CATNIPS: Collision Avoidance Through Neural Implicit Probabilistic Scenes

Timothy Chen[1], Preston Culbertson[2], Mac Schwager[1]

***Abstract*—We introduce a transformation of a Neural Radiance Field (NeRF) to an equivalent Poisson Point Process (PPP). This PPP transformation allows for rigorous quantification of uncertainty in NeRFs, in particular, for computing collision probabilities for a robot navigating through a NeRF environment. The PPP is a generalization of a probabilistic occupancy grid to the continuous volume and is fundamental to the volumetric ray-tracing model underlying radiance fields. Building upon this PPP representation, we present a chance-constrained trajectory optimization method for safe robot navigation in NeRFs. Our method relies on a voxel representation called the Probabilistic Unsafe Robot Region (PURR) that spatially fuses the chance constraint with the NeRF model to facilitate fast trajectory optimization. We then combine a graph-based search with a spline-based trajectory optimization to yield robot trajectories through the NeRF that are guaranteed to satisfy a user-specific collision probability. We validate our chance constrained planning method through simulations and hardware experiments, showing superior performance compared to prior works on trajectory planning in NeRF environments.**

***Index Terms*—Collision Avoidance, Robot Safety, Visual-Based Navigation, NeRFs**

I. Introduction

Constructing an environment model from onboard sensors, such as RGB(-D) cameras, lidar, or touch sensors, is a fundamental challenge for any autonomous system. Recently, Neural Radiance Fields (NeRFs) [1] have emerged as a promising 3D scene representation with potential applications in a variety of robotics domains including SLAM [2], pose estimation [3], [4], reinforcement learning [5], and grasping [6]. NeRFs offer several potential benefits over traditional scene representations: they can be trained using only monocular RGB images, they provide a continuous representation of obstacle geometry, and they are memory-efficient, especially considering the photo-realistic quality of their renders. Using current implementations [7], [8], NeRFs can be trained in seconds using only RGB images captured from monocular cameras, making onboard, online NeRF training a viable option for robotic systems.

However, NeRFs do not directly give information about spatial occupancy, which poses a challenge in using NeRF models for safe robot navigation. In other 3D scene representations, such as (watertight) triangle meshes [9], occupancy grids [10], or Signed Distance Fields (SDFs) [11], occupancy is well-defined and simple to query. NeRFs, however, do not admit simple point-wise occupancy queries, since they represent the scene geometry implicitly through a continuous volumetric density field. For this reason, integrating NeRF models into robotic planners with mathematical safety guarantees remains an open problem.

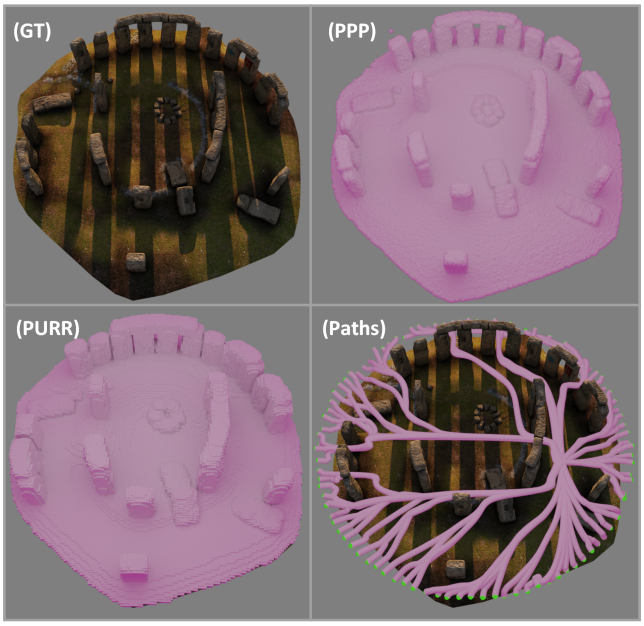

Fig. 1: (a) Ground-truth of the Stonehenge scene, (b) Poisson Point Process (PPP) of the scene represented as a point cloud, (c) Probabilistically Unsafe Robot Region (PURR) of scene, (d) Generated safe paths from our method (CATNIPS).

To this end, we develop a framework for robot trajectory planning that can generate trajectories through a NeRF scene with probabilistic safety guarantees. To do this, we propose a mathematical transformation of a NeRF to a Poisson Point Process (PPP), which allows for the rigorous computation of collision probabilities for a robot moving through a NeRF scene. We further introduce a novel scene representation, a Probabilistically Unsafe Robot Region (PURR), that convolves the robot geometry with the NeRF to yield a 3D map of all robot positions with collision probabilities less than a user-specified threshold. Finally, we propose a fast, chance-constrained trajectory planner that uses the PURR to ensure the trajectories are collision free up to the user-specified

*The NASA University Leadership Initiative (grant #80NSSC20M0163) provided funds to assist the authors with their research, but this article solely reflects the opinions and conclusions of its authors and not any NASA entity. Toyota Research Institute provided funds to support this work. The first author was supported by a NASA NSTGRO Fellowship, and the second author was supported on a NASA NSTRF Fellowship.

[1]Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305, USA {chengine, schwager}@stanford.edu

[2]Department of Mechanical and Civil Engineering, California Institute of Technology, Pasadena, CA 91125, USA, pculbert@caltech.edu

probability threshold. Our method, called CATNIPS, can compute probabilistically safe trajectories at more than 3 Hz. This is many times faster than existing NeRF-based trajectory planners that provide no safety guarantees [12].

The key theoretical advance underpinning our results is the novel transformation of the NeRF into a PPP. Existing works on radiance fields either ignore the underlying probabilistic interpretation of the field or treat it as a nuisance. A naive approach is to convert the NeRF representation into a more traditional deterministic mesh or occupancy representation. We argue that such conversions are computationally slow, and they destroy any potential mathematical safety guarantees for a downstream planner. For example, generating a triangle mesh (e.g., using marching cubes [13]) that represents a level set of the density field requires the arbitrary selection of a density cutoff value, and collapses the uncertainty represented by the density field into a binary occupancy measure. In contrast, our method computes rigorous collision probabilities using the NeRF density directly.

We provide simulation studies to show that our planner generates safe, but not overly-conservative, trajectories through the environment. We contrast our paths to those generated using a level-set based environment representation and those from prior work [12]. We find that the paths our method generates are more intuitive and easier to tune than these baselines, as collision is interpretably defined through violation of a collision probability as opposed to violation of an arbitrary level set of the density. We show our method to be real-time, replanning online at 3 Hz on a laptop computer, compared to the gradient descent-based planner proposed in [12], which requires approximately 2 seconds for replanning.

The rest of this paper is organized as follows. In Section II we discuss related work. In Section III we review background concepts from NeRFs, and in Section IV we derive the Poisson Point Process interpretation of the NeRF. In Section V we compute collision probabilities for a robot in a NeRF environment, and in Section VI we present our trajectory planning algorithm, CATNIPS. Section VII gives our simulation results and Conclusions are in Section VIII.

II. Related Work

Here we review the related literature in robot planning and control with NeRF representations, compare it with planning in a Signed Distance Function (SDF) representation, discuss other uses of NeRFs in robotics, and summarize chance-constrained planning.

A. Planning and Control with Onboard Sensing and SDFs

Planning and control based on onboard sensing has already yielded a large amount of literature. Typically these works present reactive control schemes [14], using the sensed depth directly to perform collision checking in real-time. These methods typically are myopic, reasoning only locally about the scene. An alternate approach is to construct a map of the environment using the depth measurements. Often a Signed Distance Field (SDF) is constructed from depth data [15], [16], which in this work is encoded within voxels. [16] also integrates their system onboard a quadrotor to validate their method. Such a representation is typical in dynamic robotic motion planning, providing fast collision checking and gradients in planning.

We believe NeRF is a promising alternative to more familiar 3D geometry representations like SDFs due to some key NeRF properties. We show that the NeRF inherently encodes uncertainty in the environment, whereas SDFs are typically deterministic. Moreover, we find that deep network SDFs are difficult to train, often requiring synthetic training points with heuristically generated, error-prone depth labels. In contrast, NeRFs can be supervised directly from RGB images, and can be trained reliably and quickly with NeRF training packages such as [8]. The modularity of NeRFs in perception pipelines, especially those involving visual data, is another benefit of NeRFs. However, this is not to say that the two cannot co-exist. There exists in the literature deep learning architectures that simultaneously learn SDF and NeRF outputs based on empirical consistency between the two (e.g., NeuS [17]). We hope that our probabilistic interpretation of NeRFs can help bridge the gap between these two representations and enable future pipelines to access advantages of both representations.

B. Planning and Control using NeRFs

Safety has been a largely unexplored topic in the NeRF literature, with only preliminary approaches being studied in simulation. The authors' previous work NeRF-Nav [12] presents a planner that avoids collisions in a NeRF environment model by avoiding high-density areas in the scene. An alternative work [18] instead uses the predicted depth map at sampled poses to enforce step-wise safety using a control barrier function. The two methods are not at odds, as the philosophy of [12] serves as a high-level planner that encourages non-myoptic behavior while [18] can be used as a safety filter for a myopic low-level controller interfacing directly with the system dynamics.

More specifically, NeRF-Nav [12] adapts trajectory optimization tools to plan trajectories for a robot through a NeRF environment. Collisions are discouraged with a penalty in the trajectory cost, but the probability of collision is not quantified or directly constrained. In this work, we instead rigorously quantify collision probabilities for a robot in a NeRF, and develop a trajectory planning method to satisfy user-defined chance constraints on collision. In addition, NeRF-Nav requires about 2 seconds for each online trajectory re-solve, while our proposed method requires about 0.3 seconds per online trajectory re-solve on similar computing hardware.

C. Other Uses of NeRFs in Robotics

Some works have considered NeRFs as a 3D scene representation for robotic grasping and manipulation. For example, Dex-NeRF [6] uses NeRF-rendered depth images to obtain higher-quality grasps for a robot manipulator than using a depth camera. Similarly, one can use dense object descriptors supervised with a NeRF model for robot grasping [4].

Some works have also considered SLAM and mapping using a NeRF map representation. The papers [2], [3] use the

photometric error between rendered and observed images to simultaneously optimize the NeRF weights and the robot/camera poses. The approach in [19] uses a grid-interpolation-decoder NeRF architecture in a similar SLAM pipeline. The work [20] proposes a combination of an existing visual odometry pipeline for camera trajectory estimation together with online NeRF training for the 3D scene. NeRFs have also been used for tracking the pose of a robot using an on-board camera and IMU. For example, iNeRF [3] finds a single camera pose from a single image and a pre-trained NeRF model, and [12] proposes a nonlinear optimization-based filter for tracking a trajectory of an on-board camera using a sequence of images and a pre-trained NeRF. Loc-NeRF [21] approaches a similar problem using a particle filter instead of a nonlinear optimization-based filter.

Other papers have considered active view planning for NeRFs. ActiveNeRF [22] treats the radiance value of the NeRF as Gaussian distributed random variables, and performs Bayesian filtering to find the next best view. An alternative work [23] uses disagreement among an ensemble of NeRFs to choose the next best camera view. Similarly, [24] uses ensembles in a next best view strategy while also adding ray-termination densities to the information gain metric. S-NeRF [25] uses variational inference to train a probability distribution over NeRFs for next best view selection. ActiveRMAP [26] considers a full informative trajectory planning pipeline for a robot moving through a NeRF. In contrast to our work here, they do not focus on the safety of the trajectories or on quantifying collision probabilities. Perhaps the closest in spirit in terms of modelling uncertainty is Bayes' Rays [27], which reasons locally about epistemic uncertainty (i.e. the distribution over NeRF parameters) and differs from our work that pins down the distribution that the NeRF model represents. However, we envision future efforts that incorporate both works to fully explain geometric uncertainty conditioned on RGB data.

Of course, many of these works would not be applicable to robotics if they were not real-time. Massive performance gains have been made to train NeRFs in real-time [7], [28], [29]. Moreover, NeRFs must be able to capture reality as well. They are known to suffer in quality when reconstructing rich, real environments over a large range of length scales. Attempts have been made to fix this issue by extrapolating over the entire camera frustum rather than a ray [30], [31].

