bounding box. In Figure 4, we show the true image captured by the drone in the Mini-office scene and the rendered image from the Gaussian Splat at the same camera pose. We note that the rendered image is photorealistic, highlighting the remarkable visual quality of the trained Gaussian Splat.

We distill embeddings from the 2D vision-language foundation model CLIP [51] into the 3D Gaussian Splatting scene representation to enable the specification of a goal location for the robot via natural-language. Consequently, a user can ask a robot to navigate to the “door” of a room represented using Gaussian Splatting, where the goal location of the robot is specified by the natural-language query “door.” Our approach follows similar distillation techniques applied in NeRF-based scene representations [52, 53]. However, unlike these methods, we distill the semantic embeddings into a Gaussian Splatting scene.

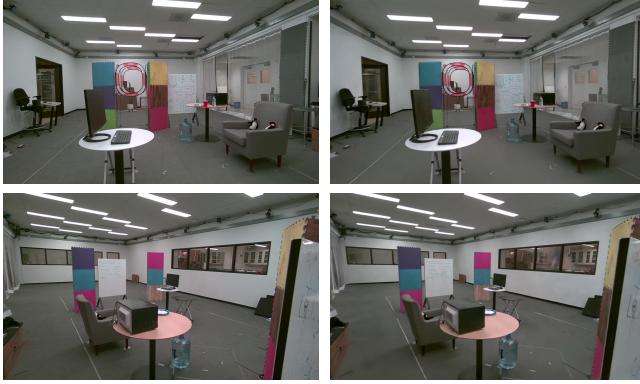


Fig. 4. Mini-Office Scene: (left) a real image from the scene, captured by the onboard camera, and (right) a photorealistic rendered image from the GSplat representation.

2) Hardware: We test our pipeline on a custom quadcopter with frame dimensions 33 cm x 30 cm x 12 cm equipped with an onboard Intel Realsense D455 camera. In the hardware tests, we approximate the robot using a sphere with radius $R = 0.2$ m. In the simulated tests, we represent the robot using a sphere of radius $R = 0.03$ because the scene is not in metric scale. We run the pose estimator and the planner ROS2 nodes on a desktop computer with an Nvidia RTX 4090 GPU and an Intel i9 13900K CPU, which communicates with the drone via WiFi. We emphasize that both modules are running asynchronously and in a receding horizon fashion. The drone transmits images from its cameras to the desktop station, which is utilized by the pose estimator in computing an estimate of the pose of the drone. We run the estimator continuously, synchronized with the stream of images published from the drone via ROS2.

The planning module utilizes the pose estimates computed by the pose estimator in computing a safe trajectory for the drone to follow. The planning node publishes a set of waypoints towards the goal every 10 seconds in order to allow for smooth progress towards the goal and discourage switching behavior (e.g., the A* may instantaneously switch from going left to going right). To preserve safety, whenever the planner is planning, the drone stays at hover until a new set of waypoints is published.

The resulting trajectory is sent to the drone for tracking. We utilize the OptiTrack Mocap system for low-level trajectory tracking. Training and rendering from the Gaussian Splats occur on the GPU, while the pseudo-measurement computation and the bulk of the planning module runs on the CPU. In essence,

the most time-consuming operations occur on the CPU, once the Gaussian Splat has been trained.

3) Goal Specification: In the hardware experiments, we specify the goal locations for the drone via natural language, comprising of the following objects within the Mini-office scene: a microwave, 5-gallon water jar, keyboard, and an arm-chair (depicted in Figure 4). We query the semantic Gaussian Splat for the location of these objects using the following text prompts: “black microwave,” “keyboard,” “gallon,” and “gray sofa,” corresponding to these objects. We show the semantic maps generated from the semantic Gaussian Splat in Figure 5 for each goal location. For each goal location, we run four trials with different starting configurations of the drone, selecting the starting configurations such that the goal location is not visible to the drone from its starting configuration, to enforce that the drone navigates around obstacles to reach the goal location. In addition, we select starting configurations such that the drone is compelled to navigate through the first-person view (FPV) drone gates, shown in Figure 6, to examine the capability of our proposed planning and pose estimation modules in navigation through narrow, confined spaces. We specify the starting locations of the drone outside the boundaries defined by the two sets of pillars, whereas the goal locations lies relatively close to the center of the scene. This setup results in 16 different flight experiments. Please refer to the videos on our webpage for more details.



Fig. 5. Goal specification in the Mini-Office Scene: We show the semantic maps for the natural-language queries used to specify the goal locations, including a (top-left) “black microwave,” (top-right) “keyboard,” (bottom-left) “gallon,” and (bottom-right) “gray sofa.”

B. Splat-Loc Experiments

We validate the performance of Splat-Loc in simulated experiments in the Statues scene and on hardware in the Mini-Office. See also Appendices G and H for additional results in other scenes (including the Stonehenge and Flightroom scenes) and with different values for the initial pose error. In these experiments, we demonstrate the effectiveness of the pseudo-measurements in RGB-only pose localization, as noted in Remark 8.

1) Hardware: We present the performance statistics of Splat-Loc on the drone navigation problem in the Mini-office scene in Table I. For each goal location, Splat-Loc achieves average rotation errors less than 5° and translation errors less than



Fig. 6. Visualizations of the hardware experiment in the **Mini-office** scene with the *Gallon* goal location. The drone flies through the FPV gates to reach the goal location, which is not visible in the figure. The drone is highlighted by the blue bounding box. The drone trajectories are best visualized in video, which is accessible on our webpage.

5 cm. Moreover, Splat-Loc runs at about 25 Hz on average, which is fast-enough for real-time operation. The bulk of the computation time is utilized in computing the feature matches and in solving the PnP problem, which requires about 10 milliseconds. We note that Splat-Loc was successful in estimating the pose of the drone across all trials. In Figure 7, we show the rotation and translation errors, as well as the computation time, on a random trial across all goal locations. As shown in the figure, Splat-Loc is able to recover from spikes in the pose errors, which may occur due to disturbances or occlusions.

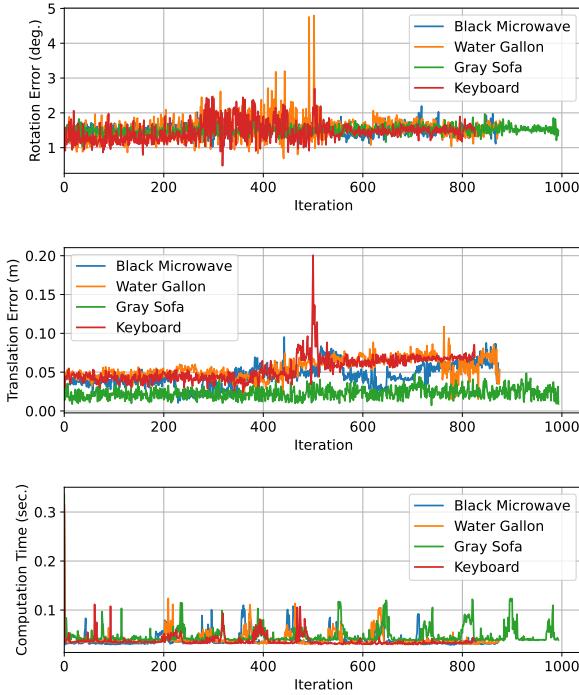


Fig. 7. Pose estimation errors and computation time required by Splat-Loc on a random trial of the hardware experiments in the Mini-office scene. Splat-Loc computes high-accuracy pose estimates in real-time.

2) *Comparison to Baselines*: We compare Splat-Loc to other pose estimation methods, including a baseline GS-Loc, based on the localization component of existing Gaussian Splatting SLAM methods [36, 37]. While these methods optimize over

the re-rendering loss composed of the photometric loss, and in some cases, depth and semantic-related loss terms, in our baseline, we optimize only over the photometric loss, since we assume the robot in these evaluations does not have an RGB-D camera for depth measurements. As a result, our baseline essentially matches the Gaussian Splatting SLAM method in [38]. In addition, we compare our pose estimator to the Point-to-plane Iterative Closest Point (ICP) [54] and Colored-ICP [50] algorithms.

