CATNIPS: Collision Avoidance Through Neural Implicit Probabilistic Scenes

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

***Abstract*—We formalize a novel interpretation of Neural Radiance Fields (NeRFs) as giving rise to a Poisson Point Process (PPP). This PPP interpretation allows for rigorous quantification of uncertainty in NeRFs, in particular, for computing collision probabilities for a robot navigating through a NeRF environment model. 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 model, 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, showing superior performance compared with two other methods for trajectory planning in NeRF environment models.**

***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 potnetial benefits over traditional scene representations: they can be trained using only monocular RGB images, they provide a continuous representation of obstacle geometry, and they represent scene geometry accurately even when specular or transparent materials are present while sensors such as depth cameras and lidar often fail [4], [6]. 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.

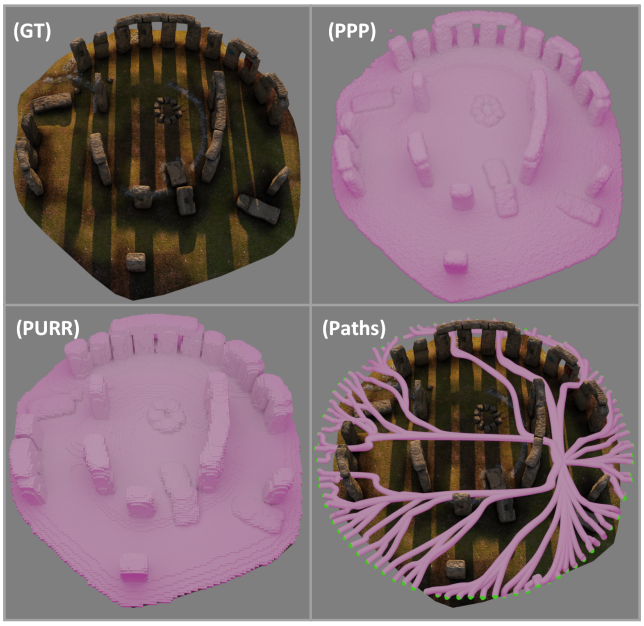

Fig. 1: (a) Ground-truth of the Stonehenge scene, (b) Poisson Point Process (PPP) of the scene represented as a point cloud, (c) Probabilistically Unsafe Robot Region (PURR) of scene, (d) Generated safe paths from our method.

In this work 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 interpretation of a NeRF as 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 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 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

the trajectories are collision free up to the user-specified probability threshold. Our method can compute multiple trajectory options from a cold start in about 1 sec, and can re-solve for single trajectories online at 50 Hz, while guaranteeing safety. This is 5 times faster at initial planning and 100 times faster at online re-planning than existing NeRF-based trajectory planners that provide no safety guarantee [12].

The key theoretical advance underpinning our results is the novel interpretation of the NeRF as 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 that these baselines, as collision is strictly 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 50Hz on a desktop computer, compared to the gradient descent-based planner proposed in [12], which requires approximately 4 sec 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, other uses of NeRFs in robotics, planning and control for robots using traditional scene representations, and chance-constrained planning.

A. 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] attempts to perform safe navigation of NeRF environments by penalizing the densities evaluated within the robot body modeled as a point cloud to discourage inter-NeRF penetration. [14] 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 [14] 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 traditional trajectory optimization tools to plan trajectories 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 .02 sec per online trajectory resolve on similar computing hardware.

B. 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, [4] uses dense object descriptors supervised with a NeRF model for robot gasping.

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 trajectory. The approach in [15] uses a grid-interpolation-decoder NeRF architecture in a similar SLAM pipeline. The work [16] 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, [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. [17] approaches a similar problem using a particle filter instead of a nonlinear optimization-based filter.

Other papers have considered active view planning for NeRFs. [18] treats the radiance value of the NeRF as Gaussian distributed random variables, and performs Bayesian filtering to find the next best view. [19] uses disagreement among an ensemble of NeRFs to choose the next best camera view. Similarly, [20] uses ensembles in a next best view strategy while also adding ray-termination densities to the information gain metric. [21] uses variational inference to train a probability distribution over NeRFs for next best view selection. [22] 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.