D. Chance-constrained Planning

Outside of NeRFs, there exists a large literature on trajectory planning for robots that seeks to impose constraints on the probability of collision when the underlying scene geometry is unknown; this approach is known as chance-constrained planning. The robot state is typically modeled as stochastic, while the map is typically considered to be known deterministically. Some works do consider uncertainty in both the map and the robot state, but they typically rely on a linear system or Gaussian noise assumption to make computation convex or analytical and efficient to solve. Du Toit and Burdick [32] assume Gaussian-distributed obstacle states, and approximates the collision probability as constant over the robot body (suitable only for small robots). Blackmore et al. [33] encode the probability of collision with faces of polytopic obstacles as a linear constraint, but the resulting trajectory optimization is a combinatorial problem, making it difficult to solve quickly. Zhu and Alonso-Mora [34] incorporate this linear probabilistic constraint into RRT. Luders et al. [35] again use this linear constraint in an MPC framework with nonlinear dynamics, executed on real hardware with dynamic obstacles and extended to a multi-agent context. None of these methods consider NeRF environment models, which is our focus here.

III. Neural Radiance Fields (NeRFs)

In this section, we introduce the mathematical preliminaries and notation used in NeRFs. For clarity, we use bold face for vector variables and functions that output vectors, and non-bold text for scalar variables, functions that output scalars, and—in some instances—rotations.

A NeRF is a neural network that stores a density and color field over the 3D environment. When coupled with a differentiable image rendering model (usually a differentiable version of ray tracing), the NeRF can be trained from a collection of RGB images with known camera poses, and can generate photo-realistic synthetic images rendered from camera view points that are different from the training images.

More specifically, the NeRF is a pair of functions $(\rho(\mathbf{p}), c(\mathbf{p}, \mathbf{d}))$. The density function, $\rho : \mathbb{R}^3 \mapsto \mathbb{R}_{\geq 0}$, maps a 3D location $\mathbf{p} = (x, y, z)$ to a non-negative density value ρ that encodes the differential probability of a light ray stopping at that point.[1] The radiance (i.e., RGB color) function $\mathbf{c} : \mathbb{R}^3 \times \mathbb{R}^2 \mapsto \mathbb{R}^3$ maps a 3D location $\mathbf{p} = (x, y, z)$ and camera view direction $\mathbf{d} \in \{\mathbf{x} \in \mathbb{R}^3 \mid ||\mathbf{x}|| = 1\}$ (alternatively parameterized as a 2D vector of angles (θ, ϕ)) to an emitted RGB color $\mathbf{c}$ represented as a vector in $\mathbb{R}^3$. In this paper, we focus specifically on the density function $\rho(\mathbf{p})$ as a proxy for occupancy, which should ideally be zero in free space and take on large values in occupied space. We use this $\rho(\mathbf{p})$ function as a map representation for planning robot trajectories. We also define $\mathbf{C}(\mathbf{o}, \mathbf{d}) \in [0, 1]^3$ as the rendered pixel color in an image when taking the expected color value from the NeRF along a ray $\mathbf{r}(t; \mathbf{o}, \mathbf{d})$ with camera origin $\mathbf{o}$ and pixel orientation $\mathbf{d}$, where $\mathbf{r}(t) = \mathbf{o} + t \cdot \mathbf{d}$. The rendered color is given by

$$\mathbf{C}(\mathbf{o}, \mathbf{d}) = \int_{t_n}^{t_f} \rho\left(\mathbf{r}(t)\right)\ e^{-\int_{t_n}^{t} \rho(\mathbf{r}(\tau)) d\tau}\ \mathbf{c}(\mathbf{r}(t), \mathbf{d})\ dt, \quad (1)$$

where we only integrate points along the ray between t_n and t_f (i.e. the near and far planes). In practice, this integral is evaluated numerically using Monte Carlo integration with stratified sampling. The resulting rendering equation (1) is differentiable.

[1] This density field can be stored entirely as a multi-layer perceptron (MLP), as in the original NeRF work [1], as a function interpolated on a discrete voxel grid [28], or using a combination of interpolated voxel features and an MLP decoder [7], [29]. Our method can work with any of these representations.

A rendered image I_i is then an array of pixel colors associated with a single camera pose, where the color of pixel j in image I_i is given by $\mathbf{C}(\mathbf{o}_i, \mathbf{d}_{ij})$ with associated origin $\mathbf{o}_i$ (determined by the camera) and direction $\mathbf{d}_{ij}$ computed with an angular offset from the camera optical axis for pixel j. We denote the set of pixel indices for image I_i as $\mathcal{I}_i$. The corresponding ground truth image $\bar{I}_i$ is an array of pixels with colors $\bar{\mathbf{C}}_{\mathbf{ij}}$. A dataset D for training a NeRF consists of a collection of such ground truth images with known poses. The parameters of the NeRF are trained by minimizing the loss function

$$J(\boldsymbol{\theta}) = \frac{1}{|D|} \sum_{i \in D} \frac{1}{|\mathcal{I}_i|} \sum_{j \in \mathcal{I}_i} ||\mathbf{C}(\mathbf{o}_i, \mathbf{d}_{ij}; \boldsymbol{\theta}) - \bar{\mathbf{C}}_{\mathbf{ij}}||_2^2, \quad (2)$$

where $\boldsymbol{\theta}$ are the parameters of the neural networks representing the density and radiance fields ρ and $\mathbf{c}$, which appear in the computation of the pixel color $\mathbf{C}(\mathbf{o}_i, \mathbf{d}_{ij}; \boldsymbol{\theta})$ through the rendering equation (1). This mean squared error is called the photometric error (or photomoetric loss) and is optimized with standard stochastic gradient descent tools in, e.g., Pytorch. Intuitively, the goal is to train the network so that the synthetic images generated from the NeRF match the training images at the specified camera poses as closely as possible.

While the camera poses are required to find $\mathbf{o}_i$ and $\mathbf{d}_{ij}$ for each pixel to train the NeRF, a standard pipeline has emerged that takes images without camera poses, uses a classical structure-from-motion algorithm (e.g., COLMAP [36]) to estimate the camera poses, and supervises the NeRF training with these poses. Recent methods also optimize the camera poses jointly with the NeRF weights to improve performance [8].

Hence, in practice a NeRF model can be obtained from only RGB images (without camera poses). However the quality and extent of the NeRF is limited by the quality of the training images, and the volume covered by those images. Few images, with low resolution, low photographic quality, and poor coverage will yield a poor-quality NeRF. A large number of sharp, high-resolution images from a rich diversity of view points will yield a high-quality NeRF. Our goal is to accurately quantify collision risk for a robot navigating through the scene regardless of the quality of the trained NeRF. With our approach, the same robot pose in the same 3D scene may have a high collision probability in a poor-quality NeRF, and a low collision probability in a high-quality NeRF. The probability of collision is itself an expression of the NeRF quality in the vicinity of the robot.

IV. NeRF Density as a Poission Point Process

In this section, we show that a NeRF density field can be transformed into the density of a Poisson Point Process (PPP), and the NeRF color and density fields together give rise to a "marked" PPP [37], [38]. To do this, we demonstrate that the NeRF volumetric rendering equation is precisely the computation that is required to compute expected pixel color if the color and density under this marked PPP model. Training the NeRF can be interpreted as fitting the PPP density parameters through moment matching on the expected pixel color.

This connection is significant since the PPP derived from the NeRF density field enables computation of probabilistic quantities, such as the probability of a given volume being occupied (e.g., of a robot body colliding with the NeRF), or the entropy in the NeRF model. This also settles a debate in the literature about whether the NeRF density can be probabilistically (it can), and paves the way for practical utility in other domains beyond safety (e.g., in active sensing and active view planning). In short, we find that the NeRF density encodes a probabilistic model of the geometry of the scene, the uncertainties of which can be rigorously quantified through an underlying PPP.

A. Poisson Point Processes

Here we review the definition and properties of the Poisson Point Process (PPP), a stochastic process that models the distribution of a random collection of points in a continuous space. Much of this discussion is drawn from [37], to which we refer the reader for a more detailed and rigorous treatment.

First, we recall that a discrete random variable (RV) N that takes values in $\mathbb{N}$ is said to have a Poisson distribution with parameter $\lambda \geq 0$ if its probability mass function is given by

$$Pr(N = m) = \frac{\lambda^m \exp(-\lambda)}{m!}.$$

Poisson RVs are often used to model the distribution of the number of discrete events in a fixed amount of time (e.g., customers arriving at a store), or over a fixed region of space (e.g. the number of rides hailed daily in a given neighborhood). The PPP naturally extends this concept to the distribution over the number of points in any subset of a multi-dimensional Euclidean space.

Definition 1 (Poisson Point Process)**.** *Consider a random process N on $\mathbb{R}^n$ that maps subsets[2] $B \subset \mathbb{R}^n$ of the state space to the random number $N(B)$ of points that lie in B. We say N is a Poisson Point Process (PPP) with intensity $\lambda : \mathbb{R}^n \mapsto \mathbb{R}_+$ if:*

(i) The number of points $N(B)$ that lie in B is a Poisson RV with distribution

$$Pr(N(B) = m) = \frac{\Lambda(B)^m \exp(-\Lambda(B))}{m!},$$

where $\Lambda(B) = \int_{\mathbf{x} \in B} \lambda(\mathbf{x})\, d\mathbf{x}$.

(ii) For k disjoint subsets $B_1, \ldots B_k \subset \mathbb{R}^n$, the number of points in each subset, $N(B_1), \ldots, N(B_k)$, are independent RVs.

This is sometimes refered to as the *inhomogeneous* PPP since the intensity λ is a function of the spatial variable $\mathbf{x}$. If the intensity is constant over $\mathbf{x}$, this is called the *homogeneous* PPP.

The PPP encodes the randomness over both the *number* and the *location* of random points. An important quantity for such processes is the "void probability," or the probability that a given set B is empty. The void probability is given by

$$Pr(N(B) = 0) = \exp\left[-\int_B \lambda(\mathbf{x})\, d\mathbf{x}\right]. \quad (3)$$

[2]The subsets B must be Lebesgue measureable.

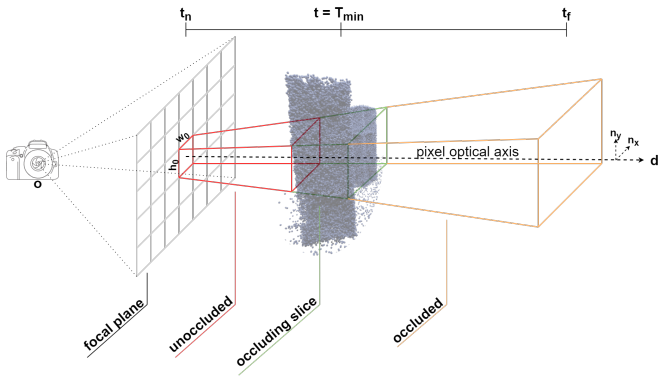

Fig. 2: In the rendering process, the probability that the pixel color takes on the color of the infintesimally small occluding slice (green) is given by the probability that all slices in the region preceding the slice (red) are unoccluded. Then, the pixel color is the expectation of the color taken by varying the position of the occluding slice along the ray.

Thus, intuitively, the void probability shrinks as either the intensity λ increases, or the set B grows larger.

It is also important to note that through the Poisson distribution, the expected number of points in the set B is identical to the integral of the intensity, in other words,

$$\mathbb{E}\left[N(B)\right] = \int_B \lambda(\mathbf{x})\, d\mathbf{x}. \tag{4}$$

When reasoning about collision probability, we want each point associated with the PPP to have a corresponding volume. Using the above fact (4), we can consider points or particles of arbitrary size by weighting the PPP accordingly.

Given some reference particle size V_{ref} associated with the initial PPP, a desired particle size V_d, an integration domain B, and the expected volume occupied by all the particles $\mathbb{E}\left[V_{total}\right]$, we can retrieve $\mathbb{E}\left[N_d(B)\right]$, the expected number of particles of size V_d, by conservation of $\mathbb{E}\left[V_{total}\right]$. Namely,

$$\begin{aligned} V_{ref}\mathbb{E}\left[N(B)\right] &= V_{ref}\int_B \lambda(\mathbf{x})\, d\mathbf{x} = \mathbb{E}\left[V_{total}\right] \\ &= V_d\mathbb{E}\left[N_d(B)\right], \\ \mathbb{E}\left[N_d(B)\right] &= \frac{\int_B \lambda(\mathbf{x})\, d\mathbf{x}}{V_d/V_{ref}}. \end{aligned} \tag{5}$$

Therefore, a PPP with the same expected occupied volume can be produced using differing particle sizes by simply scaling the density.

Finally, we note that PPPs may be "marked" or "colored" with various quantities using a deterministic labeling function $\mathbf{c}(\mathbf{x}) : \mathbb{R}^n \mapsto \mathcal{C}$. Using the statistics of the underlying PPP, it is straightforward to compute the statistics of the labels for the points appearing in a set; see [37, Ch. 5] for a detailed discussion.