We examine two variants of our pose estimator: Splat-Loc-Glue, which utilizes LightGlue for feature matching; and Splat-Loc-SIFT, which utilizes SIFT for feature matching. In each scene, we run 10 trials (of 100 frames each) of each pose estimation algorithm. We evaluate the rotation and translation errors with respect to the ground-truth pose as well as the computation time per frame and the success rate (a success means it generated a solution, regardless of quality).

The performance of pose estimation algorithms often depends on the error associated with initial estimate of the pose. As such, we test our system across a range of different errors in the initial estimate of the pose. In this study, we do not utilize the initialization procedure described in Section V, rather we assume an initial estimate of the pose is available. We generate the initial estimate by taking the ground truth pose then applying a rotation δ_R about a random axis and the translation δ_t in a random direction.

We note that many currently-available implementations of differentiable rasterizers for Gaussian Splatting do not provide the gradient of the re-rendering loss with respect to the camera pose [8, 55], required for gradient-based pose estimation. In addition, Gaussian Splatting does not support automatic differentiation, in its original form. As a result, we leverage finite differences to estimate the gradient of the photometric loss function utilized in the pose estimator. We note that our implementation might not be particularly fast or robust, especially for larger errors in the initial pose estimate, given the numerical approach utilized in estimating the gradients.

We provide the summary statistics of the error in the pose estimates computed by each algorithm, in addition to the computation time on a trial with 100 frames in the **Statues** scene in Table II. We note that all methods had a perfect success rate in this problem. The GS-Loc algorithm achieves the lowest accuracy and requires the greatest computation time, unlike Colored-ICP, Splat-Loc-SIFT, and Splat-Loc-Glue,

TABLE I
ROTATION AND TRANSLATION ERROR AND COMPUTATION TIME OF SPLAT-LOC IN HARDWARE EXPERIMENTS ON A DRONE.

Goal Location	Rotation Error (deg.)	Translation Error (m)	Computation Time (secs.)	Success Rate (%)
Black Microwave	$1.55e^0 \pm 7.23e^{-2}$	$3.97e^{-2} \pm 6.89e^{-3}$	$3.99e^{-2} \pm 1.03e^{-3}$	100
Water Gallon	$1.56e^0 \pm 5.21e^{-2}$	$4.52e^{-2} \pm 6.89e^{-3}$	$4.01e^{-2} \pm 1.71e^{-3}$	100
Gray Sofa	$1.54e^0 \pm 4.57e^{-2}$	$3.97e^{-2} \pm 1.02e^{-2}$	$4.04e^{-2} \pm 4.11e^{-3}$	100
Keyboard	$1.55e^0 \pm 1.08e^{-1}$	$4.59e^{-2} \pm 9.15e^{-3}$	$3.97e^{-2} \pm 2.90e^{-3}$	100

which achieve much-higher accuracy with a rotation error less than a degree and a translation error less than 15cm. GS-Loc requires a computation time of about 36.15 s per frame, which is about two orders of magnitude slower than the next-slowest method ICP, which requires a computation time of about 110 ms. Colored-ICP, Splat-Loc-SIFT, and Splat-Loc-Glue require less than 100ms of computation time. Compared to all methods, Splat-Loc-Glue yields pose estimates with the lowest mean rotation and translation error, less than 0.06° and 4 mm, respectively, and achieves the fastest mean computation time, less than 42 ms.

C. Splat-Plan Experiments

We evaluate the performance of Splat-Plan on hardware in the **Mini-office** scene and in three environments **Stonehenge**, **Statues**, and **Flightroom** in simulation. For each scene in simulation, we run 100 start and goal locations distributed in a circle around the boundary of the scene. In the hardware experiments, we utilize Splat-Plan MPC, executing the planning over a receding horizon during the navigation task, which enables the drone to handle disturbances and tracking errors arising from the low-level controller.

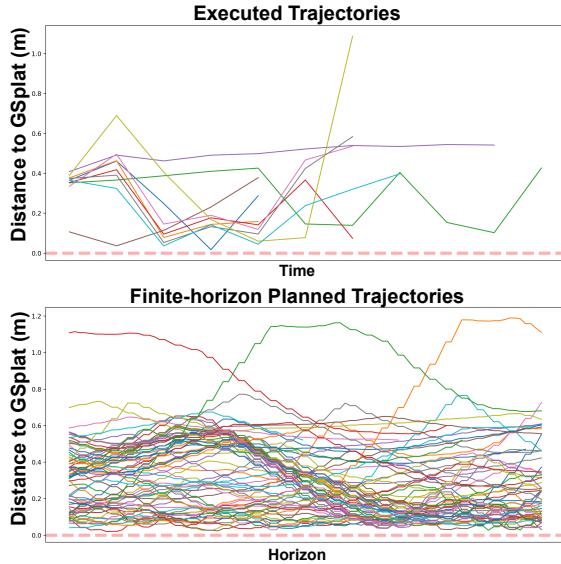


Fig. 8. Top: Executed trajectories of our drone and their distances to the surrounding GSplat environment. Bottom: The N -horizon waypoints generated by Splat-Plan MPC. Note that all lines are above the dotted red line at 0, hence all planned and executed paths are safe.

1) *Hardware*: We show the hardware results in Fig. 9, where we visualize the actual executed flight paths of our drone and

the intermediate reference Bézier curves planned through the safe polytope corridor at each time step. We ran a total of 16 flight experiments, described in Section VI-A3. Not only are all the corridors and paths (both reference and executed) safe with respect to the GSplat (Fig. 8), but also all trajectories reached the goal location successfully (i.e. 100% success rate). On average, the planner took 3.24 ± 1.78 seconds to perform replanning. On GPU, the generation of the polytopes took on average 70 ms to compute. Additionally, the extraction of the GSplat parameters (650K Gaussians) and the creation of the binary occupancy grid for A* requires 13 ms. As a result, the bulk of the compute time is spent on CPU through many iterations of A* in the complicated **Mini-office** scene and the costly solve time of the quadratic program to compute the Bézier curves. Performance optimizations can be made to the implementation by using a faster path-finding algorithm for initialization, solving for the minimum set of connected polytopes that connects the initial and goal positions, and using faster optimization solvers.

2) *Comparison to Baselines*: We compare Splat-Plan against baselines and prior work. Namely, we benchmark against: (1) an A* planner using the GSplat means as an occupancy grid, (2) an RRT* planner using the collision test Corollary 2, and (3) a gradient-based planner built for neural-network NeRFs, Nerf-Nav [10] (where we use Nerfstudio to generate a NeRF using the same input RGB images).

We also present several ablations and extensions of Splat-Plan. First, we benchmark Splat-Plan (Splat-Plan Prob.) on the full ellipsoidal GSplat representation (i.e the $\gamma\%$ -confidence ellipsoids) against naively using the ellipsoids represented by the GSplat covariances (Splat-Plan Basic). For these experiments, we take $\chi_3^2(\gamma) = 4$ for more interesting paths, especially in the Stonehenge scene.

Second, we compare the performance of Splat-Plan MPC (Iterative A* following) and Splat-Plan Global (where we solve for the full trajectory to the goal), an ablation of Splat-Plan Global that does not iterate on A* (Non-iterative A*), and a Splat-Plan MPC method that naively uses a straight line path to the goal to create polytopes instead of A* (Appendix F.A).

Visually, the paths generated by Splat-Plan are smooth, safe, and non-conservative (Fig. 10). This fact is validated in Fig. 11, where Splat-Plan's trajectories (probabilistic and basic variants) are safe (distances greater than 0) with respect to the *ground-truth mesh* in a vast majority of trials. Moreover, these trajectories are non-conservative due to small path lengths and variances. We also see that the γ variant is slightly safer and has marginally smaller path lengths. Using the means of the Gaussians in the GSplat and a gradient-based planner has no performance guarantees and therefore suffers in safety.