Of course, many of these works would not be applicable to robotics if they were not real-time. [7], [23], [24] have made massive performance gains to train NeRFs in real-time. Moreover, NeRFs must be able to capture reality as

well. They are known to suffer in quality when reconstructing rich, real environments with different length scales. [25], [26] attempts to fix this issue by extrapolating over the entire camera frustrum rather than a ray. Discussion on how their work relates to our results are in IV.

C. Planning and Control with Onboard Sensing

Planning and control onboard real hardware has already yielded a large amount of literature. Typically these works present reactive control schemes [27], using the sensed depth directly to perform collision checking in real-time. These methods typically are either 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 [28], [29], which in this work is encoded within voxels. [29] 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.

D. Chance-constrained Planning

There exists large literature on trajectory planning for robots that seeks to impose constraints on the probability of collision when the underlying scene geometry is unkown; this approach is known as chance-constrained planning. This approach models the robot state as stochastic, while the parameters defining the constraint (i.e., the map) are 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. [30] assumes Gaussian-distributed obstacle states, and approximates the collision probability as constant over the robot body (suitable only for small robots). [31] encodes 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. [32] incorporates this linear probabilistic constraint into RRT. [33] again uses 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. 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, $\rho : \mathbb{R}^3 \mapsto \mathbb{R}_{>0}$, maps a 3D location $\mathbf{p} = (x, y, z)$ to a positive density value ρ that encodes the differential probability of a light ray stopping at that point.[1] The radiance (i.e., RGB color) function $c : \mathbb{R}^3 \times \mathbb{R}^2 \mapsto \mathbb{R}^3$ maps a 3D location $\mathbf{p} = (x, y, z)$ and camera view direction $\mathbf{d} = (\theta, \phi)$ to an emitted RGB color c represented as a vector in $\mathbb{R}^3$. In this paper, we focuse 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 $C(\mathbf{o}, \mathbf{d}) \in [0, 1]^3$ 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

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

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.

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 $C(\mathbf{o}_i, \mathbf{d}_{ij})$ 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. 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 $\bar{C}_{ij}$. 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

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

where θ are the parameters of the neural networks representing the density and radiance fields ρ and c, which appear in the computation of the pixel color $C(\mathbf{o}_i, \mathbf{d}_{ij}; \theta)$ 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 [34]) 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).

[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 [23], or using a combination of interpolated voxel features and an MLP decoder [7], [24]. Our method can work with any of these representations.

IV. NeRF Density as a Poission Point Process

In this section, we show that a NeRF density field may be interpreted as a Poisson Point Process (PPP) and the NeRF color and density fields together represent a "marked" PPP [35], [36]. 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 are interpreted as a marked PPP.

This connection is significant since identifying the NeRF density field as a PPP 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 the interpretation of the NeRF density, and paves the way for practical utility in other domains beyond safety (e.g., in active sensing and active view selection). 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 the PPP interpretation.

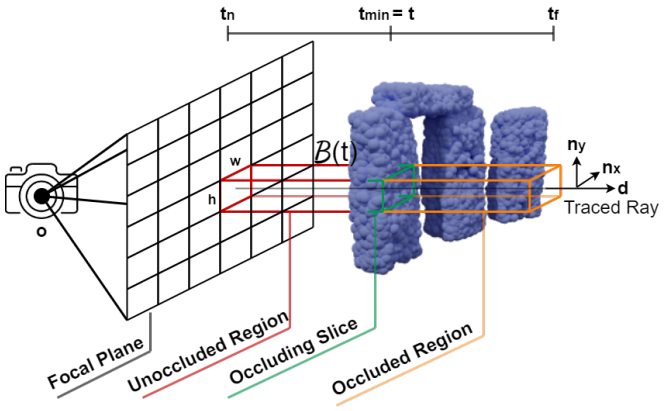

Fig. 2: In the rendering process, the probability that the pixel color takes on the color of the infintesimally small occluding slice (green) is given by the probability that nothing exists in the unoccluded region (red). Then, the pixel color is the expectation of the color taken by varying the position of the occluding slice along the ray.