B. "Rendering" a Marked PPP

The intensity function of a PPP admits an infinitesimal interpretation: $\lambda(\mathbf{x})\, d\mathbf{x}$ is the probability that a point of the process lies within an infinitesimal volume $d\mathbf{x}$ centered at $\mathbf{x}$. This is closely related to the interpretation of the density field offered by the original NeRF authors [1]: "$\rho(\mathbf{x})$ defines the infinitesimal probability of a ray terminating at a given point $\mathbf{x} \in \mathbb{R}^3$." In this section, we show the volumetric rendering procedure introduced in [1] in fact computes an expected color of a marked PPP along a given ray.

Our key problem is how to relate the ray tracing used in NeRF, a 1D process, to the 3D measure of occupancy, $\lambda(\mathbf{x})$ in the PPP. We consider the occupancy swept by a frustum (the pyramid of light that projects onto a single pixel patch). In particular, when generating the color of a particular pixel from a marked PPP, we extend a pyramid along the pixel's ray (Fig. 2), and return the color of the first point of the process encountered along the ray in expectation.

More specifically, for each pixel in the image, with associated ray $\mathbf{r}(\mathbf{o}, \mathbf{d})$, we consider the frustum (Fig. 2, red) parameterized by a length $t \in [t_n, t_f]$,

$$\begin{aligned} \mathcal{F}(t) \equiv \Big\{ \mathbf{o} + \mathbf{d}\tau + x\hat{\mathbf{n}}_x + y\hat{\mathbf{n}}_y \mid |x| \le \frac{w(\tau)}{2}, \\ |y| \le \frac{h(\tau)}{2}, \tau \in [t_n, t] \Big\} \end{aligned} \tag{6}$$

where $\hat{\mathbf{n}}_x, \hat{\mathbf{n}}_y$ are unit vectors orthogonal to $\mathbf{d}$ forming a basis in the image plane and h, w are the height and width of the frustrum cross-section, respectively, starting from the size of the pixel (h_0, w_0) on the image plane.

However, there still remains a modeling choice: given a fixed particle size V_{ref}, how many particles must be present in a given cross-section of the frustum for light to be occluded? We say the ray is occluded when the combined frontal area of the particles present at depth t occupies a given fraction $\gamma \in (0, 1]$ of the frustum's cross-section.

Since the frustum's area $A(t)$ varies with depth, for particles of a fixed size, the number of particles needed to occlude the ray would also vary with t. However, as previously discussed, reweighting a PPP is equivalent to changing the particle size. Thus, in this work we consider "dimensionless" particles whose (projected) area on the frustum is exactly $\gamma A(t)$ (so only one particle is needed to occlude the ray) by reweighting the PPP density accordingly along the ray. Thus, we say the ray terminates at the depth of the first "dimensionless" particle encountered along the ray.

C. Equivalence of NeRF and PPP Rendering

This brings us to our main result. Here we show that, under appropriate assumptions on the distribution of the training rays and the spatial variation of the NeRF density and color, the color of a given pixel in an image rendered from a NeRF (1) is exactly the expected color of the same pixel rendered from a PPP with (scaled) intensity equal to the NeRF density ρ, and marking equal to the NeRF radiance $\mathbf{c}$. We then provide intuition on the spatial relation between λ and ρ.

Assumption 1 (PPP Smoothness)**.** *Consider a Poisson Point Process $\lambda(\mathbf{x})$ and color marking $\mathbf{c}(\mathbf{x}, \mathbf{d})$. We assume the average of the PPP density over any ball B_ϵ, with center $\mathbf{x}_\epsilon$, is equal to the value of the PPP density at the center of the ball, where ϵ is the minimum radius ball required to contain a pixel projected from the near plane onto the far plane of*

the NeRF scene. We also assume the color field $\mathbf{c}(\mathbf{x}, \mathbf{d})$ *is approximately constant over any* B_ϵ*. Specifically,*

$$\int_{B_\epsilon} \lambda(\mathbf{x})\, d\mathbf{x} = \lambda(\mathbf{x}_\epsilon) V_{B_\epsilon} \tag{7}$$

and

$$\forall (\mathbf{x}, \mathbf{y}) \in B_\epsilon :\ ||\mathbf{c}(\mathbf{x}, \mathbf{d}) - \mathbf{c}(\mathbf{y}, \mathbf{d})|| = 0, \tag{8}$$

where B_ϵ *is the smallest ball that can contain an image pixel projected from the near plane onto the far plane of the NeRF scene.*

This assumption states that the variation in the PPP over a small ball integrates to 0, while the color is constant in that same region.[3] These are both mathematical idealizations which are unlikely to hold exactly in practice. However, we find empirically that these assumptions are very close to being satisfied, ultimately yielding well-calibrated collision probabilities. For example, for a scene with length scale 2 meters, a camera with focal length 50 mm, and a 1000×1000 pixel image, the far plane pixel has side length on the order of 10^{-3}m, requiring $\epsilon = \sqrt{2}$ mm. Empirically, this is consistent with the smallest resolution of detail in a well-trained NeRF scene of a 2 m^3 volume.

Assumption 1 is reasonable due to the loss of information when encoding the continuous environment into the discretized observation space of pixelated images. Due to the resolution of the camera, color is constant across a pixel, hence we do not have information to distinguish generating environments whose colors only differ over the length scales of a single pixel. Since reconstructing the continuous geometry given the pixel-discretized images is ill-posed, we see this smoothness requirement as a kind of regularization prior for the density and color fields.

Moreover, we tolerate stricter assumptions on the color because, in practice, the radiance field (i.e., color) is also smoother than the density field. The primary reason is that the radiance field is defined even in regions of empty space and is therefore allowed to smoothly change across surfaces; on the other hand, the density must change more sharply across surfaces in order to reflect the underlying discrete change in geometry. This is indeed reflected in the literature, where the original NeRF work [1] uses a smaller network to model color and later extensions like [28] replace the network with smooth low-order spherical harmonics.

Assumption 2 (Ray Redundancy)**.** *Given a dataset of training rays derived from training images, poses, and camera intrinsics, no two rays intersect.*

This is a weak assumption as ray intersection is a zero-measure event. Moreover, floating point precision and noise in the poses further reduces the likelihood of ray intersection.

Given these assumptions, we now state our main result.

Proposition 1 (Rendering of PPP)**.** *Consider a PPP* $\lambda(\mathbf{x})$ *and radiance* $\mathbf{c}(\mathbf{x}, \mathbf{d})$ *and let the radiance satisfy (8) from Assumption 1. Then, the expected color of a pixel rendered from the PPP matches the form of the rendering equation (1).*

Proof. Let us consider a ray $\mathbf{r}(t) = \mathbf{o} + t \cdot \mathbf{d}$, where $t \in [t_n, t_f]$. We consider again the corresponding frustum $\mathcal{F}(t)$ (6), the pyramid created by sweeping the scaled pixel area along $\mathbf{r}$ from t_n to t.

As discussed in our rendering model, we consider "dimensionless" particles whose projected area is $\gamma A(t)$ (so the ray is occluded by the first particle encountered). Thus, the intensity/expected number of particles of this slice $\delta\mathcal{F}(t)$ for those of the occluding size ($V_d = \gamma V_{\delta\mathcal{F}}(t)$) is defined as

$$\begin{aligned}
&\Lambda\big(\delta\mathcal{F}(t)\big) = \int_{\mathbf{x} \in \delta\mathcal{F}(t)} \frac{V_{ref}\lambda(\mathbf{x})}{\gamma V_{\delta\mathcal{F}}(t)}\, d\mathbf{x} \\
&= \int_{t-\delta t/2}^{t+\delta t/2} \int_{(x,y)\in A(\tau)} \frac{V_{ref}\lambda(\mathbf{r}(\tau) + x\hat{\mathbf{n}}_x + y\hat{\mathbf{n}}_y)}{\gamma V_{\delta\mathcal{F}}(t)}\, dx\, dy\, d\tau \\
&= \frac{\delta t \int_{A(t)} A_{ref}\delta t \lambda(\mathbf{r}(t) + x\hat{\mathbf{n}}_x + y\hat{\mathbf{n}}_y)\, dx\, dy}{\gamma A(t)\delta t}
\end{aligned}$$

if the reference particle is some small ball of size $V_{ref} = A_{ref}\delta t$.

Hence, the void probability of the slice (3) (equivalently, the probability of no occlusion) is

$$\begin{aligned}
&Pr(N(\delta\mathcal{F}(t)) = 0) = \\
&\exp\left[-\frac{\int_{A(t)} A_{ref}\lambda(\mathbf{r}(t) + x\hat{\mathbf{n}}_x + y\hat{\mathbf{n}}_y)\, dx\, dy}{\gamma A(t)} \delta t \right].
\end{aligned}$$

Now we consider the event where any slice of the frustum is occluded, up to a given depth t. To do this, we divide the frustum into smaller subsections along its length. Because the number of particles in disjoint subsets are independent by definition of the PPP (Def. 1(ii)), then the probability of occlusion of each section is independent of that of other sections. Hence, the probability that the frustum up to t is not occluded requires all sections to not be occluded,

$$\begin{aligned}
&\prod_t Pr(N(\delta\mathcal{F}(t)) = 0) \\
&= \prod_t \exp\left[-\frac{\int_{A(t)} A_{ref}\lambda(\mathbf{r}(t) + x\hat{\mathbf{n}}_x + y\hat{\mathbf{n}}_y)\, dx\, dy}{\gamma A(t)} \delta t \right] \\
&= \exp\left[-\sum_t \frac{\int_{A(t)} A_{ref}\lambda(\mathbf{r}(t) + x\hat{\mathbf{n}}_x + y\hat{\mathbf{n}}_y)\, dx\, dy}{\gamma A(t)} \delta t \right].
\end{aligned}$$

In the limit as the section widths δt approach zero, they become slices, and the summation becomes an integral. Thus, the probability that the frustum up to t is occluded is

$$\begin{aligned}
&Pr(\mathcal{F}(t) \text{ not occluded}) \\
&= \exp\left[-\int_t \frac{\int_{A(\tau)} A_{ref}\lambda(\mathbf{r}(\tau) + x\hat{\mathbf{n}}_x + y\hat{\mathbf{n}}_y)\, dx\, dy}{\gamma A(\tau)} d\tau \right].
\end{aligned}$$

For notational simplicity, we henceforth denote the surface integral divided by the scaled area of the slice by $\kappa(\tau)$.

Let us now consider a random variable T_{min} which defines the distance of the first occluding slice (denoted green in Fig. 2). We can define the cumulative distribution function of this

[3]Note that we do not require the density assumption for NeRF-PPP equivalence (Prop. 1, 2). Its use is in extracting an approximate scaling factor to transform between the density ρ and PPP intensity λ (Cor. 1).

variable, for some $t > t_n$, $T_{\text{min}} \leq t$ as the probability that $\mathcal{F}(t)$ is occluded. Thus, the CDF of T_{min} can be defined using the above equation,

$$\begin{aligned} Pr(T_{\text{min}} \leq t) \equiv F_{T_{\text{min}}}(t) &= 1 - P\left(\mathcal{F}(t) \text{ not occluded}\right) \\ &= 1 - \exp\left[-\int_{t_n}^{t} \kappa(\tau)\, d\tau\right]. \end{aligned}$$

We can then compute the PDF of T_{min} by differentiating $F_{T_{\text{min}}}$ with respect to t, yielding,

$$\begin{aligned} f_{T_{\text{min}}}(t) &= \frac{d}{dt} F_{T_{\text{min}}}(t) \\ &= \kappa(t) \exp\left[-\int_{t_n}^{t} \kappa(\tau)\, d\tau\right]. \end{aligned} \tag{9}$$

This defines a probability distribution over the extent of the unoccluded region.