TABLE II
COMPARISON OF BASELINE POSE ESTIMATION ALGORITHMS IN THE STATUES SCENE WITH $\delta_R = 20^\circ$ AND $\delta_t = 0.1$ m.

Algorithm	Rotation Error (deg.)	Translation Error (m)	Computation Time (secs.)	Success Rate (%)
ICP [54]	$7.31e^1 \pm 4.59e^1$	$1.29e^0 \pm 7.50e^{-1}$	$1.07e^{-1} \pm 1.92e^{-2}$	100
Colored-ICP [50]	$8.28e^{-1} \pm 3.72e^{-1}$	$1.31e^{-2} \pm 5.98e^{-3}$	$4.33e^{-2} \pm 9.70e^{-3}$	100
Splat-Loc-SIFT (ours)	$8.59e^{-2} \pm 6.05e^{-2}$	$5.59e^{-3} \pm 7.45e^{-3}$	$6.33e^{-2} \pm 2.39e^{-3}$	100
Splat-Loc-Glue (ours)	$5.04e^{-2} \pm 3.09e^{-2}$	$3.30e^{-3} \pm 2.70e^{-3}$	$4.12e^{-2} \pm 7.32e^{-2}$	100
GS-Loc [36, 37, 38]	$1.22e^2 \pm 3.38e^1$	$2.45e^0 \pm 9.12e^{-1}$	$3.62e^1 \pm 5.44e^0$	100

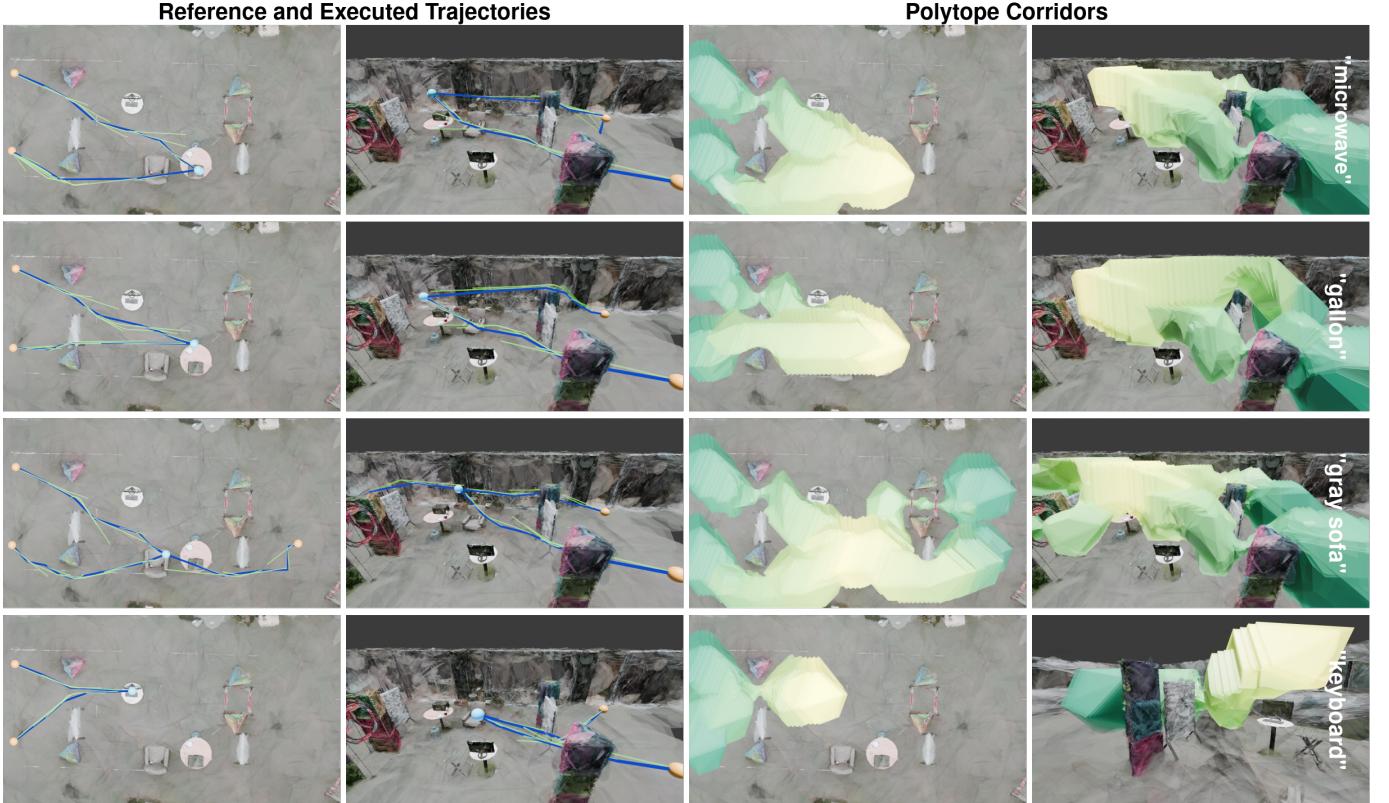


Fig. 9. Actual flight paths and corridors executed on hardware, visualized on top of the GSplat mesh. The generated safe polytope corridors are safe with respect to the GSplat, color-coded from start (green) to end (yellow). The blue path represents the actual flight path executed by the drone using our Splat-Plan MPC framework from start (orange) to goal (blue). The finite-horizon Bézier curve reference trajectory (planned through the corridors) at each time step are visualized in green.

The RRT* also suffers due to tradeoff between resolution of the collision checking and the lack of scalability in sampling random points.

Because the simulation scenes are simpler compared to the Mini-office scene, the compute times in these scenes are much faster, operating on the order of 5 Hz. Typically, A* requires about 150 ms, the polytope corridor generation requires about 10 ms, and a variable amount of time (between 50 ms to 1 second) is required to compute the Bézier curves depending on whether one is using Splat-Plan MPC or Global. See also Appendix F.B for results on computation times and additional discussion.

VII. CONCLUSION

We introduce an efficient navigation pipeline termed *Splat-Nav* for robots operating in Gaussian Splatting envi-

ronments. Splat-Nav consists of a guaranteed-safe planning module *Splat-Plan*, which allows for real-time planning (5 Hz) by leveraging the ellipsoidal representation inherent in GSplats for efficient collision-checking and safe corridor generation, facilitating real-time online replanning. Our proposed pose estimation module *Splat-Loc* computes high-accuracy pose estimates faster (25 Hz) and more reliably compared to existing pose estimation algorithms for radiance fields, such as NeRFs. We present extensive hardware and simulation results, highlighting the effectiveness of Splat-Nav.

VIII. LIMITATIONS AND FUTURE WORK

We only tested Splat-Nav in pre-constructed scenes. Existing Gaussian Splatting SLAM algorithms do not run in real-time [38, 37], limiting the application of our method in online mapping. In future work, we seek to examine the derivation

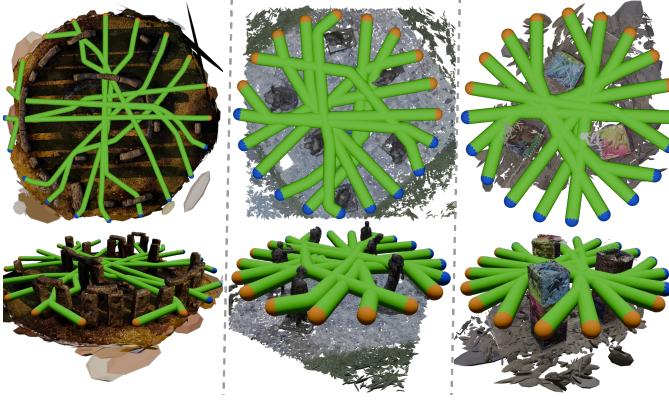


Fig. 10. Qualitative results of 10 safe trajectories using our proposed method with start (orange) and goal (blue) states spread over a circle. The widths of the trajectories are the same size as the robot body. We see that the trajectories are safe, but not conservative.