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 [35], 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 $\lambda > 0$ 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 $\Lambda(B) = \int_{x \in B} \lambda(x) dx$.

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

This is sometimes refered to as the *inhomogeneous* PPP since the intensity λ is a function of the spatial variable x. If the intensity is constant over 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

$$Pr(N(B) = 0) = \exp\left[-\int_B \lambda(x) dx\right]. \tag{3}$$

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

Finally, we note that PPPs may be "marked" or "colored" with various quantities using a deterministic labeling function $c(x) : \mathbb{R}^n \mapsto \mathcal{C}$. 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 [35, Ch. 5] for a detailed discussion.

B. "Rendering" a Marked PPP

The intensity function of a PPP admits an important infinitesimal interpretation: $\lambda(x)dx$ is the probability that a point of the process lies within an infinitesimal volume dx centered at x. We note this is closely related to the interpretation of the density field offered by the original NeRF authors [1]: $\rho(x)$ defines the infinitesimal probability of a ray terminating at a given point $x \in \mathbb{R}^3$. In this section, we make this connection explicit by showing 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 a 3D measure of occupancy. To do this, we model "rendering" as beam tracing [37], which considers the occupancy of "thick beams" (i.e., the pixel area swept along a particular ray) rather than the occupancy of infinitesimal rays. In particular, when generating the color of a particular pixel from a marked PPP, we extend a rectangular

[2]The subsets B must be Lebesgue measureable.

prism along the pixel's ray (Fig. 2), and return the color of the first point of the process encountered along the ray.

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

$$\mathcal{B}(t) \equiv \left\{ \mathbf{o} + \mathbf{d}\tau + x\hat{\mathbf{n}}_x + y\hat{\mathbf{n}}_y \mid |x| \leq \frac{w}{2}, |y| \leq \frac{h}{2}, \tau \in [t_n, t] \right\}, \tag{4}$$

where $\hat{\mathbf{n}}_x, \hat{\mathbf{n}}_y$ are unit vectors tangent to $\mathbf{d}$ forming a basis in the image plane and h, w are the height and width of the pixel, respectively, measured on the image plane. We say the ray terminates wherever it first encounters a point of the process, i.e., at the first depth t where the set $\mathcal{B}(t)$ is nonempty.

C. Equivalence of NeRF and PPP Rendering

This brings us to our main result. Here we show that, under some assumptions about 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 $\mathbf{c}$.

Proposition 1 (NeRF as PPP)**.** *Consider a NeRF with density $\rho(\mathbf{x})$ and radiance $\mathbf{c}(\mathbf{o}, \mathbf{d})$. Let $C(\mathbf{o}, \mathbf{r})$ be the color of a pixel of width w and height h rendered from the NeRF* (1)*. Assume the NeRF density and radiance are constant over lengths smaller than $R = \sqrt{\frac{w^2+h^2}{2}}$, i.e., $\rho(\mathbf{x}+\mathbf{z}) = \rho(\mathbf{x}), \mathbf{c}(\mathbf{x}+\mathbf{z}, \mathbf{d}) = \mathbf{c}(\mathbf{x}, \mathbf{d})$ if $||\mathbf{z}|| \leq R$. Then, the pixel color $C(\mathbf{o}, \mathbf{r})$ of the NeRF is the expected color of a pixel rendered from a PPP with intensity $\lambda(\mathbf{x}) = \frac{\rho(\mathbf{x})}{wh}$ and marking $\mathbf{c}(\mathbf{x}, \mathbf{d})$.*

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 beam $\mathcal{B}(t)$ (4), the rectangular prism created by sweeping the pixel area along $\mathbf{r}$ from t_n to t. For this set, we can compute the intensity $\Lambda\big(\mathcal{B}(t)\big)$ by computing the integral

$$\begin{aligned}
\Lambda\big(\mathcal{B}(t)\big) &= \int_{\mathbf{x} \in \mathcal{B}(t)} \lambda(\mathbf{x}) d\mathbf{x} \\
&= \int_{-\frac{w}{2}}^{\frac{w}{2}} \int_{-\frac{h}{2}}^{\frac{h}{2}} \int_{t_n}^{t} \frac{\rho(\mathbf{r}(\tau) + x\hat{\mathbf{n}}_x + y\hat{\mathbf{n}}_y)}{wh} d\tau dy dx, \\
&= \int_{t_n}^{t} \rho(\mathbf{r}(\tau)) d\tau
\end{aligned}$$

where the last equality follows from the assumption that ρ is constant for $||x\hat{\mathbf{n}}_x + y\hat{\mathbf{n}}_y||^2 \leq \frac{w^2+h^2}{2}$.