Finally, due to Assumption 1, the color of the slice returned by the PPP rendering is equal to the marking function evaluated at $\mathbf{r}(T_{\text{min}})$. Thus, we can compute the expected color of the PPP rendering by computing the expectation of $\mathbf{c}(\mathbf{r}(T_{\text{min}}))$, yielding

$$\begin{aligned} \mathbf{C}(\mathbf{r}) &= \mathbb{E}\left[\mathbf{c}(\mathbf{r}(T_{\text{min}}))\right], \\ &= \int_{t_n}^{t_f} \kappa(t)\mathbf{c}(\mathbf{r}(t)) \exp\left[-\int_{t_n}^{t} \kappa(\tau)\, d\tau\right] dt. \end{aligned}$$

The PPP expected color matches exactly the expression given in (1) from the original NeRF paper [1], completing the proof. □

Proposition 2 (NeRF-PPP Equivalence)**.** *Consider a NeRF with density $\rho(\mathbf{x})$ and let Assumption 2 hold. Then the NeRF is a locally area-averaged PPP.*

Proof. Following from the above proof, we make equivalences between the two rendering equations for all training rays, namely

$$\forall\ \mathbf{r}(t; \mathbf{o}, \mathbf{d}) : \\ \rho(\mathbf{r}(t)) = \frac{\int_{A(t)} A_{ref} \lambda(\mathbf{r}(t) + x\hat{\mathbf{n}}_x + y\hat{\mathbf{n}}_y)\, dx\, dy}{\gamma A(t)}.$$

Note that we require Assumption 2 because when two rays intersect, the left hand side of the above equation is necessarily identical for both rays, yet the right hand side may not be. Specifically, the integration domains for the two rays at the intersection point may not be identical, hence the numerator and denominator on the right hand side are not the same for both rays. Moreover, note that for $\gamma = 1$ (full occlusion), the density is the area-averaged number of particles over the slice $A(t)$. □

Crucially, this matches the intuition on the NeRF density proposed by [39], [1]. However, our derivation is more general than that of [39], which assumes a constant-area frustum. Although we used a pyramidal frustum for illustration purposes, note that our derivation does not assume the form of $A(t)$ (e.g. rectangular, circular) and therefore the shape of the frustum, so long as the frustum can be decomposed into disjoint slices that are themselves connected sets. Finally, our derivation suggests a more general rendering equation since we had to assume local homogeneity of color to retrieve (1). Without this limitation, we could derive a more expressive and accurate rendering equation. Moreover, our derivation even proposes a parameter γ that can be tuned to more precisely define occlusion and perhaps increase the fidelity of the render.

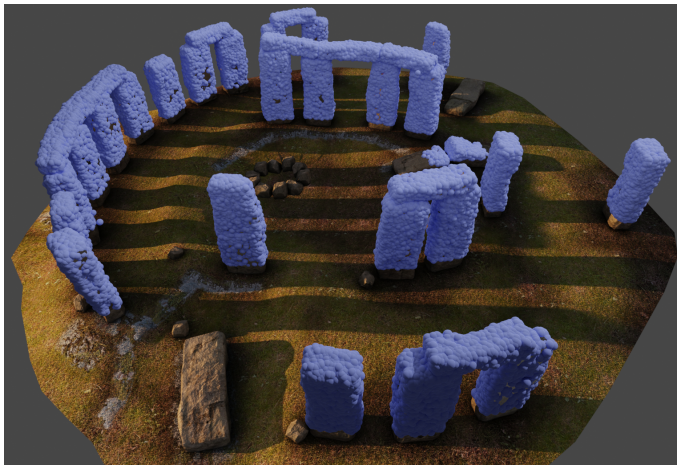

Fig. 3: Overlay of a realization of the PPP with the ground-truth mesh of Stonehenge. The two have strong spatial agreement.

Corollary 1. *By (7) from Assumption 1, $\rho(x) = A_{ref}\frac{\lambda(x)}{\gamma}$, where $0 < \gamma \leq 1$, so the NeRF density is related to an equivalent PPP through a constant scale factor A_{ref}/γ.*

In general, the constant A_{ref}/γ is unknown, however we show in Sec. V below that we do not need its exact value to compute the collision probability for a robot body.

Having shown that a PPP can be derived from the NeRF, we visually show this relationship in Fig. 3, where we generate a point cloud randomly drawn from the PPP (blue spheres) superimposed on the NeRF rendering of the same scene. The correspondence in geometry of the NeRF scene and the PPP point cloud is clear.

Note that while area-averaging of λ to ρ yielded the rendered color as a line integral as opposed to a volume integral, we have lost information about the λ field (i.e. the reference particle A_{ref} and occlusion γ) when using a learning framework to learn ρ. In fact, we conjecture the observed aliasing phenomena in which NeRFs fail at different scales [30] is due to this averaging scheme. The success of works like Mip-NeRF [30] that reason about the pixel not as a projection of a ray, but along a frustrum and evaluating rendering as a volume integral (i.e., learning λ rather than ρ) gives us reason to believe that learning the parameters of the PPP directly could yield higher quality geometry.

D. Discussion of PPP Interpretation

An important advantage of taking a PPP interpretation of the NeRF density is that it allows us to leverage well-studied properties of PPPs when reasoning about NeRFs. Specifically,

uncertainty metrics such as likelihood and entropy are well-defined for PPPs. Suppose we measure a point cloud of our environment using an onboard lidar or depth camera; i.e., for a set of rays $\{\mathbf{r}_1, \dots, \mathbf{r}_k\}$ we obtain noiseless depth measurements $\{d_1, \dots d_k\}$. We can write the likelihood of obtaining these depths under our NeRF using (9),

$$\log P(d_1, \dots d_k) = \sum_{i=0}^{k} \left[\log \rho(\mathbf{r}(d_i)) - \int_{t_n}^{d_i} \rho(\mathbf{r}(t)) dt \right]. \tag{10}$$

If, say, the robot's state is uncertain, and the depth measurements are corrupted by noise, this likelihood can be used in the computation of Bayes' rule for pose estimation. Since previous literature on NeRFs contained no such likelihood interpretation, existing approaches to state estimation [3], [12] instead only minimize a photometric loss as a proxy for maximum likelihood estimation.

Further, notions of entropy and mutual information can be generalized to point processes, as discussed in [40]. In particular, the entropy of a point process over a set B is defined as

$$\mathbb{H}(B) \equiv \int_B \lambda(\mathbf{x})(1 - \log \lambda(\mathbf{x})) d\mathbf{x} \tag{11}$$

$$= \frac{\gamma}{A_{ref}} \int_B \rho(\mathbf{x})(1 + \log A_{ref} - \log \gamma - \log \rho(\mathbf{x})) d\mathbf{x}. \tag{12}$$

Thus, $\mathbb{H}$ can be used as a measure of the uncertainty of a NeRF over some set B, which would be useful to reason about which parts of the NeRF are poorly-supervised (i.e., have high entropy) for problems such as next-best-view selection and active perception.

Finally, the PPP interpretation of the NeRF allows us to provide a novel perspective on NeRF training: minimizing the photometric loss (2) proposed in [1] can be interpreted as performing moment-matching [41, Ch. 4] on the first moment of the color distribution along the supervised rays. In particular, the photometric loss will be zero if the color rendered from the NeRF (i.e., an expected color along the ray of a PPP) matches the sample distribution (i.e., the color label in the dataset). We believe our probabilistic interpretation of the NeRF density could also inspire other loss functions beyond (2), to perform other methods of parameter estimation such as maximum likelihood estimation, expectation maximization, and so on.

V. Computing Collision Probability with NeRF Scenes

To leverage the probabilistic interpretation of NeRFs to evaluate the probability of collision between a robot body and the NeRF, we first define $B(\mathbf{p}, \mathbf{R}) \subset \mathbb{R}^n$ as the robot body parameterized by its pose $(\mathbf{p}, \mathbf{R})$ (the set of points occupied by the robot with position $\mathbf{p} \in \mathbb{R}^3$ and orientation $\mathbf{R} \in \mathbb{SO}(3)$), and consider an environment represented as a NeRF with density field $\rho(\mathbf{p})$, which we have shown to be related to the PPP field through a constant scale factor $\lambda(\mathbf{p}) = \frac{\gamma \rho(\mathbf{p})}{A_{ref}}$.

We define collision probability, or the probability that a collection of points from a PPP intersects with the robot body, as the probability that at most some volume V_{max} from the NeRF can exist within the robot volume. We call V_{max} the specified or allowable inter-pentration volume. Given some auxiliary particle (which may not be the same as the reference particle) that is user-defined and has some volume $V_{aux} < V_{max}$, we can solve for the maximum number of auxiliary particles that should exist in the robot volume $N_{aux}^{max} = \frac{V_{max}}{V_{aux}}$. The definition of this new type of particle is necessary because one does not have access to the reference particle dimensions of the underlying PPP. We show that this knowledge is not necessary to compute collision. Nonetheless, we are simply solving for the Cumulative Distribution Function (CDF) of the Poisson Point Process associated with the auxiliary particle (i.e. the number of particles that exist in the robot body),

$$Pr(X \leq N_{aux}^{max}; \Lambda_B) = \exp^{-\Lambda_B} \sum_{i=0}^{\lfloor N_{aux}^{max} \rfloor} \frac{\Lambda_B^i}{i!}, \tag{13}$$

where Λ_B is the intensity for the auxiliary particle over the robot body.

Note that Λ_B is the PPP associated with the auxiliary particle and not the reference; therefore, we must scale the NeRF density appropriately,

$$\Lambda_B = \int_B \lambda_{aux}(\mathbf{x})\, d\mathbf{x} = \frac{V_{ref}}{V_{aux}} \int_B \lambda(\mathbf{x})\, d\mathbf{x}. \tag{14}$$

Recall that $V_{ref} = A_{ref} \delta t$ and assume an identical form for the auxiliary volume $V_{aux} = A_{aux} \delta t$. Additionally, we can substitute the relationship between λ and ρ (Cor. 1) to get the following

$$\Lambda_B = \frac{A_{ref} \delta t}{A_{aux} \delta t} \int_B \frac{\gamma \rho(\mathbf{x})}{A_{ref}}\, d\mathbf{x} = \frac{\gamma}{A_{aux}} \int_B \rho(\mathbf{x})\, d\mathbf{x}. \tag{15}$$

This brings us to the first formal definition of NeRF collision.

Definition 2 (Probabilisitically Safe)**.** *A robot body parametrized by its pose $B(\mathbf{p}, \mathbf{R})$ is probabilistically safe if the collision probability given in (13) and (15) satisfies $Pr(N(B(\mathbf{p}, \mathbf{R})) \leq N_{aux}^{max}; \Lambda_B) \geq \sigma$, for desired probability threshold σ, auxiliary particle associated with the specified inter-penetration volume, and occlusion threshold.*

More succinctly, we consider a robot as probabilistically safe if the interpenetration volume between the robot and the NeRF is less that a threshold V_{max} with probability at least σ. A major remaining question is how to choose the auxiliary particle size and occlusion threshold γ. Since we are assuming smoothness on the length scales of a pixel, it is appropriate to use an auxiliary particle of this size. We use the approximate size of a pixel at the near plane and the sampling distance along a ray to find the dimensions of the auxiliary particle. Specifically, given a camera with focal length 50 mm, a scene of length 2 m, a FOV of 90°, and a 1000 by 1000 image, the pixel side on the image plane is 10^{-4} m, hence for a square pixel, $A_{aux} = 10^{-8}$ m^2. If the pixel size is the 2D resolution of the image, we can think of δt as the sampling resolution used to learn the NeRF, hence V_{aux} the 3D resolution of

the NeRF learned from data. Typically, there are anywhere between 100 to 200 sample points along a ray between the near and far plane, so we choose $\delta t = 20$ mm, yielding $V_{aux} = 2 \cdot 10^{-8}$ m^3. In fact, the CDF is relatively insensitive to A_{aux}, as both N_{aux}^{max} and Λ_B are scaled the same amount for changes in A_{aux}. γ is essentially a modelling parameter as there is no way to know what the environment defines as an occlusion event. However, in the interest of safety and interpretability, we set $\gamma = 1$ so that it is meaningful (i.e. full occlusion) and such that it yields the most conservative estimate for λ.

VI. Chance-Constrained Trajectory Generation in NeRFs

We now consider the problem of real-time trajectory planning for a robot in an environment represented as a NeRF, subject to constraints on the probability of collision (13). Our proposed algorithm, CATNIPS, has two parts: we first generate a lightweight, voxel-based scene representation, which we term a Probabilistically Unsafe Robot Region (PURR), that encodes the robot locations that satisfy the collision constraint for a particular robot geometry. We then use Bézier curves to plan safe trajectories in position space for a robot traversing the PURR subject to the probabilistic collision constraint. By assuming differential flatness of our robot, we can guarantee dynamic feasibility of our solution when planning in the flat output space.

We note that the following collision calculations may be conservative due to the approximation of the robot as a sphere such that safety is invariant to orientation. However, the collision metric (13) could be calculated while taking into account the position, orientation, and the physical geometry of the robot. We also note that the differentiability of the collision probability permits its use in gradient-based trajectory optimizers. Nonetheless, our design choices were motivated by speed, scalability, and modularity with other downstream tasks (e.g. active view planning where the orientation can be freely manipulated).