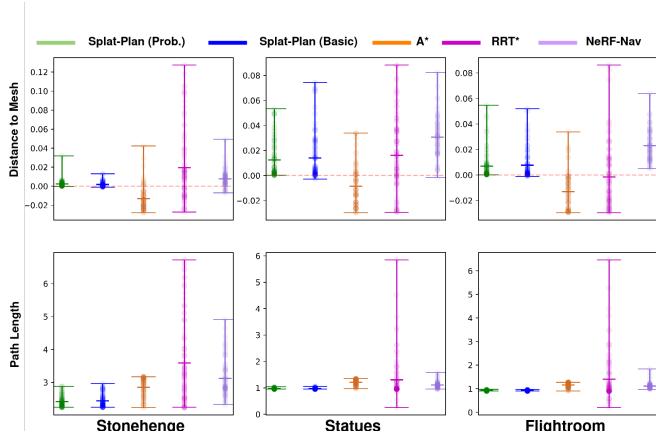


Fig. 11. A comparison of trajectories generated by our method and three other baselines (A^* , RRT^* , and NeRF-Nav [10]) for the same 100 start and end locations for each scene (Stonehenge, Statues, and Flightroom). We evaluate the safety and conservativeness of trajectories based on distance to the scene mesh and the length of the path, respectively. Each dot represents a particular trajectory's minimum distance to the mesh and its path length, and horizontal markings indicate the max, mean, and min over all trajectories. By using $\chi_3^2(\gamma) = 4$ in defining the confidence ellipsoids in the representation, our proposed method is able to be safe (min. distance greater than 0) and is not conservative (small path length and variance).

of real-time Gaussian Splatting mapping methods, integrated with the planning and pose estimation algorithms proposed in this work. Additionally, the results from Section VI-B2 suggest that we can incorporate Splat-Loc as a localization module within online Gaussian Splatting SLAM algorithms to improve localization accuracy and the resulting map quality.

We assumed that the pre-constructed scenes were correct. Safety of the planned trajectories depends on the quality of the underlying Gaussian Splatting map. As noted in Remark 3, we can use different confidence levels of the ellipsoids to account for uncertainty in an object in the GSplat map. Splat-Plan cannot do anything if an obstacle is completely missing from the scene, which is a fundamental limitation of the GSplat map representation. Future work will examine uncertainty

quantification of different regions within a GSplat scene to aid the design of active-planning algorithms that enable a robot to collect additional observations in low-quality regions, such as areas with missing/non-existent geometry, while updating the Gaussian Splatting scene representation via online mapping.

We only tested Splat-Nav in a static scene. This could be a limitation in many practical problems. Using NeRFs and GSplat for dynamic environments remains an open area of research, especially in scenes without prerecorded motion [56, 57, 58]. Splat-Plan can be extended easily to problems with dynamic scenes so long as the underlying dynamic Gaussian Splatting representation is available. Specifically, we can use our Splat-Plan Local algorithm to plan short horizon paths, leveraging new observations of the dynamic obstacles to predict future motion of each object during the planning procedure.

The performance of Splat-Loc depends on the presence of informative features in the scene. We can address this in two ways: through planning and by incorporating additional sensor data. Future work will explore the design of planning algorithms that bias the path towards feature-rich regions, improving localization accuracy during path execution. Future work will also incorporate IMU data for to improve the robustness of the pose estimator, particularly in featureless regions of the scene where the PnP -RANSAC procedure might fail.

In addition, we seek to more deeply explore the utilization of Splat-Nav across a diversity of hardware platforms and real-life environments. Future work will also aim to run all modules of Splat-Nav onboard the robot.

APPENDIX A PROOF OF PROPOSITION 1

This proposition was stated by [59] without proof. We provide the proof here. The goal is to diagonalize both Σ_a and Σ_b in such a way that they share the same eigenbasis. In this way, the matrix inversion amounts to a reciprocal of the eigenvalues. We can find such an eigenbasis through the generalized eigenvalue problem, which solves the following system of equations $\Sigma_a \phi_i = \lambda_i \Sigma_b \phi_i$. These generalized eigenvalues and eigenvectors satisfy the identity $\Sigma_a \phi = \Sigma_b \phi \Lambda$.

To solve the generalized eigenvalue problem, we utilize the fact that Σ_a and Σ_b are symmetric, positive-definite, so the Cholesky decomposition of $\Sigma_b = LL^T$, where L is lower triangular. This also means that we can represent Σ_a by construction as: $\Sigma_a = LCL^T = LV\Lambda V^T L^T$, where C is also symmetric, positive-definite, so its eigen-decomposition is $C = V\Lambda V^T$, where the eigenvectors V are orthonormal $V^T V = I$. To retrieve the original eigenbasis ϕ , we solve the triangular system $L^T \phi = V$. A consequence of this solution is that: $V^T V = \phi^T LL^T \phi = \phi^T \Sigma_b \phi = I$. Moreover, this means that $\Sigma_b = \phi^{-T} \phi^{-1}$. We can show that Λ and ϕ indeed solve the generalized eigenvalue problem through the following substitution:

$$\Sigma_a \phi = LV\Lambda V^T L^T \phi = \underbrace{LL^T}_{\Sigma_b} \phi \Lambda \phi^T \underbrace{LL^T}_{\Sigma_b} \phi = \Sigma_b \phi \Lambda \underbrace{\phi^T \Sigma_b \phi}_{I},$$

where $\Lambda = \text{diag}(\lambda_i)$ is the diagonal matrix of eigenvalues.

As a result, we can write the matrix inversion in Theorem 1 as the following:

$$\begin{aligned} \left[\frac{1}{1-s} \Sigma_a + \frac{1}{s} \Sigma_b \right]^{-1} &= \left[\frac{1}{1-s} \Sigma_b \phi \Lambda \phi^T \Sigma_b + \frac{1}{s} \Sigma_b \right]^{-1} \\ &= \left[\phi^{-T} \left(\frac{1}{1-s} \Lambda + \frac{1}{s} I \right) \phi^{-1} \right]^{-1} \\ &= \phi \operatorname{diag} \left(\frac{s(1-s)}{1+s(\lambda_i-1)} \right) \phi^T. \end{aligned}$$

Multiplying the left and right by $\mu_b - \mu_a$ completes the proof.

APPENDIX B PROOF OF COROLLARY 1

Using a finite number of samples, the test can fail (no sample points yield $K(s) > 1$) or succeed (there is a sample point such that $K(s) > 1$). Should the test succeed, by definition, there is no intersection. If it fails, there either exists an unsampled point with $K(s) > 1$ or such a point does not exist. If no such point exists, then the test remains correct in predicting collision. If such a point exists, it simply means our test predicted collision while in reality the ellipsoids are intersection-free. In other words, using finite samples leads to a safe, but conservative test. Due to the concavity of $K(s)$, more samples will asymptotically decrease the conservativeness. In fact, due to the monotonicity of the derivative of $K(s)$ and the bounds on s , even performing iterations of bisection search will yield exponential convergence.

APPENDIX C ELLIPSOID-TO-ELLIPSOID INTERSECTION TESTS

In the subsequent discussion, we examine existing algorithms for detecting if a pair of ellipsoids intersect. Recall that we have the original test in Proposition 1 that involves solving the generalized eigenvalue problem and the sampling-based approximation in Corollary 1, with guarantees on safety due to function convexity.

A. Test Based on Sherman-Morrison formula

We introduce an alternative approach for solving Theorem 1 that neither requires solving a generalized eigenvalue problem nor explicitly computing the inverse of a matrix. Given the representation of the ellipsoids: $\Sigma_a = R_a S_a S_a^T R_a^T$ and $\Sigma_b = R_b S_b S_b^T R_b^T$, let $S_a = \operatorname{diag}(\sqrt{\lambda_{a,(1)}}, \dots, \sqrt{\lambda_{a,(3)}})$, with S_b defined similarly.