Let us now consider a random variable T_{min} which defines the distance of the first (or closest) point of the Poisson process to lie along the ray (i.e. the location of the occluding slice denoted green in Fig. 2). We can define the cumulative distribution function of this variable by noting, for some $t > t_n$, $T_{\text{min}} \leq t$ if and only if $N\big(\mathcal{B}(t)\big) > 0$. Thus, the CDF of T_{min} can be defined using the void probability of $\mathcal{B}(t)$ (3),

$$\begin{aligned}
P(T_{\text{min}} \leq t) \equiv F_{T_{\text{min}}}(t) &= 1 - P\left(N\big(\mathcal{B}(t)\big) = 0\right), \\
&= 1 - \exp\left[-\Lambda\big(\mathcal{B}(t)\big)\right], \\
&= 1 - \exp\left[-\int_{t_n}^{t} \rho(\mathbf{r}(\tau)) d\tau\right].
\end{aligned}$$

□

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

$$f_{T_{\text{min}}}(t) = \frac{d}{dt} F_{T_{\text{min}}}(t) = \rho(\mathbf{r}(t)) \exp\left[-\int_{t_n}^{t} \rho(\mathbf{r}(\tau)) d\tau\right]. \tag{5}$$

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

Finally, we note that, since we assume the radiance of the NeRF is constant over the pixel area, the color of the point 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 PPP rendering by computing the expectation of $\mathbf{c}(\mathbf{r}(T_{\text{min}}))$, yielding

$$\begin{aligned}
C(\mathbf{r}) &= \mathbb{E}\left[\mathbf{c}(\mathbf{r}(T_{\text{min}}))\right], \\
&= \int_{t_n}^{t_f} \rho(\mathbf{r}(t))\ \mathbf{c}(\mathbf{r}(t))\ \Theta(t) dt,
\end{aligned}$$

where $\Theta(t) = \exp\left[-\int_{t_n}^{t} \rho(\mathbf{r}(\tau)) d\tau\right]$. This expected color matches exactly the expression given in (1) from the original NeRF paper [1], completing the proof.

We visually show the relationship between the PPP and the NeRF 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 when using

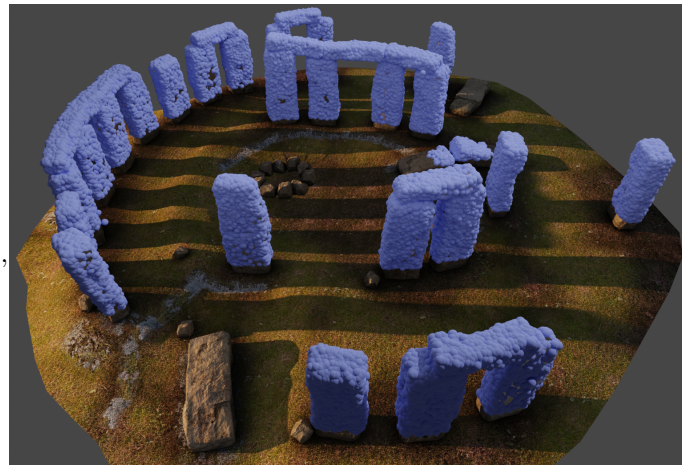

Fig. 3: Overlay of a realization of the PPP with the ground-truth mesh of Stonehenge. The two have strong spatial agreement.

a learning framework to learn ρ. In fact, we conjecture the observed aliasing phenomena in which NeRFs fail at different scales [25] is due to simplistic area-averaging of λ to retrieve ρ. The success of works like Mip-NeRF [25] 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 (5),

$$\log P(d_1, \dots d_k) = \sum_{i=0}^{k} \log \rho(\mathbf{r}(d_i)) - \int_{t_n}^{d_i} \rho(\mathbf{r}(t)) dt. \quad (6)$$