A. Generating the PURR

The PURR is a binary, voxelized representation of the NeRF that indicates collision in $\mathbb{R}^3$. If the robot's position is located within a free voxel in the PURR, the robot is probabilistically safe, i.e., the chance collision constraint is guaranteed to be satisfied. Otherwise, safety is not guaranteed—the robot may (or may not) be in violation of the chance constraint.

Similar to classical configuration space planning [42], the core idea behind the PURR is to inflate the occupied space in the map such that planning a path through free space in the inflated map corresponds to a robot trajectory that satisfies the collision probability constraint in the underlying NeRF map. We essentially "inflate" the NeRF density function to account for both the robot geometry and the chance constraint.

Fig. 4 shows a schematic of the PURR generation process, as well as how the PURR fits into our trajectory planning pipeline. We first voxelize the space of the map and label each cell with the integral of the NeRF density multiplied with a

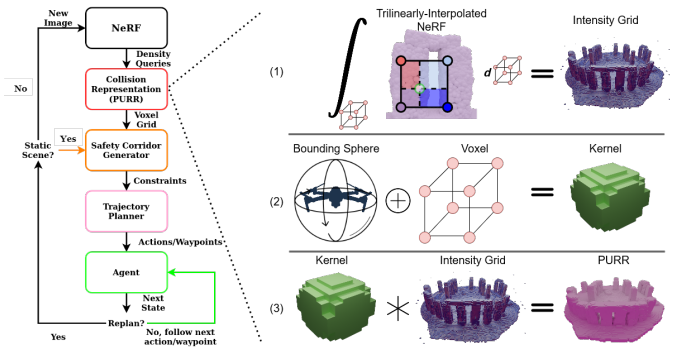

Fig. 4: NeRF to PURR pipeline. (1) A density grid is sampled from the NeRF, which is then trilinearly interpolated and integrated over a particular voxel to retrieve the cell intensity grid. (2) A robot kernel is generated by taking the Minkowski sum between the minimal bounding sphere of the robot and a single voxel. (3) The kernel is used in a `Conv3D` operation with the cell intensity voxel grid to create the robot intensity grid, which we then threshold by the user-defined collision probability σ to create the PURR.

scaling factor over each cell to give the voxelized *cell* intensity grid, $\mathbb{I}_c$. We then take the Minkowski sum between a sphere bounding the robot and one underlying voxel cell to produce the set of all voxels that the robot could be touching, in any orientation, if its center of mass were located anywhere in one cell; we call this the robot kernel, $\mathbb{K}$. Finally, we convolve the robot kernel $\mathbb{K}$ with the cell intensity grid $\mathbb{I}_c$ to obtain the *robot* intensity grid, $\mathbb{I}_r$. Finally, we evaluate the Poisson CDF using the robot intensity (as well as the auxiliary particle parameters) and threshold the robot intensity grid with the user-defined collision probability threshold σ to get the binary PURR map. These operations are described in more mathematical detail as follows.

1) Cell Intensity Grid: The *cell* intensity grid $\mathcal{I}_c$ computes, for each grid cell, the expected number of auxiliary particles in voxel v_{ijk}

$$\mathbb{I}_c(v_{ijk}) = \int_{v_{ijk}} \frac{\gamma \rho(x)}{A_{aux}} dx. \tag{16}$$

This integral, in general, cannot be computed analytically since the density ρ is typically represented using a neural network. We compute a high-quality approximation of this integral using a trilinear interpolation scheme. In fact, if the NeRF density uses an underlying voxel-based representation (as do the most high-speed and high quality NeRF variants in the literature [7], [28]) our integral of the trilinear interpolation is exact.

We first discretize the environment spatially, using a rectangular grid and query the NeRF for the density values at the grid vertices. We then represent the continuous density field using trilinear interpolation as [39],

$$\begin{aligned}\hat{\rho}(x,y,z) =& c_1 + c_2 x + c_3 y + c_4 z + \\ & c_5 xy + c_6 yz + c_7 xz + c_8 xyz.\end{aligned} \tag{17}$$

The coefficients $\mathbf{c}_{1:8}$ are the solution of a linear system $\mathbf{A}_{1:8}\mathbf{c}_{1:8} = \rho_{1:8}$, where $\mathbf{A}_{1:8}$ is the matrix of stacked row

vectors of the terms involving cell vertex locations, and $\rho_{1:8}$ the densities at the vertices. Note that at the vertices of the cell, $\hat{\rho}(x, y, z) = \rho(x, y, z)$. Different interpolations exist for other finite element geometries, although we only consider rectilinear cells in this work. The cell values of the *cell* intensity grid are computed from a closed form analytic solution to the integral (16) plugging in (17) for ρ over the extent of the cell. The analytic expression is given in Appendix A.

2) Robot Kernel: The robot kernel $\mathbb{K}(v_{ijk})$ is a mask that indicates the neighborhood of cells around voxel v_{ijk} that are considered in the computation of the collision probability when the robot position $\mathbf{p}$ is anywhere in v_{ijk}. We first find the Minkowski sum of the minimum bounding sphere[4] of the robot with the cell in which the robot center is located. We then compute the smallest collection of voxels that contains this Minkowski sum. This collection of voxels is the robot kernel, which can be efficiently convolved with the 3D voxelized grid using standard PyTorch functions.

3) Robot Intensity Grid: Once the robot kernel is defined, computing the collision probability for the robot in a particular cell simply requires a convolution between the kernel $\mathbb{K}$ and the cell intensity grid $\mathbb{I}_c$. In particular, we generate a *robot* intensity grid $\mathbb{I}_r$, by convolving the kernel with each cell, giving the expected number of auxiliary particles in the body as follows,

$$\begin{aligned}\mathbb{I}_r(v_{ijk}) &= \sum_{v_{lmn}\in\mathbb{K}(v_{ijk})} \mathbb{I}_c(v_{lmn}) \\ &= \sum_{v_{lmn}\in\mathbb{K}(v_{ijk})} \int_{v_{lmn}} \frac{\gamma\hat{\rho}(\mathbf{x})}{A_{aux}}\, d\mathbf{x} \\ &\geq \int_{B(\mathbf{p},\mathbf{R})} \frac{\gamma\hat{\rho}(\mathbf{x})}{A_{aux}}\, d\mathbf{x} \;\forall\, \mathbf{R}\in\mathbb{SO}(3), \mathbf{p}\in v_{ijk} \\ &\approx \int_{B(\mathbf{p},\mathbf{R})} \frac{\gamma\rho(\mathbf{x})}{A_{aux}}\, d\mathbf{x} \;\forall\, \mathbf{R}\in\mathbb{SO}(3), \mathbf{p}\in v_{ijk}.\end{aligned} \tag{18}$$

Finally, the PURR $\mathbb{P}$ is generated by calculating the CDF (13) using the *robot* intensity grid and thresholding by the collision probability threshold σ,

$$\mathbb{P}(v_{ijk}) = Pr(X \leq N_{aux}^{max}; \mathbb{I}_r(v_{ijk})) < \sigma. \tag{19}$$

The resulting PURR is visualized at different specified inter-penetration volumes at a reasonable probability of $\sigma = 95\%$ in Fig. 5 (Top) for the Flightroom NeRF environment. On the bottom of Fig. 5, a simple density thresholded grid can yield visually similar voxel maps, but can degenerate arbitrarily quickly for large density cutoffs. This is because the threshold for the PURR is calibrated to a precise probability of collision, while thresholding on the density provides no interpretable safety metrics.

Finally, we note that the trilinear interpolation of the density (17) to compute the integral in (16) introduces a potential source of approximation error. In practice, this error is much smaller than the over-approximation built into the various voxelization steps, yielding a PURR with a conservative probability of collision. However, one can remove any doubt about the conservatism of the approach by introducing an upper bound on the trilinear approximation error into the formulation. We call this approximation error bound the *collision offset factor*, α.

Definition 3 (Collision Offset Factor)**.** *The collision offset factor α is a map-wide upper bound on the difference between (i) the maximum collision probability achieved by integrating the NeRF density ρ over the robot kernel (i.e. the "true" collision probability) and (ii) the collision probability using the trilinearly-interpolated density $\hat{\rho}$ from (17),*

$$\alpha \leq \min_{i,j,k}\Big\{ \min_{\mathbf{p}\in v_{ijk},\mathbf{R}\in\mathbb{SO}(3)} Pr(X \leq N_{aux}^{max}; \int_{B(\mathbf{p},\mathbf{R})} \frac{\gamma\rho(\mathbf{x})}{A_{aux}}\, d\mathbf{x}) - \mathbb{P}(v_{ijk})\Big\}.$$

Finally, if α exists, then (19) and consequently our PURR are inflated with this collision offset factor to give the PURR a rigorous collision probability guarantee

$$\mathbb{P}(v_{ijk}) = Pr(X \leq N_{aux}^{max}; \mathbb{I}_r(v_{ijk})) < \sigma - \alpha. \tag{20}$$

Theorem 1. *Given a collision offset α and the desired collision probability threshold σ, a robot with position $\mathbf{p}$ in the complement of $\mathbb{P}$ is guaranteed to be probabilistically safe.*

Proof. Following the expressions in (18), the '$\approx$' in the last line becomes '$\geq$' for a collision offset factor, α, that satisfies Definition 3. Therefore, the resulting PURR is an upper bound on the probabilistic collision constraint of (13). □

Remark 1. *If the discretizations match between the PURR and a voxel-based NeRF architecture, as in [28] then α is*

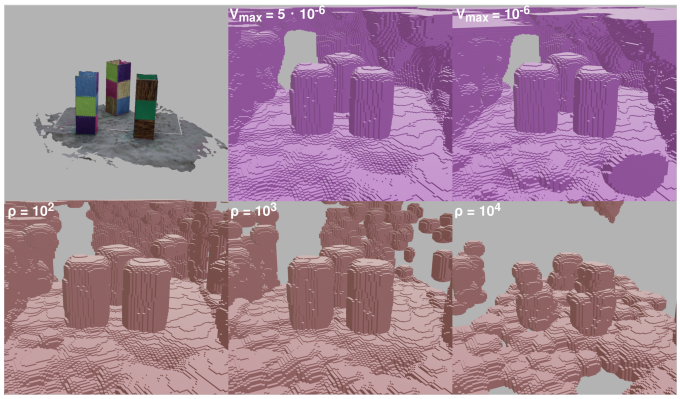

Fig. 5: Top: Mesh of Flightroom, and PURR with varying specified inter-penetration values V_{max} at fixed probability $\sigma = 95\%$. Bottom: Density-thresholded voxel maps with varying density values ρ. The PURR is precisely calibrated to give a desired probability of collision, while thresholding the NeRF density directly offers no particular safety guarantee. The density map can quickly degenerate based on the threshold, while the PURR can still capture the geometry for reasonable ranges of V_{max}.

[4]Inflating the robot body to a sphere removes the effects of robot orientation on safety. As a result, the PURR can be efficiently created in position space, while downstream tasks have the ability control the robot orientation without impacting safety.

identically 0. In practice, we still set α to 0 regardless of the NeRF architecture, and we find that the PURR free space is safe invariant to the size of the cells used in trilinear interpolation (Fig. 9). We conclude that $\alpha \approx 0$ (i.e. trilinear interpolation closely approximates the neural network) or $\alpha \geq 0$ (i.e. trilinear interpolation is over-approximating the network).

Remark 2. *Although voxel representations are undesirable in memory compared to compact neural networks, we note that the PURR is binary. The memory footprint can be further reduced by using octrees and by storing the PURR with a compression scheme, e.g, with hashing.*

B. Path Planning in the PURR

We now turn to the problem of chance-constrained trajectory optimization through the PURR. In particular, we seek to plan a dynamically feasible path for a robot through the environment such that all points along the trajectory satisfy a chance constraint on collision (rather than enforcing the chance constraints at discrete "knot points" along the trajectory). In particular, we seek to find trajectories that are *probabilistically safe*.