Proposition 3. *The concave function $K(s)$ in the intersection test given in Theorem 1 can be simplified to:*

$$K(s) := (\mu_a - \mu_b)^T V_3(s) (\mu_a - \mu_b), \quad (9)$$

where $V_3 : \mathbb{R} \rightarrow \mathbb{R}^{3 \times 3}$ represents a matrix-valued function of s . For simplicity, we present $V_3(s)$ as a composition of the following functions of s : $\forall i \in \{1, 2, 3\}$,

$$V_i(s) := V_{i-1}(s) - \frac{V_{i-1}(s) d_{b,(i)} r_{b,(i)}^T V_{i-1}(s)}{1 + d_{b,(i)} r_{b,(i)}^T V_{i-1}(s) r_{b,(i)}},$$

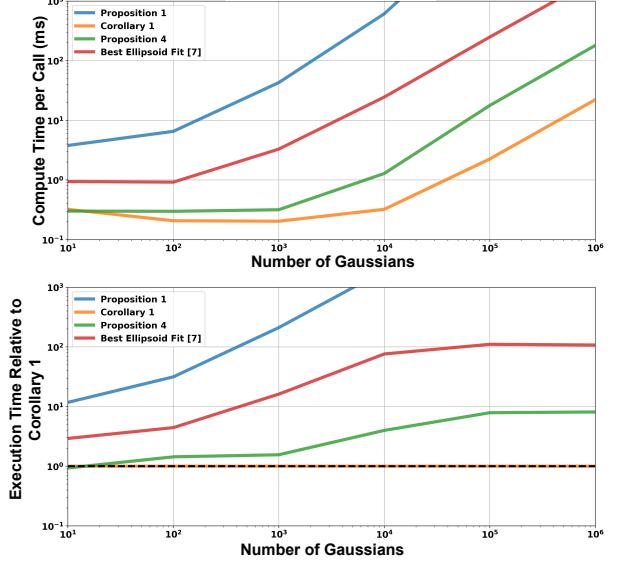


Fig. 12. Comparison of mean compute times and performance gap for various ellipsoid-ellipsoid collision detection algorithms, each executed on 100 different sets of Gaussians per fixed number of Gaussians.

with $V_0(s) := R_a \Lambda_a(s) R_a^T$, where $\Lambda_a : \mathbb{R} \rightarrow \mathbb{R}^{3 \times 3}$ yields a diagonal matrix, whose i th diagonal element is given by $\frac{(1-s)}{\lambda_{a,(i)}}$, $r_{b,(i)} \in \mathbb{R}^3$ denotes the i th column of R_b , and $d_{b,(i)} := \frac{\lambda_{b,(i)}}{s}$.

We omit the proof as it follows from the iterative application of the Sherman-Morrison formula [60].

B. Results

We compare the methods in [61] to the collision-detection algorithms presented in Proposition 1, Corollary 1, and Proposition 3. We did not compare against the method in [9], as well as optimization-based methods that require the computation of a separating hyperplane, since these methods do not scale efficiently to dense scenes with many Gaussians, which arise in Gaussian Splattting Maps. Figure 12 shows the results, where we see that Corollary 1 is the fastest across the full range of map densities.

APPENDIX D MINIMAL COLLISION TEST SET

The minimal collision test set G_x is a ball centered at a test point x with a radius that guarantees the ball will contain the center of the largest ellipsoid (with largest eigenvalue λ_{\max}) whose circumscribed circle intersects with the robot body centered at x . We note that ellipsoids outside of this ball cannot intersect with the robot body. Further, this ball must have radius at most $\lambda_{\max} + \kappa$, meaning the center of this ellipsoid lies on the surface of the ball. We stop iterating when the largest eigenvalue between successive queries to the K-D tree does not change, meaning the ellipsoid corresponding with the largest eigenvalue is the largest ellipsoid that could intersect our robot. The algorithm will only find the minimal set if the center associated with λ_{\max} lies on the surface of the ball. However, the gap between the converged set and the minimal set is relatively small.

APPENDIX E PROOF OF PROPOSITION 2

From Proposition 2, we know that we must find an x such that $K(s) > 1$ for all ellipsoids in the test set \mathcal{G}^* . We will approximate this safe set as a polytope. Given a test point x^* and the j th ellipsoid in the collision test set \mathcal{G}^* , we can use our collision test (Corollary 2) to derive these polytopes. Notice that μ_a represents the position of the robot (i.e., the test point x^*) and μ_b is the centroid of ellipsoid j . For any value of $s \in (0, 1)$, the safety test is a quadratic constraint of the test point x^* , namely $\Delta_j^T Q_j \Delta_j > 1$.

To derive the supporting hyperplane for the safety check, we will first find the point on the ellipsoid and then linearize our test about this point. Let $f_j(x) = (x - \mu_j)^T Q_j (x - \mu_j)$. Note that $f_j(x^*) = \Delta_j^T Q_j \Delta_j = k_j^2$. We can immediately see that point $x_0 = \mu_j + \frac{1+\epsilon}{k_j} \Delta_j$ will be outside of the ellipsoid for any $\epsilon > 0$ since:

$$f_j(x_0) = \frac{(1+\epsilon)^2}{k_j^2} \Delta_j^T Q_j \Delta_j = (1+\epsilon)^2 > 1. \quad (10)$$

Note that this point x_0 lies on the ray starting from the center of the ellipsoid μ_j and passing through the center of the robot at x^* . We then linearize the constraint $f_j(x)$ about x_0 . Taking the derivative yields $\frac{df_j}{dx}(x) = 2(x - \mu_j)^T Q_j$. The linear approximation of the constraint is then:

$$f_j(x) \approx f_j(x_0) + \left[\frac{df_j}{dx}(x_0) \right] (x - x_0) \geq 1. \quad (11)$$

Plugging in $x_0 = \mu_j + \frac{1+\epsilon}{k_j} \Delta_j$ and simplifying yields the given expression $\Delta_j^T Q_j x \geq (1+\epsilon)k_j + \Delta_j^T Q_j \mu_j$.

We need to also prove that the above constraint is a supporting hyperplane by showing that all feasible points when the constraint is equality is necessarily outside of the ellipsoid parametrized by Q_j . Recall that $x_0 = \mu_j + \frac{\Delta_j}{k_j}$ is a point both on the surface of the ellipsoid and on the hyperplane. Therefore, all feasible points on the hyperplane can be expressed as $x = x_0 + \delta$, where $\Delta_j^T Q_j \delta = 0$. If the hyperplane supports the ellipsoid parametrized by Q_j , then necessarily $f(x) = (x - \mu_j)^T Q_j (x - \mu_j) \geq 1$. For all points on the hyperplane, the ellipsoid constraint evaluates to

$$\begin{aligned} f_j(x_0 + \delta) &= \left(\frac{1+\epsilon}{k_j} \Delta_j + \delta \right)^T Q_j \left(\frac{1+\epsilon}{k_j} \Delta_j + \delta \right) \\ &= \underbrace{\frac{(1+\epsilon)^2}{k_j^2} \Delta_j^T Q_j \Delta_j}_{=(1+\epsilon)^2} + \underbrace{\delta^T Q_j \delta}_{\geq 0} + \underbrace{\frac{2(1+\epsilon)}{k_j} \Delta_j^T Q_j \delta}_{=0} > 1. \end{aligned}$$

Hence, the constraint in Proposition 2 is a supporting hyperplane.