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 [38]. In particular, the entropy of a point process over a set B is defined as:

$$h(B) \equiv \int_B \lambda(\mathbf{x})(1 - \log \lambda(\mathbf{x})) d\mathbf{x} \quad (7)$$

$$= \frac{1}{wh} \int_B \rho(\mathbf{x})(1 + \log wh - \log \rho(\mathbf{x})) d\mathbf{x}. \quad (8)$$

Thus, 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 well-supervised for problems such as 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 [39, 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

In this section, we leverage the probabilistic interpretation of NeRFs to evaluate the probability of collision between a robot body and the NeRF, a NeRF and a NeRF, and provide extensions to state uncertainty in Appendix B.

A. Collision Probability with a Robot Body

Define $B(\mathbf{p}, R) \subset \mathbb{R}^n$ as the robot body parameterized by its pose $\mathcal{T} = (\mathbf{p}, R)$, (i.e., the set of points occupied by the robot with position $\mathbf{p} \in \mathbb{R}^3$ and orientation $R \in \mathbb{SO}(3)$) and consider an environment represented as a NeRF with density field $\rho(\mathbf{p})$, which we have shown to be equivalent to the PPP field through $\lambda(\mathbf{p}) = \frac{\rho(\mathbf{p})}{wh}$. The collision probability, or the probability that a point from a PPP intersects with the robot body, is equivalent to the probability that B is non-empty. This is the complement of the void probability (3) and is trivially extended to NeRF environments through the conversion from λ to ρ

$$Pr(N(B) > 0) = 1 - \exp\left[- \int_B \frac{\rho(x)}{wh} \, dx \right], \quad (9)$$

which naturally leads to our probabilisitc notion of safety.

Definition 2 (Probabilisitically Safe)**.** *A robot body parametrized by its pose $B(\mathbf{p}, R)$ is probabilistically safe if the collision probability given in (9) satisfies $Pr(N(B(\mathbf{p}, R))) \leq \sigma$, for some desired probability threshold σ.*

B. Collision Probability between Two NeRF Objects

Using the PPP formulation, we can extend collision not only between a NeRF and an arbitrary set, but also between two or more NeRF objects. For simplicity, we derive collision for two NeRF objects, which we define by their density fields ρ_1 and ρ_2. These fields are now parametrized by the pose of the objects $\rho_i(x; \mathbf{p}_i, R_i)$ due to rigid body motion. A point $x \in \mathbb{R}^3$ is in collision with both objects if $\rho_1(x) > 0$ and $\rho_2(x) > 0$. We then define the set

$$K(\mathbf{p}_1, R_1, \mathbf{p}_2, R_2) \equiv \{x | \rho_1(x) > 0 \text{ and } \rho_2(x) > 0\} \quad (10)$$

as the maximal collision set. In other words, this is the largest possible set where collisions can exclusively happen.

We restrict our analysis to be within K, as points outside K do not contribute to collision. We can then use the same argument as the body-to-NeRF scenario to derive collision. Because they are different objects, the density fields ρ_1 and ρ_2 are spatially independent. Hence the probability that K is non-empty is the product of the probability of K being non-empty due to ρ_1 and the probability of K being non-empty due to ρ_2. This allows us to directly use (9) to calculate the NeRF-NeRF collision probability

$$Pr(N(K) > 0) = \left(1 - \exp\left[- \int_K \frac{\rho_1(x)}{wh} \, dx \right]\right) \times \left(1 - \exp\left[- \int_K \frac{\rho_2(x)}{wh} \, dx \right]\right). \quad (11)$$

The scaling factor(s) can be adjusted to reflect the camera parameters used to train a particular object if different cameras were used to train the objects.

The maximal collision set K need not be explicitly found when evaluating (11), as multiplying the integrands with an indicator function

$$\phi(x) = \begin{cases} 1 & \rho_1(x) \times \rho_2(x) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

and integrating over a set containing both objects (e.g. $\mathbb{R}^3$) yields identical results. While we do not use this computation in the remainder of this work, this could be used to quantify collision probabilities between multiple NeRF objects in tasks such as manipulating sensed objects (represented as a NeRF) through a sensed environment (also as a NeRF).