Definition 4 (Probabilistically Safe Trajectory)**.** *A trajectory is probabilistically safe if all points in the trajectory are pointwise probabilistically safe.*

We propose an algorithm containing three components used to create these safe, continuous trajectories. The first step is to find an initial, discrete path through the free space of the PURR by solving a constrained shortest path problem (Fig. 6a) from our initial position $\mathbf{p}_0$ to our final position $\mathbf{p}_f$ (e.g., using A*). While dynamically infeasible, this path provides a connected, collision-free path through the PURR that we will refine into a smooth, feasible trajectory. The second step is to create a "tube" of bounding boxes around this initial path that is not in collision with the PURR (Fig. 6b). We opt for this design choice because our emphasis in this paper is in quantifying collision risk in the NeRF in combination with a relatively simple planning scheme. One can introduce more flexible and sophisticated planning schemes [43], [44] to improve on our method. Finally, we generate a smooth curve connecting our initial and final positions by solving a constrained convex optimization, requiring the curve to lie in the free "tube" generated previously (Fig. 6c). We call the resulting trajectory planning algorithm Collision Avoidance Through Neural Implicit Probabilisitc Scenes (CATNIPS). CATNIPS executes in real-time given a pre-computed PURR map.

1) A* Search: We first find a rough, discrete initial path through the NeRF using an A* search over the voxel grid defining the PURR free space. Specifically, given an initial position $\mathbf{p}_0$ and final position $\mathbf{p}_f$ of the robot, we find a minimum-length (measured in the Manhattan distance) path between the corresponding initial and final voxels. We search a 6-connected graph, i.e., the robot can move into neighboring free voxels of the PURR along the x-, y-, and z-axes. Using A* with the usual heuristic (distance to goal, not considering collision) yields a connected, collision-free, but dynamically infeasible path from the start to the goal.

2) Bounding Box Generation: We now seek to refine the discrete, collision-free path returned by A* into a continuous trajectory that is energy-efficient, and dynamically feasible, for our robot. To this end, we first generate a "tube" around our A* path that is both large (so we minimally constrain our trajectory optimization) and lies in the free space of the PURR (so trajectories in this tube still remain collision-free). We represent this tube as a union of bounding boxes, as shown in Fig. 6b.

To generate these bounding boxes, we first split the A* trajectory into straight-line segments (i.e., if the path returned by A* begins by moving along the z-axis for 6 voxels, we join these into a single line segment between the start and endpoint of this sequence). We then expand a bounding box around each line segment by "marching" each face along its normal direction until it is marginally in collision with the PURR (i.e., at least one cell on the face borders an unsafe cell). To speed collision-checking for the prospective boxes, we convert the PURR to a KD-tree representation for this step.

Once this process is complete, we now have one bounding box for each line segment in our original A* path; the union of these bounding boxes both lies in the free space of the PURR, and contains at least one connected, collision-free path between our initial and final positions. However, for voxel grids with fine spatial resolution, the number of bounding boxes will grow, adding to computational complexity; thus, as a final step we eliminate all "similar" bounding boxes (i.e., any bounding box whose volume has sufficient overlap with a bounding box earlier in the sequence) to generate a simplified representation. We find that these simple axis-aligned rectangular bounding boxes are more well-behaved in dense voxel grids where narrow corridors exist, while general polytopic alternatives like [43] introduce numerical instability into the planner due to acute corners in the safe corridors (demonstrated by [44]).

3) Smooth Trajectory Generation via Bézier Curves: The final step of our planner is to generate a smooth trajectory that lies entirely in the union of the bounding boxes. To do this, we represent our trajectory as a connected series of Bézier curves. In particular, a Bézier curve in $\mathbb{R}^n$, of order N, is given by

$$\mathbf{p}(t; \mathbf{s}_k) = \sum_{k=0}^{N} \binom{N}{k} (1-t)^{N-k} t^k \mathbf{s}_k, \tag{21}$$

where $\mathbf{s}_k \in \mathbb{R}^n$ are a set of "control points" defining the geometry of the curve, and the curve is traced by a free parameter $t \in [0, 1]$. For any $t \in [0, 1]$, the Bézier curve $\mathbf{p}(t)$ is simply an interpolation of the control points $\mathbf{s}_k$, which means the parametric curve lies in the convex hull of the control points [45]. Thus, to generate a probabilistically safe trajectory through the PURR, we find a set of Bézier curves connecting our initial position $\mathbf{p}_0$ and final position $\mathbf{p}_f$, whose control points lie in the bounding boxes generated previously; since each Bézier curve must lie in the convex hull of its control points, the entire curve will lie in the complement of the PURR. We note that this is a common method for enforcing safety constraints in the path planning literature [46].

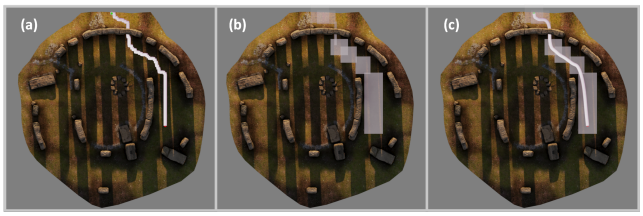

Fig. 6: (a) Discrete path returned by A*, (b) Union of bounding boxes containing subsets of the A* path whilst remaining strictly in the complement of the PURR, (c) Smooth path represented as the union of Bézier curves whose control points lie within a particular bounding box.

To find this trajectory, suppose we have L bounding boxes $\{\mathbb{B}_1, \ldots, \mathbb{B}_L\}$ generated from the previous step. We then find a set of L Bézier curves with control points given by $\mathbf{s}_k^i$, constraining all control points of the i^{th} curve to lie in the corresponding bounding box $\mathbb{B}_i$. Since the Bézier curves are linear functions of the control points, we can in turn represent the i^{th} curve as $\mathbf{p}_i(t) \equiv \beta(t)\mathbf{s}^i$, where $\mathbf{s}^i \in \mathbb{R}^{n(N+1)}$ is a concatenated vector of all $N+1$ control points for curve i, and $\beta(t) : [0,1] \mapsto \mathbb{R}^{n \times n(N+1)}$ is a coefficient matrix that only depends on the curve parameter t. We can similarly represent higher derivatives of the curve as $\mathbf{p}_i^{(d)}(t) = \beta^{(d)}(t)\mathbf{s}^i$. Since we are only concerned with positions and their derivatives, $n = 3$. We refer the reader to [45] for a more detailed treatment of Bézier curves and splines.

To help smooth the spline and discourage looping behavior, we introduce the objective

$$J(\mathbf{s}^1, \ldots \mathbf{s}^L) = \sum_{i=1}^{L} \left(\int_0^1 ||\beta^{(d)}(t)\mathbf{s}^i||_2^2 dt + \sum_{k=0}^{N-1} ||\mathbf{s}_k^i - \mathbf{s}_{k+1}^i||_2^2 \right),$$

which is quadratic in our decision variables $\mathbf{s}^i$. A typical choice is to penalize the snap of the trajectory ($d = 4$) as a proxy for control effort [47].

To generate our desired trajectory, we then solve the following optimization:

$$\begin{aligned} \min_{\mathbf{s}^1,\ldots,\mathbf{s}^L} \quad & J(\mathbf{s}^1, \ldots, \mathbf{s}^L) && (22) \\ \text{s.t.} \quad & \mathbf{s_j^i} \in \mathbb{B}^i, && \forall i \leq L,\ j \leq N \\ & \beta^{(d)}(1)\mathbf{s}^i = \beta^{(d)}(0)\mathbf{s}^{i+1}, && \forall i \leq L,\ d \leq D \\ & \beta(0)\mathbf{s}^1 = \mathbf{p}_0, \\ & \beta(1)\mathbf{s}^L = \mathbf{p}_f. \end{aligned}$$

In particular, we constrain the control points of every segment so that $\mathbf{s}^i$ must lie in the corresponding bounding box $\mathbb{B}_i$, which defines a set of linear inequaities in $\mathbf{s}^i$. We also enforce continuity of each spline up to a desired derivative D, which defines a set of linear equality constraints. Finally, we enforce the boundary conditions of our trajectory, i.e., that the curve begins at our initial position $\mathbf{p}_0$ and ends at our final position $\mathbf{p}_f$. We choose to optimize Bézier curves of order $N = 8$, to balance the expressiveness of our model (which needs non-trivial derivatives up to order $d = 4$) with the number of parameters needed to specify the curve.

Since our objective is quadratic in the control points, and our constraints are defined by linear inequalities and equality constraints, the resulting optimization (22) is a quadratic program (QP) that can be solved in real time.

Corollary 2. *The trajectory given by the solution of the QP* (22) *is probabilistically safe.*

Proof. The QP (22) constrains each Bézier curve to live within a bounding box that is probabilistically safe, rendering each curve safe. The resulting trajectory, given by the union of the Bézier curves is therefore probabilistically safe. □

Remark 3. *We emphasize that all trajectories are probabilistically safe in that all points in any trajectory satisfies the inter-penetration constraint (13) with probability σ. This is not equivalent to the statement that some σ fraction of all trajectories do not contain any points that violate the inter-penetration constraint.*

Remark 4. *If the robot system dynamics is differentially flat such that its position is a subset of the flat outputs, then the paths generated by the proposed QP* (22) *(with D set to the highest derivative of position in the flat outputs) are dynamically feasible. Therefore a robot tracking a trajectory from this planner remains probabilistically safe.*

Remark 5. *We note that each segment of the trajectory returned by our planner is parameterized by a simple curve parameter t, which need not correspond to time. However, since we assume our system is differentiably flat, there exists a time scaling such that the curve is dynamically feasible. To resolve this, we use a simple time rescaling (as in [47]) to generate the final trajectory as a function of time.*

VII. Numerical Results

In this section, we study our proposed chance-constrained trajectory optimizer on the simulated Stonehenge scene and real Statues and Flightroom environments. The real scenes were captured using a hand-held phone camera with poses extracted from COLMAP. Using our proposed trajectory optimizer, we generate trajectories for a simulated and real quadrotor flying through the scene, and study the safety and conservativeness of trajectories generated across a large number of initial and final conditions. Using the same path planning algorithm, we perform a comparison between the PURR and a baseline voxel occupancy representation obtained by thresholding the raw NeRF density at a desired density level. We also compare these methods to the authors' previous work NeRF-Nav [12]. Because this method requires an A* initialization, we use the same A* initialization for both the baseline grid and NeRF-Nav. Specifically, the NeRF-Nav A* initialization is generated from the baseline density grid corresponding to the cutoff $\rho = 10^2$ (the most conservative cutoff).

We demonstrate that both voxel methods are more computationally efficient than NeRF-Nav, and also generate trajectories that are safer (with fewer collisions) and less conservative (shorter paths). Further, we find that our method, CATNIPS, allows the user to set a clearly defined probability threshold

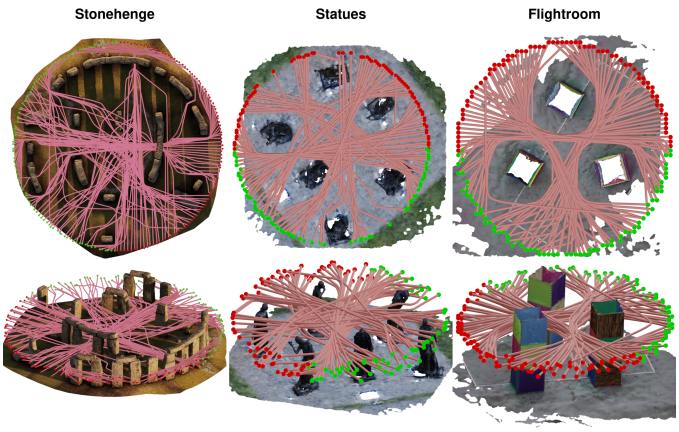

Fig. 7: Generated safe paths across 100 configurations from Stonehenge, Statues, and Flightroom, visualized from the top and sides. Both Statues and Flightroom NeRFs were trained from images of the real environments.

for collision. In contrast, the density threshold baseline does not give such a probabilistic guarantee. In other words, similar behavior can be obtained from a density thresholded map, but this requires a user to tune the density threshold through trial and error to reach a desired qualitative level of safety, and thresholds that work for one environment may not generalize to others. Even after tuning to get good empirical behavior, the baseline offers no accompanying safety guarantee.

A. Algorithm Performance

Qualitative results of the proposed method for 100 start and goal locations on a circle (perturbed randomly in the up-down direction) for all 3 environments are shown in Figure 7. The PURR was generated using a resolution of 150 voxels per side, with probability threshold $\sigma = 0.95$, and $V_{max} = 10^{-6}$ specified volume penetration [5]. Moreover, because our quadrotor system is differentially flat with flat outputs in position, the robot can follow these paths with a standard differential flatness based control pipeline [47].