APPENDIX F EXTENDED PLANNING RESULTS

We remind the reader of the following baselines: RRT*, A*, and polytopic A*. In RRT* (implemented in OMPL), we replace the default collision checker with our ellipsoid-to-ellipsoid collision test from Corollary 1. The A* methods use the means of the Gaussians in GSplat to create a binary

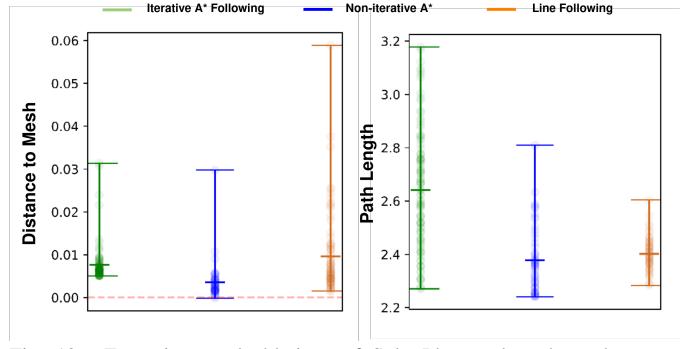


Fig. 13. Extensions and ablations of Splat-Plan evaluated on the same configurations in **Stonehenge**. We compare three methods: iterative A* following, non-iterative A* following, and line following. All the methods are still safe (distances greater than 0) because they use polytopic corridors. Iterative A* has a 100% success rate by nature of the interaction between the safe polytopes and a guaranteed safe A* path. Non-iterative A* could not find a solution 45% of the time due to the inability to find a connected union of polytopes generated by the A*. Finally, line following produced infeasible programs 18% of the time, and timed out getting to the goal (i.e. "stuck") 24% of the time.

occupancy grid. A* simply plans a shortest path through the binary grid. Polytopic A* goes one step further, constructing safe polytopes through Proposition 2 on the A* waypoints, from which it computes a B-spline trajectory passing through all the polytopes. Splat-Plan is a further refinement of the Polytopic A* algorithm. Instead of a single instance of A*, we iterate between querying A* and updating the binary occupancy grid at the A* waypoints that are unsafe via the collision test in Proposition 1. Typically, the Splat-Plan method requires only a handful of A* iterations (< 5), and the generation of polytopes for about 100 A* waypoints runs at about 100 Hz.

A. Ablations

We test the limits of the safe corridor generation by benchmarking the Splat-Plan MPC method and the Splat-Plan Global method. We see that all methods are safe (Fig. 13) due to the robust guarantees of Proposition 2. The Splat-Plan MPC (Iterative A* Following) always succeeds in finding feasible trajectories to the goal, validating our performance claim. The sub-optimality of the path lengths are also apparent. Without relying on the iterative A* procedure (Non-Iterative A*), the feasibility and connectivity of trajectories falls to 55% due to the "corner" phenomenon (Remark 1). Interestingly, removing A* entirely and sampling corridor query points on the line between the current and goal location (Line Following) is surprisingly performant, able to navigate to the goal 58% of the time.

B. Splat-Plan Times

In addition to outperforming other existing methods in safety and path length of computed paths (Fig. 13), our Splat-Plan method achieves faster computation times compared to the point-based motion planners RRT* and NeRF-Nav; generates smooth trajectories unlike the RRT*, NeRF-Nav, and A*; and detects infeasible planning problems.

Since Splat-Plan requires the use of A* to initialize a path to generate the safety polytope corridors, it will necessarily be

TABLE III
COMPARISON OF FULL TRAJECTORY COMPUTATION TIMES

Algorithm	Times (s)	Smooth	Failure Signal
NeRF-Nav [10]	2.33	✗	✗
RRT*	9.49 ± 5.13	✗	✗
A*	0.051 ± 0.0096	✗	✗
Non-iterative, Polytopic A*	0.13 ± 0.037	✓	✓(55%)
Splat-Plan Global (Simulated, ours)	0.90 ± 0.29	✓	✓(100%)

slower than these A* baselines. As Table III shows, Splat-Plan is much faster than RRT* and NeRF-Nav [10] due to our method’s inherent sampling efficiency. The faster computation time in Splat-Plan results primarily from the fast safe polytope generation procedure. The dominating cost in Splat-Plan Global is the calculation of the Bézier spline due to the number of polytopes in the constraints, with many facets per polytope. We solved the QP using CVXPY with the Clarabel solver. We note that the computation time associated with the Bézier spline computation procedure can be reduced using embedded solvers or faster programming languages.

C. Discussion of Safety

The creation of polytopes also endows our method with the ability to fail gracefully. A smooth curve can be created from the start to goal locations so long as a connected series of safe polytopes are given. Since the polytopes generated are inherently safe, we can simply check for overlap between consecutive pairs of polytopes. In this work, for simplicity, we choose to solve an LP to determine overlap. Typically, this operation can be performed at more than 10 Hz for the entire trajectory. Of course, less computationally-intensive methods exist, such as the GJK algorithm to determine collision between convex objects. Using the connectivity of polytopes as a failure signal, we observe that by not updating the occupancy grid and subsequently executing A*, polytopic A* can only achieve feasible safe trajectories 55% out of 100 configurations. This is due to the corner phenomenon in Remark 8, where safe polytopes existing on both sides of an ellipsoidal wall are disconnected.

Additionally, because RRT* and A* enforce safety point-wise, they have finite resolution. Therefore, these baselines may not be dynamically feasible for some robotic tasks, and may be unsafe in certain regions of the computed trajectory, if the resolution is insufficient. Given this, we found the 55% success rate of the polytopic A* to be surprisingly high, which speaks to the ability of our polytopes to be generated even when query points are unsafe.

D. Splat-Plan MPC

Since the polytope and A* computation procedures are relatively fast, while computation of the B-spline is relatively slow, we can compute the B-spline for some small time step in the future, in a receding-horizon manner. In our Splat-Plan MPC, we do this by fitting a B-spline through the current and next polytope. This still guarantees safety and feasibility since the entire constraint set is separable based on time, at the expense of the optimality of the solution.

TABLE IV
COMPARISON OF LOCAL TRAJECTORY PLANNING ALGORITHM COMPUTATION TIMES.

Algorithm	Times (s)	Smooth	Failure Signal
Line Following	0.015 ± 0.0086	✓	✓(58%)
Splat-Plan MPC (Simulated, ours)	0.16 ± 0.076	✓	✓(100%)
Splat-Plan MPC (Hardware, ours)	3.24 ± 1.78	✓	✓(100%)

To push the limits of generalizability of the polytopes, we implemented a simple line following planner based on the polytopes. Specifically, in this baseline, we query a polytope at the robot’s current position, and another polytope at a point some small distance away (0.02 in our evaluations) on the line between the current position and the goal position, then solve for a B-spline. Such a method did not yield connected polytopes for 18% of configurations, and got stuck (ran for over 200 iterations) for 24% of configurations. Note that while the robot was stuck, the Bézier spline was still guaranteed safe (Table IV).

APPENDIX G RECURSIVE POSE ESTIMATION

We present additional results demonstrating the performance of Splat-Loc compared to other algorithms. Table V shows our performance metrics in the **Statues** scene with $\delta_R = 20^\circ$ and $\delta_t = 0.1$ m. First, we see that only the Colored-ICP algorithm fails to achieve a 100% success rate. Second, our algorithms Splat-Loc-Glue and Splat-Loc-SIFT yield higher-accuracy and more consistent solutions, with mean rotation and translation errors less than 0.1 deg and 7 mm, respectively, and lower standard deviations. In these tests, Splat-Loc-Glue performs the best on each metric with a computation time of about 35 ms.

In Table VI, we present results for the **Statues** scene with $\delta_R = 30^\circ$ and $\delta_t = 0.5$ m. Note that the success rates of Colored-ICP and Splat-Loc-SIFT decrease sharply, suggesting that these methods require a good initial estimate of the pose of the robot. However, when they do succeed, these algorithms achieve low rotation and translation errors, with Splat-Loc-SIFT achieving a much-higher accuracy compared to Colored-ICP. In contrast, the Splat-Loc-Glue algorithm achieves a perfect success rate, along with the lowest rotation and translation errors and computation time. The results suggest the greater robustness of Splat-Loc-Glue compared to Splat-Loc-SIFT.

In addition, we examine the performance of the pose estimation algorithms in problems with a larger error in the initial estimate of the pose, with $\delta_R = 30^\circ$ and $\delta_t = 0.5$ m in the synthetic **Stonehenge** scene. We present the performance of each algorithm on each metric in Table VII, where we note that ICP and Colored-ICP do not provide accurate estimate of the robot’s pose. Moreover, the pose estimation errors achieved by ICP and Colored-ICP have a significant variance. In contrast, Splat-Loc-SIFT and Splat-Loc-Glue yield pose estimates of high accuracy with average rotation and translation errors less than 0.5 deg. and 5mm, respectively, similar to the setting with low errors in the initial pose estimate. However, Splat-Loc-SIFT

TABLE V
COMPARISON OF POSE ESTIMATION ALGORITHMS IN THE STATUES SCENE WITH $\delta_R = 20^\circ$ AND $\delta_t = 0.1$ m

Algorithm	Rotation Error (deg.)	Translation Error (m)	Computation Time (secs.)	Success Rate (%)
ICP [54]	$8.42e^1 \pm 5.58e^1$	$3.83e^0 \pm 8.51e^0$	$8.60e^{-2} \pm 4.40e^{-2}$	100
Colored-ICP [50]	$3.02e^1 \pm 4.58e^1$	$3.46e^{-1} \pm 5.11e^{-1}$	$6.27e^{-2} \pm 2.72e^{-2}$	80
Splat-Loc-SIFT (ours)	$8.32e^{-2} \pm 9.37e^{-3}$	$5.72e^{-3} \pm 7.62e^{-4}$	$6.56e^{-2} \pm 7.16e^{-4}$	100
Splat-Loc-Glue (ours)	$5.73e^{-2} \pm 4.82e^{-3}$	$3.13e^{-3} \pm 3.24e^{-4}$	$3.42e^{-2} \pm 2.00e^{-3}$	100

TABLE VI
COMPARISON OF POSE ESTIMATION ALGORITHMS IN THE STATUES SCENE WITH $\delta_R = 30^\circ$ AND $\delta_t = 0.5$ m

Algorithm	Rotation Error (deg.)	Translation Error (m)	Computation Time (secs.)	Success Rate (%)
ICP [54]	$9.90e^1 \pm 3.06e^1$	$1.41e^0 \pm 2.30e^{-1}$	$4.36e^{-2} \pm 5.08e^{-2}$	100
Colored-ICP [50]	$2.42e^0 \pm 1.61e^0$	$2.19e^{-1} \pm 2.02e^{-1}$	$7.03e^{-2} \pm 2.44e^{-2}$	20
Splat-Loc-SIFT (ours)	$7.85e^{-2} \pm 5.96e^{-3}$	$6.28e^{-3} \pm 8.40e^{-4}$	$7.32e^{-2} \pm 1.50e^{-3}$	30
Splat-Loc-Glue (ours)	$6.14e^{-2} \pm 6.00e^{-3}$	$3.68e^{-3} \pm 6.54e^{-4}$	$4.28e^{-2} \pm 1.50e^{-3}$	100

achieves a lower success rate, compared to Splat-Loc-Glue, which achieves a perfect success rate.

APPENDIX H GLOBAL POSE INITIALIZATION OF SPLAT-LOC

We evaluate the global point-cloud alignment initialization procedure from Section V-E when utilized by Splat-Loc-Glue, in the real-world scene Statues and synthetic scene Stonehenge, with the results in Table VIII. We see that the success rate is 80% in both scenes, and it runs at approximately 20-30 Hz. When it does succeed, the pose errors are on the order of 0.3 deg and 1 cm. Although these estimates are relatively good, we note that these are less accurate than the recursive pose estimates discussed above.

REFERENCES

- [1] A. Elfes, “Using occupancy grids for mobile robot perception and navigation,” *Computer*, vol. 22, no. 6, pp. 46–57, Jun. 1989.
- [2] H. Edelsbrunner, “Surface Reconstruction by Wrapping Finite Sets in Space,” in *Discrete and Computational Geometry: The Goodman-Pollack Festschrift*, ser. Algorithms and Combinatorics, B. Aronov, S. Basu, J. Pach, and M. Sharir, Eds. Berlin, Heidelberg: Springer, 2003, pp. 379–404.
- [3] P. Kim, J. Chen, and Y. K. Cho, “Slam-driven robotic mapping and registration of 3d point clouds,” *Automation in Construction*, vol. 89, pp. 38–48, 2018.
- [4] S. Osher and R. P. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*. New York: Springer, 2003.
- [5] 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 *European Conference on Computer Vision (ECCV)*, 2020.
- [6] 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.
- [7] M. Tancik, E. Weber, R. Li, B. Yi, T. Wang, A. Kristoffersen, J. Austin, K. Salahi, A. Ahuja, D. McAllister, A. Kanazawa, and E. Ng, “Nerfstudio: A framework for neural radiance field development,” in *SIGGRAPH*, 2023.
- [8] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, “3D Gaussian splatting for real-time radiance field rendering,” *ACM Transactions on Graphics*, vol. 42, no. 4, July 2023. [Online]. Available: <https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/>
- [9] I. Gilitschenski and U. D. Hanebeck, “A robust computational test for overlap of two arbitrary-dimensional ellipsoids in fault-detection of kalman filters,” in *2012 15th International Conference on Information Fusion*, 2012, pp. 396–401.
- [10] 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 (RA-L)*, vol. 7, no. 2, pp. 4606–4613, 2022.
- [11] S. LaValle, “Planning algorithms,” *Cambridge University Press google schola*, vol. 2, pp. 3671–3678, 2006.
- [12] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.
- [13] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, “Anytime motion planning using the rrt,” in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 1478–1483.
- [14] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, “Chomp: Gradient optimization techniques for efficient motion planning,” in *2009 IEEE international conference on robotics and automation*. IEEE, 2009, pp. 489–494.
- [15] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, “Motion planning with sequential convex optimization and convex collision checking,” *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.
- [16] I. A. Sucan, M. Moll, and L. E. Kavraki, “The Open Motion Planning Library,” *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012,

TABLE VII
COMPARISON OF POSE ESTIMATION ALGORITHMS IN THE STONEHENGE SCENE WITH $\delta_R = 30^\circ$ AND $\delta_t = 0.5$ m.

Algorithm	Rotation Error (deg.)	Translation Error (m)	Computation Time (secs.)	Success Rate (%)
ICP [54]	$1.31e^2 \pm 2.26e^1$	$3.70e^0 \pm 5.54e^0$	$1.22e^{-1} \pm 1.53e^{-1}$	100
Colored-ICP [50]	$9.49e^1 \pm 5.13e^1$	$5.74e^{-1} \pm 2.88e^{-1}$	$4.88e^{-1} \pm 1.04e^{-1}$	20
Splat-Loc-SIFT (ours)	$2.17e^{-1} \pm 3.69e^{-2}$	$3.34e^{-3} \pm 5.63e^{-4}$	$1.39e^{-1} \pm 3.15e^{-3}$	70
Splat-Loc-Glue (ours)	$2.20e^{-1} \pm 2.03e^{-1}$	$3.15e^{-3} \pm 2.10e^{-3}$	$4.51e^{-2} \pm 6.11e^{-4}$	100

TABLE VIII
PERFORMANCE OF THE POSE INITIALIZATION MODULE IN SPLAT-LOC

Scene	Rotation Error (deg.)	Translation Error (m)	Computation Time (secs.)	Success Rate (%)
Statues	$3.69e^{-1} \pm 7.21e^{-1}$	$1.34e^{-2} \pm 2.33e^{-2}$	$3.39e^{-2} \pm 5.70e^{-4}$	80
Stonehenge	$2.14e^{-1} \pm 2.07e^{-1}$	$3.03e^{-3} \pm 2.28e^{-3}$	$4.58e^{-2} \pm 1.70e^{-3}$	80

<https://ompl.kavrakilab.org>.