We first compare our method to two baselines: the "baseline grid" which computes occupancy from a density threshold, and NeRF-Nav [12], a gradient-based trajectory planner for NeRFs in Fig. 8. We analyze their performance on 3 metrics: the minimum signed distance achieved during the trajectory to the ground-truth mesh (negative is within the mesh), the maximum inter-penetration volume achieved during the trajectory, and the difference in lengths between the generated trajectory and the shortest straight line path. The first two metrics quantify safety, while the last quantifies conservativeness. We evaluate the algorithms on the 100 randomized configurations distributed evenly on a circle.

We choose to display the same 6 parameter combinations for CATNIPS, varying the specific inter-penetration and probability cutoff, over all scenes to demonstrate generalizability and interpretability. The cutoffs we choose to be reasonable thresholds of 95% and 99%. The inter-penetration we choose to be some fraction of the robot body. For Stonehenge, $5{\cdot}10^{-6}, 10^{-6}, 10^{-7}$ correspond to $13\%, 3\%, .3\%$, while for the real environments, these values correspond to $4\%, .8\%, .08\%$, respectively.

To benchmark against the baselines, we vary the density cutoffs of the baseline grid and the collision penalty weight in NeRF-Nav. We again stress the lack of interpretable parameters for both NeRF-Nav and the baseline grid paths and their required parameter tuning to get a desired safety performance, which is impossible to know a priori as one does not have access to the ground-truth in reality. However, in order to benchmark the methods in good faith, we choose the density cutoffs on the baseline grid such that the performance on these metrics were similar to those of CATNIPS in a synthetic environment (i.e. Stonehenge), since we have access to its ground-truth mesh. We then use the same cutoffs for the real environments. In our experience, these are also thresholds that are typically used to extract meshes from NeRFs using marching cubes [13]. For NeRF-Nav, the collision penalty weights are chosen so that they are the dominant term in the loss.

We can see that NeRF-Nav trajectories are unsafe when compared to paths generated from either voxel method (negative is in collision with the mesh) over all collision loss weights $(10^2, 10^3, 10^4)$. As we increase the weighting on the collision penalty, we do see that the algorithm can be increasingly safe on average (higher SDF, lower volume intersection). However, such a high collision penalty (10^4) will typically cause numerical issues in the trajectory optimizer. Moreover, these trajectories in the worst case are simply less safe than either voxel method. Finally, these trajectories also deviate from the shortest path the farthest, illustrating conservatism and non-smooth paths.

The trajectories derived from the baseline grid can exhibit safe and non-conservative behavior. However, it is clear that the parameters necessary to achieve this behavior cannot be generalized over all scenes. This is evident for the cutoff $\rho = 10^4$, where safety performance in Stonehenge is reasonable, but we see unsafe performance in the real environments.

We see that our method, CATNIPS, is safe (by construction) and non-conservative. On average, these trajectories are similar in conservativeness compared to the baseline grid paths, while exhibiting reasonable levels of safety and respond as expected to changes in parameters (greater SDF and lower intersection when decreasing specified volume intersection and/or increasing probability threshold). We draw the readers attention to the volume intersection metric (Fig. 8, middle row) for CATNIPS, where trajectory-wise safety is expressed (please see Remark 3 for the distinction). The arrowhead represents the σ quantile over 100 trajectories, while the dotted lines represent the specified volume intersection. While we make no claims on full-trajectory rates of safety (i.e. the arrowhead need not be below the corresponding dotted line), we see that, indeed, a σ proportion of the trajectories tends to be completely safe (with no unsafe points existing on the whole trajectory).

[5]Because the cameras are mapped to be within a unit box, all reported length scales are assumed to be in this normalized system unless explicitly stated.

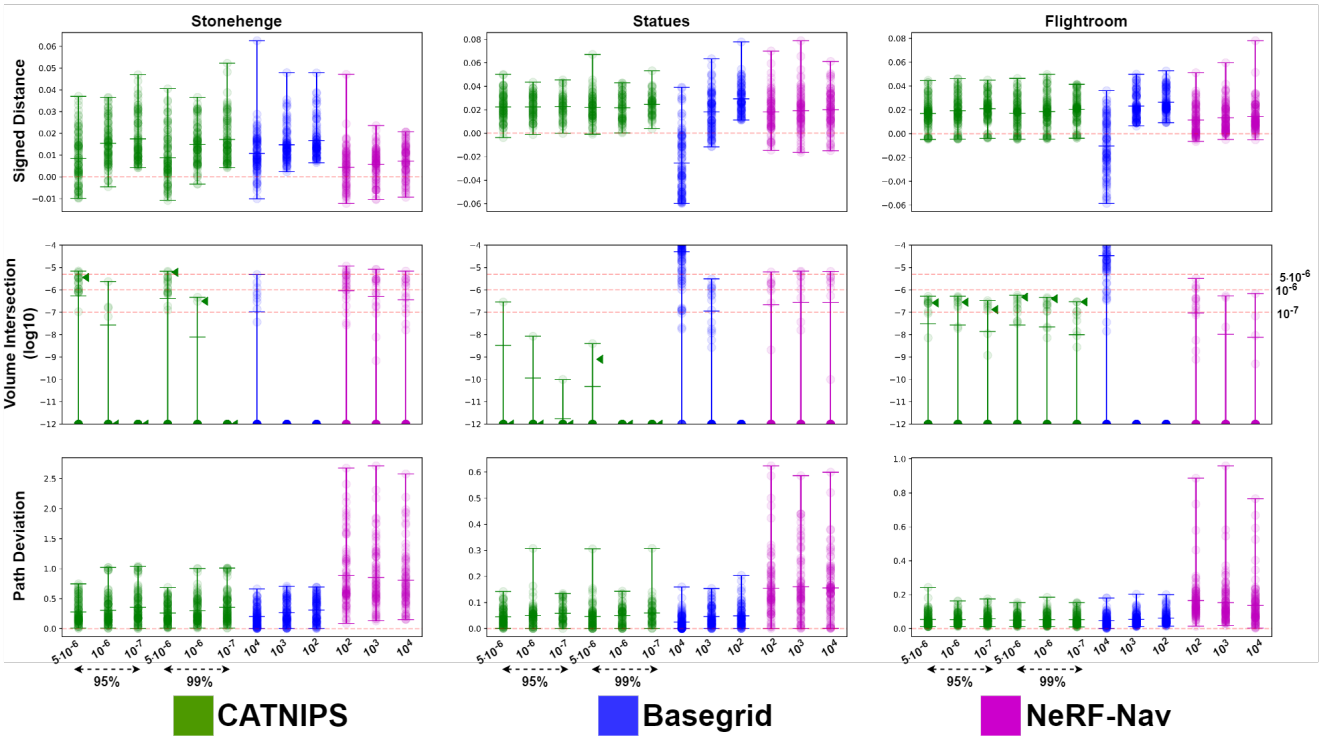

Fig. 8: Statistics on distances to obstacles, inter-penetration, and path conservativeness over 100 trajectories for each environment (Stonehenge, Statues, Flightroom). We benchmark our PPP method (three different inter-penetration distances, each at at 95% and 99% probability) against the paths using the baseline grid and NeRF-Nav [12]. Whiskers indicate max/mean/min over all trajectories, and the density of color represents the spread. Top: The minimum distance to the ground-truth mesh for every trajectory. Our method uses interpretable parameters, such that we can tune for safety (lower volume intersection) without being overly conservative (lower path deviation). Meanwhile, the effect of the density cutoff in the baseline grid is unpredictable across scenes, and the NeRF-Nav paths can lead to collision violations (low whiskers) and overly-conservative paths. Mid: The maximum inter-penetration volume per trajectory. Although we only make claims about point-wise safety, we see that trajectory-wise safety is approximately satisfied (i.e. arrowheads representing 95, 99% of trajectories tend to be below the specified inter-penetration indicated by red dotted lines). Bottom: Difference between the minimum length of a straight line path and the executed path. We see that both CATNIPS and the baseline grid are less conservative than NeRF-Nav.

We validate the point-wise safety claims we make, as well as ablate CATNIPS over grid discretizations, in Fig. 9. The figure contains runs with parameter combinations of two different collision probabilities, three different specified volume intersections, three different grid discretizations (100, 150, 200), and three different environments. Note that some columns can contain multiple bars, representing different grid discretizations or environments while maintaining the same collision cutoff and volume intersection. Bar heights represent the fraction of all 100 trajectories that contain at least a point with volume intersection higher than what was specified for that combination of parameters. Numbers on top of these bars indicate the percentage of all points in all trajectories that fall below the specified volume intersection for the same parameter setting. Our theoretical claims pertain to the rate across all points on all trajectories (number on top of bars), yet we observe for our method the desired collision rate tends to hold across full trajectories as well (the height of the bars). Combinations not visualized mean there were no points in any trajectories that violated the volume intersection constraint.

Note that the reported percentage of all points being safe (all percentages greater than the probability cutoff) means that our derived point-wise probabilistic safety constraint is validated and that satisfaction of this constraint is invariant to the parameters. This makes (13), the PURR, and the planning architecture surrounding it generalizable to arbitrary environments and reasonable grid discretizations. This result also implies that the error introduced through trilinear interpolation of neural network-based density fields is small in terms of its impact on safety (i.e. collision offset factor $\alpha \geq 0$). This is especially attractive for real scenes where there is no way to validate safety a priori, and for applications where coarser grid discretizations are necessary for computational performance. Moreover, trajectory-wise safety (like in Fig. 8) is generally satisfied over grid discretizations and scenes.

Here we would like to summarize several subtle points regarding collision violation to the reader. The violation of the specified volume penetration at some points (Fig. 8, 9) is due to both the probabilistic nature of the collision constraint, and due to the fact that the NeRF does not exactly capture the ground truth surface. For the NeRF density to exactly represent the true surface, under our PPP interpretation, it

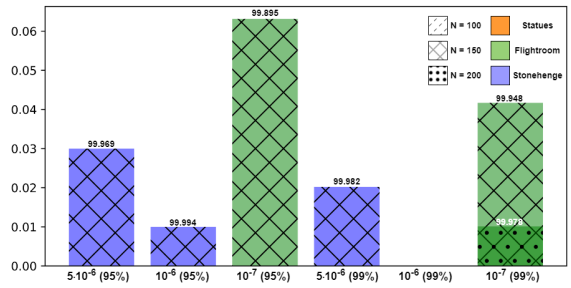

Fig. 9: Effect of grid discretizations and environments on safety. Bars represent fraction of trajectories out of 100 that contain at least one point greater than the specified inter-penetration volume. This ablation is performed over specified inter-penetration volumes, collision probabilities $\sigma = (95\%, 99\%)$, grid discretizations $N = (100, 150, 200)$, and environments (Stonehenge, Statues, Flightroom). The total percent of points across all trajectories below the specified inter-penetration volume for a specific combination of parameters and scenes is indicated above the bar. Combinations not shown means that all points in that setting were below the specified inter-penetration volume. We see that point-wise safety is always satisfied, and trajectory-wise safety is approximately valid.

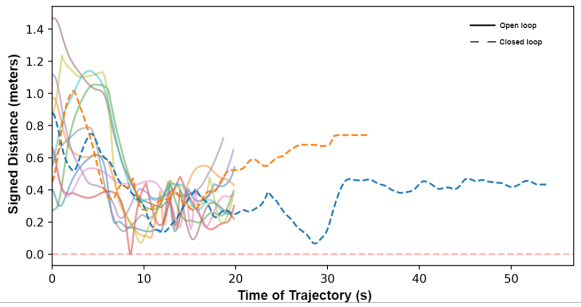

Fig. 10: Signed distances along the executed trajectories on a real drone in the Flightroom environment. There were 10 open-loop trajectories, and 2 additional ones with online replanning. All trajectories are above 0 and hence safe.

would have to be exactly 0 outside the surface, and ∞ inside. In this case, collision would be deterministic because the only way to satisfy the chance constraint on collision would be to have no collision. In practice, a NeRF density field cannot be either 0 or ∞ both because of continuity in the representation and embedded uncertainty about where the true surface is. Therefore, a collision with the true surface may occur with the prescribed probability. We believe that an accurate representation of uncertainty in perception-based planning must admit for some probability of collision, as there is always a nonzero probability that perception errors have led to an incorrect estimation of occupancy in the scene. We further alleviate this "NeRF-to-real" gap by embedding several conservative approximations into our method through the construction of the PURR. Therefore, in practice, we collide with the true surface less frequently than required by the collision constraint. Again, this fact is illustrated in Fig. 9, where the collision constraint satisfaction is very close to 100% but not conservative enough to be unappealing to use (Fig. 7, 8).

Finally, we validate our CATNIPS pipeline on drone hardware experiments in our Flightroom environment. Using a pre-trained static NeRF, we compute 10 trajectories from start-goal points distributed on the perimeter of a circle around the scene, and drive the on-board controller to follow these waypoints (open-loop). Then, we choose two of these start-goal locations and run the CATNIPS A*, bounding box generation, and convex program online (choosing the next predicted point as a waypoint) while simultaneously updating the drone to follow the stream of waypoints (closed-loop). All trajectories are run until they reach the goal location. The open-loop trajectories are about 20 seconds long since they are pre-defined, while the closed loop trajectories can have varying times since they are not pre-defined. We see that all trajectories have a signed distance greater than 0 and are therefore safe (no collision). Even for the open-loop trajectory that comes close to 0, we visually verify a collision-less trajectory. The near-zero SDF is due to over-approximating the drone body with a bounding box when computing the signed distance.

B. *Computation Times*

Computation Time (seconds)		
Operation	CATNIPS/Basegrid	NeRF-Nav
Offline		
Robot Kernel	0.002	0.002
PURR/Basegrid	1.11	1.11
Gradients	N/A	9.31*
Online		
A*	0.16 ± 0.05	0.16
Bounding Box	0.12 ± 0.09	N/A
B-Spline	0.034 ± 0.029	N/A
Gradients	N/A	0.93**

* 1000 gradient steps.
** 100 gradient steps.

TABLE I: Timing results (performed on a laptop with an RTX 4060 GPU) between our voxel methods (CATNIPS and baseline grid) and NeRF-Nav. Because both voxel methods use similar operations, they have identical times. Since NeRF-Nav depends on an A* initialization, it also inherits some computation from the voxel methods. Offline operations only need to be performed once for a static NeRF environment, while online operations need to be performed whenever replanning occurs.

The implementation of the above algorithms are performed in Pytorch on a laptop with an RTX 4060 GPU. Our method is built on top of NeRFStudio [8]. Little effort was made to optimize code for fast computation, so we expect these execution time could be substantially reduced. Moreover, for a fair comparison, we ported NeRF-Nav to NeRFStudio. In general, the planning portion of CATNIPS (A*, bounding box, and Bézier curve generation) operates at around 3 Hz. The

operation from querying the NeRF density to the creation of the PURR runs at 1 Hz. The break-down of each operation is shown in Table I. As a promising direction for future work, one can further reduce the computation time for creating the PURR and optimizing trajectories in the PURR through parallelization and code optimization on a GPU. [6].

Note that the proposed method produces smooth trajectories from the current position to the goal. Also note that in an online re-planning scenario, usually only the next waypoint is tracked before the entire trajectory is updated. Thus, certain parts of our method can be adapted to only consider the vicinity of the robot, trading computation time for suboptimality. In comparison, NeRF-Nav takes longer to converge (if at all) to a reasonable tolerance, without any safety guarantees. We believe this is primarily due to the difficulty of optimizing its highly nonconvex objective and the many queries required to the density neural network within the trajectory optimizer. In order to produce the best performance from NeRF-Nav in terms of safety, we ran the algorithm for 1000 gradient steps.

VIII. Conclusion

In this paper, we present a novel method for chance-constrained trajectory optimization through NeRF scenes. We present a method to transform the NeRF into a Poisson Point Process (PPP), which we use to generate rigorous collision probabilities for a robot body moving through the scene. Leveraging this expression for collision probability, we develop a fast method for online trajectory generation through NeRF scenes, which, offline, distills the NeRF density into a voxel-based representation of collision probability called the PURR. Using the PURR, we present an algorithm to plan trajectories represented as Bézier splines that guarantee a robot traversing the spline does not exceed a user-defined maximum collision probability. In numerical experiments, we show our proposed method generates safer and less conservative paths than a state-of-the-art method [12] for trajectory planning through NeRFs, and also gives more well-behaved and more user interpretable results than a baseline planner that uses a threshold on the NeRF density as a proxy for collision probability. We also demonstrate that our pipeline can run in real-time.

This work opens numerous directions for future research. Since our entire pipeline (both PURR generation and trajectory optimization) can run at real-time rates, our planner could be combined with a NeRF-based state estimator (e.g., [21], [19]) to perform active exploration or next-best-view planning on NeRFs, allowing a robot to autonomously explore a novel environment using only onboard vision. Building on navigation, another interesting direction is to tune the collision metric on-line during execution. Because the collision probability is differentiable with respect to the pose, it is possible to tune the collision probability online in response to data collected on-the-fly (e.g., sensed minimum distance to the nearest obstacle). This could be implemented with auto-differentiation as part of a PyTorch-based planning pipeline. The probabilistic collision framework developed here could have interesting applications to problems like differentiable simulation of rigid bodies represented as NeRFs [49] or planning for problems like contact-rich manipulation and locomotion. Finally, since our derivation a PPP from a NeRF is rigorous and generalizable, we hope that our interpretation of NeRFs will be useful to research beyond robotics, for example in computer vision and computer graphics.

[6]The A* library [48] we use allows users to pre-process a static voxel grid such that generating the A* initialization is a look-up operation that is near instant in exchange for a one-time processing cost of approximately a second.

Appendix A
Integration over Trilinearly Interpolated Cells

For a trilinearly interpolated density over a cell v_{ijk} given by (17) with local coordinates $(x, y, z) \in ([a_x, b_x], [a_y, b_y], [a_z, b_z])$, the volume integral over the cell can be computed analytically as:

$$\begin{aligned}\int_{a_x}^{b_x}\int_{a_y}^{b_y}\int_{a_z}^{b_z}\rho(x)dxdydz &= (b_x-a_x)(b_y-a_y)(b_z-a_z)\\ &+\frac{c_2}{2}(b_y-a_y)(b_z-a_z)(b_x^2-a_x^2)\\ &+\frac{c_3}{2}(b_x-a_x)(b_z-a_z)(b_y^2-a_y^2)\\ &+\frac{c_4}{2}(b_x-a_x)(b_y-a_y)(b_z^2-a_z^2)\\ &+\frac{c_5}{4}(b_z-a_z)(b_x^2-a_x^2)(b_y^2-a_y^2)\\ &+\frac{c_6}{4}(b_x-a_x)(b_y^2-a_y^2)(b_z^2-a_z^2)\\ &+\frac{c_7}{4}(b_y-a_y)(b_x^2-a_x^2)(b_z^2-a_z^2)\\ &+\frac{c_8}{8}(b_x^2-a_x^2)(b_y^2-a_y^2)(b_z^2-a_z^2)\end{aligned} \tag{23}$$

where c_i are the coefficients of the trilinear interpolation.

Acknowledgements

We would like to thank Keiko Nagami, Adam Caccavale, Gadi Camps, and Jun En Low for their insights throughout this project.

References

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *European Conference on Computer Vision (ECCV)*, 2020.

[2] E. Sucar, S. Liu, J. Ortiz, and A. Davison, "iMAP: Implicit mapping and positioning in real-time," in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021.

[3] 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.

[4] L. Yen-Chen, P. Florence, J. T. Barron, T.-Y. Lin, A. Rodriguez, and P. Isola, "NeRF-Supervision: Learning dense object descriptors from neural radiance fields," in *IEEE Conference on Robotics and Automation (ICRA)*, 2022.

[5] D. Driess, Z. Huang, Y. Li, R. Tedrake, and M. Toussaint, "Learning multi-object dynamics with compositional neural radiance fields," in *Conference on Robot Learning (CoRL)*, 2022.

[6] J. Ichnowski*, Y. Avigal*, J. Kerr, and K. Goldberg, "Dex-NeRF: Using a neural radiance field to grasp transparent objects," in *Conference on Robot Learning (CoRL)*, 2020.

[7] 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.

[8] 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.

[9] 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.

[10] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, Jun. 1989.

[11] S. Osher and R. P. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*. New York: Springer, 2003.

[12] 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.

[13] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 163–169, Aug. 1987.

[14] 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.

[15] 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.

[16] 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.

[17] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura, and W. Wang, "Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction," *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] X. Pan, Z. Lai, S. Song, and G. Huang, "Activenerf: Learning where to see with uncertainty estimation," in *European Conference on Computer Vision (ECCV)*. Springer, 2022, pp. 230–246.

[23] K. Lin and B. Yi, "Active view planning for radiance fields," in *Robotics Science and Systems*, 2022.

[24] N. Sünderhauf, J. Abou-Chakra, and D. Miller, "Density-aware nerf ensembles: Quantifying predictive uncertainty in neural radiance fields," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 9370–9376.

[25] J. Shen, A. Ruiz, A. Agudo, and F. Moreno-Noguer, "Stochastic neural radiance fields: Quantifying uncertainty in implicit 3d representations," in *International Conference on 3D Vision (3DV)*. IEEE, 2021, pp. 972–981.

[26] H. Zhan, J. Zheng, Y. Xu, I. Reid, and H. Rezatofighi, "ActiveRMAP: Radiance field for active mapping and planning," 2022. [Online]. Available: https://arxiv.org/abs/2211.12656

[27] L. Goli, C. Reading, S. Selllán, A. Jacobson, and A. Tagliasacchi, "Bayes' rays: Uncertainty quantification for neural radiance fields," *arXiv preprint arXiv:2309.03185*, 2023.

[28] Sara Fridovich-Keil and Alex Yu, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa, "Plenoxels: Radiance fields without neural networks," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[29] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa, "PlenOctrees for real-time rendering of neural radiance fields," in *International Conference on Computer Vision (ICCV)*, 2021.

[30] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan, "Mip-NeRF: A multiscale representation for anti-aliasing neural radiance fields," *International Conference on Computer Vision (ICCV)*, 2021.

[31] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, "Mip-NeRF360: Unbounded anti-aliased neural radiance fields," *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[32] N. E. Du Toit and J. W. Burdick, "Probabilistic collision checking with chance constraints," *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 809–815, 2011.

[33] L. Blackmore, M. Ono, and B. C. Williams, "Chance-constrained optimal path planning with obstacles," *IEEE Transactions on Robotics*, vol. 27, no. 6, pp. 1080–1094, 2011.

[34] H. Zhu and J. Alonso-Mora, "Chance-constrained collision avoidance for MAVs in dynamic environments," *IEEE Robotics and Automation Letters (RA-L)*, vol. 4, no. 2, pp. 776–783, 2019.

[35] B. Luders, M. Kothari, and J. How, "Chance constrained RRT for probabilistic robustness to environmental uncertainty," *AIAA Guidance, Navigation, and Control Conference*, Aug. 2010.

[36] J. L. Schonberger and J.-M. Frahm, "Structure-from-motion revisited," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[37] J. F. C. Kingman, *Poisson processes*, ser. Oxf. Stud. Probab. Oxford: Clarendon Press, 1993, vol. 3.

[38] S. N. Chiu, D. Stoyan, W. Kendall, and J. Mecke, "Point processes I — The Poisson point process," in *Stochastic Geometry and Its Applications*. John Wiley & Sons, Ltd, 2013, ch. 2, pp. 35–63.

[39] P. L. Williams and N. Max, "A volume density optical model," in *Proceedings of the 1992 Workshop on Volume Visualization*. Association for Computing Machinery (ACM), 1992, p. 61–68.

[40] F. Baccelli and J. O. Woo, "On the entropy and mutual information of point processes," in *IEEE International Symposium on Information Theory (ISIT)*, 2016, pp. 695–699.

[41] A. W. v. d. Vaart, *Asymptotic Statistics*, ser. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1998.

[42] T. Lozano-Perez, *Spatial planning: A configuration space approach*. Springer, 1990.

[43] 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.

[44] C. Toumieh and A. Lambert, "Voxel-grid based convex decomposition of 3D space for safe corridor generation," *Journal of Intelligent and Robotic Systems*, vol. 105, no. 4, 2022.

[45] K. I. Joy, "Bernstein polynomials." [Online]. Available: http://www.inf.ufsc.br/ aldo.vw/grafica/apostilas/Bernstein-Polynomials.pdf

[46] A. Gasparetto and V. Zanotto, "A new method for smooth trajectory planning of robot manipulators," *Mechanism and Machine Theory*, vol. 42, no. 4, pp. 455–471, Apr. 2007.

[47] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 2520–2525.

[48] "dijkstra3d," 2021, https://github.com/seung-lab/dijkstra3d.

[49] S. Le Cleac'h, H.-X. Yu, M. Guo, T. Howell, R. Gao, J. Wu, Z. Manchester, and M. Schwager, "Differentiable physics simulation of dynamics-augmented neural objects," *IEEE Robotics and Automation Letters (RA-L)*, vol. 8, no. 5, pp. 2780–2787, 2023.