- [17] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, “Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3D complex environments,” *IEEE Robotics and Automation Letters (RA-L)*, vol. 2, no. 3, pp. 1688–1695, 2017.
- [18] J. Van den Berg, M. Lin, and D. Manocha, “Reciprocal velocity obstacles for real-time multi-agent navigation,” in *2008 IEEE international conference on robotics and automation*. Ieee, 2008, pp. 1928–1935.
- [19] X. Wu, S. Chen, K. Sreenath, and M. W. Mueller, “Perception-aware receding horizon trajectory planning for multicopters with visual-inertial odometry,” *IEEE Access*, vol. 10, pp. 87 911–87 922, 2022.
- [20] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, “Voxblox: Incremental 3D Euclidean signed distance fields for on-board MAV planning,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.
- [21] L. Han, F. Gao, B. Zhou, and S. Shen, “Fiesta: Fast incremental euclidean distance fields for online motion planning of aerial robots,” *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4423–4430, 2019.
- [22] L. J. Guibas, R. Motwani, and P. Raghavan, “The robot localization problem,” *SIAM Journal on Computing*, vol. 26, no. 4, pp. 1120–1138, 1997.
- [23] A. Eman and H. Ramdane, “Mobile robot localization using extended kalman filter,” in *2020 3rd International Conference on Computer Applications & Information Security (ICCAIS)*. IEEE, 2020, pp. 1–5.
- [24] D. Fox, S. Thrun, W. Burgard, and F. Dellaert, “Particle filters for mobile robot localization,” in *Sequential Monte Carlo methods in practice*. Springer, 2001, pp. 401–428.
- [25] Q.-b. Zhang, P. Wang, and Z.-h. Chen, “An improved particle filter for mobile robot localization based on particle swarm optimization,” *Expert Systems with Applications*, vol. 135, pp. 181–193, 2019.
- [26] I. Ullah, Y. Shen, X. Su, C. Esposito, and C. Choi, “A localization based on unscented kalman filter and particle filter localization algorithms,” *IEEE Access*, vol. 8, pp. 2233–2246, 2019.
- [27] J. Biswas and M. Veloso, “Depth camera based indoor mobile robot localization and navigation,” in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 1697–1702.
- [28] P. Karkus, D. Hsu, and W. S. Lee, “Particle filter networks with application to visual localization,” in *Conference on robot learning*. PMLR, 2018, pp. 169–178.
- [29] R. Jonschkowski, D. Rastogi, and O. Brock, “Differentiable particle filters: End-to-end learning with algorithmic priors,” *arXiv preprint arXiv:1805.11122*, 2018.
- [30] T. Chen, P. Culbertson, and M. Schwager, “Catnips: Collision avoidance through neural implicit probabilistic scenes,” *arXiv preprint arXiv:2302.12931*, 2023.
- [31] 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 (ICRA)*. IEEE, 2023, pp. 10 511–10 517.
- [32] L. Yen-Chen, P. Florence, J. T. Barron, A. Rodriguez, P. Isola, and T.-Y. Lin, “Inerf: Inverting neural radiance fields for pose estimation,” in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 1323–1330.
- [33] D. Maggio, M. Abate, J. Shi, C. Mario, and L. Carlone, “Loc-nerf: Monte carlo localization using neural radiance fields,” in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 4018–4025.
- [34] Z. Zhu, S. Peng, V. Larsson, W. Xu, H. Bao, Z. Cui, M. R. Oswald, and M. Pollefeys, “Nice-SLAM: Neural implicit scalable encoding for SLAM,” in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022.
- [35] A. Rosinol, J. J. Leonard, and L. Carlone, “NeRF-SLAM: Real-Time Dense Monocular SLAM with Neural Radiance Fields,” *arXiv preprint arXiv:2210.13641*, 2022.
- [36] C. Yan, D. Qu, D. Wang, D. Xu, Z. Wang, B. Zhao, and X. Li, “GS-SLAM: Dense visual slam with 3d gaussian splatting,” *arXiv preprint arXiv:2311.11700*, 2023.
- [37] V. Yugay, Y. Li, T. Gevers, and M. R. Oswald, “Gaussian-SLAM: Photo-realistic dense slam with gaussian splatting,” *arXiv preprint arXiv:2312.10070*, 2023.

- [38] H. Matsuki, R. Murai, P. H. Kelly, and A. J. Davison, “Gaussian splatting slam,” *arXiv preprint arXiv:2312.06741*, 2023.
- [39] J. L. Schönberger and J.-M. Frahm, “Structure-from-motion revisited,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [40] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, “A fast procedure for computing the distance between complex objects in three-dimensional space,” *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.
- [41] S. Ruan, K. L. Poblete, H. Wu, Q. Ma, and G. S. Chirikjian, “Efficient path planning in narrow passages for robots with ellipsoidal components,” *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 110–127, 2022.
- [42] E. Eade, “Lie groups for 2d and 3d transformations,” URL <http://ethaneade.com/lie.pdf>, revised Dec, vol. 117, p. 118, 2013.
- [43] S. J. Julier and J. K. Uhlmann, “New extension of the kalman filter to nonlinear systems,” in *Signal processing, sensor fusion, and target recognition VI*, vol. 3068. Spie, 1997, pp. 182–193.
- [44] D. DeTone, T. Malisiewicz, and A. Rabinovich, “Superpoint: Self-supervised interest point detection and description,” in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2018, pp. 224–236.
- [45] P. Lindenberger, P. Sarlin, and M. Pollefeys, “Lightglue: Local feature matching at light speed. arxiv 2023,” *arXiv preprint arXiv:2306.13643*, 2023.
- [46] E. Karami, S. Prasad, and M. Shehata, “Image matching using sift, surf, brief and orb: performance comparison for distorted images,” *arXiv preprint arXiv:1710.02726*, 2017.
- [47] G. Terzakis and M. Lourakis, “A consistently fast and globally optimal solution to the perspective-n-point problem,” in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*. Springer, 2020, pp. 478–494.
- [48] S. Umeyama, “Least-squares estimation of transformation parameters between two point patterns,” *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 13, no. 04, pp. 376–380, 1991.
- [49] R. B. Rusu, N. Blodow, and M. Beetz, “Fast point feature histograms (fpfh) for 3d registration,” in *2009 IEEE international conference on robotics and automation*. IEEE, 2009, pp. 3212–3217.
- [50] J. Park, Q.-Y. Zhou, and V. Koltun, “Colored point cloud registration revisited,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 143–152.
- [51] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, “Learning transferable visual models from natural language supervision,” in *International Conference on Machine Learning (ICML)*. PMLR, 2021, pp. 8748–8763.
- [52] C. Zhou, C. C. Loy, and B. Dai, “Extract free dense labels from CLIP,” in *European Conference on Computer Vision (ECCV)*. Springer, 2022, pp. 696–712.
- [53] J. Kerr, C. M. Kim, K. Goldberg, A. Kanazawa, and M. Tancik, “LERF: Language embedded radiance fields,” in *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 19729–19739.
- [54] Y. Chen and G. Medioni, “Object modelling by registration of multiple range images,” *Image and vision computing*, vol. 10, no. 3, pp. 145–155, 1992.
- [55] V. Ye, M. Turkulainen, and the Nerfstudio team, “gsplat,” 2024. [Online]. Available: <https://github.com/nerfstudio-project/gsplat>
- [56] G. Wu, T. Yi, J. Fang, L. Xie, X. Zhang, W. Wei, W. Liu, Q. Tian, and X. Wang, “4d gaussian splatting for real-time dynamic scene rendering,” *arXiv preprint arXiv:2310.08528*, 2023.
- [57] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer, “D-nerf: Neural radiance fields for dynamic scenes,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 10318–10327.
- [58] C. Gao, A. Saraf, J. Kopf, and J.-B. Huang, “Dynamic view synthesis from dynamic monocular video,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 5712–5721.
- [59] N. A. (<https://math.stackexchange.com/users/3060/nickalger>), “Detect if two ellipses intersect,” Mathematics Stack Exchange, 2020, URL: <https://math.stackexchange.com/q/3678498> (version: 2021-05-09). [Online]. Available: <https://math.stackexchange.com/q/3678498>
- [60] J. Sherman, “Adjustment of an inverse matrix corresponding to changes in the elements of a given column or a given row of the original matrix,” *Annals of mathematical statistics*, vol. 20, no. 4, p. 621, 1949.
- [61] A. Thomas, F. Mastrogiovanni, and M. Baglietto, “Exact and bounded collision probability for motion planning under gaussian uncertainty,” *IEEE Robotics and Automation Letters*, vol. 7, no. 1, pp. 167–174, 2021.