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 (9). 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 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. Our solution is real-time and we show that it is not overly-conservative.

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 be in violation of the chance constraint.

Similar to classical configuration space planning [40], 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. We first voxelize the space of the map and label each cell with the integral of the NeRF density (normalized by the pixel dimensions) over each cell to give the voxelized *cell* intensity grid, $\mathbb{I}_c$. This computation yields a log-probability of occupancy for each cell. 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 it were located in each cell; we call this the robot kernel, $\mathbb{K}$. Finally, we convolve the robot

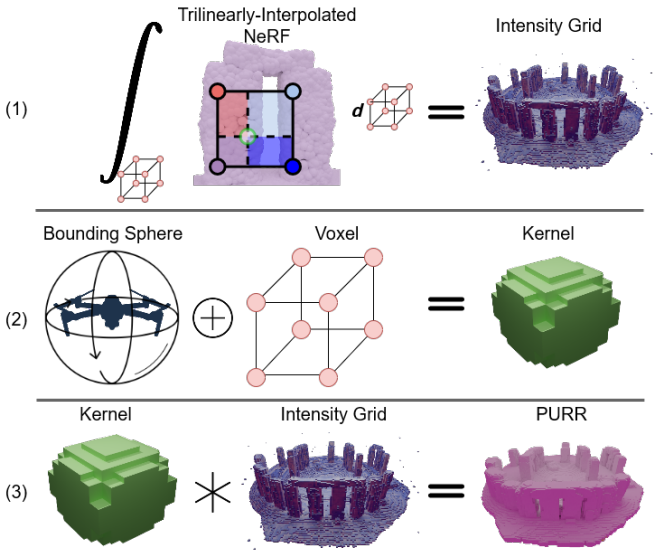

Fig. 4: NeRF to PURR pipeline. (1) A density grid is sampled from the NeRF, which is then trilinearly interpolated and integrated over a particular voxel to retrieve the cell intensity grid. (2) A robot kernel is generated by taking the Minkowski sum between the minimal bounding sphere of the robot and 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 to create the PURR.

kernel $\mathbb{K}$ with the cell intensity grid $\mathbb{I}_c$ and exponentiate the result to obtain the *robot* intensity grid, $\mathbb{I}_r$. The *robot* intensity grid gives, in each cell, an upper bound on the probability that a robot, whose position is somewhere in that cell, is in collision with the NeRF. Finally, we 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 negative log-probability of no collisions occuring in voxel v_{ijk}

$$\mathbb{I}_c(v_{ijk}) = \int_{v_{ijk}} \frac{\rho(x)}{wh} dx. \tag{13}$$

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], [23]) 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 [41],

$$\begin{aligned} \hat{\rho}(x,y,z) =& c_1 + c_2 x + c_3 y + c_4 z + \\ & c_5 xy + c_6 yz + c_7 xz + c_8 xyz. \end{aligned} \tag{14}$$

The coefficients $c_{1:8}$ are the solution of a linear system $\mathbf{A}_{1:8}c_{1:8} = \rho_{1:8}$, 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 (13) plugging in (14) 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[3] 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, and exponentiating the result to give the desired probability of collision as follows,

$$\begin{aligned}
\mathbb{I}_r(v_{ijk}) =& 1 - \exp\left[-\sum_{v_{lmn}\in\mathbb{K}(v_{ijk})} \mathbb{I}_c(v_{lmn})\right] \\
=& 1 - \exp\left[-\sum_{v_{lmn}\in\mathbb{K}(v_{ijk})} \int_{v_{lmn}} \frac{\hat{\rho}(x)}{wh} dx\right] \\
\geq& 1 - \exp\left[-\int_{B(\mathcal{T})} \frac{\hat{\rho}(x)}{wh} dx\right] \forall R \in \mathbb{SO}(3), \mathbf{p} \in v_{ijk} \\
\approx& 1 - \exp\left[-\int_{B(\mathcal{T})} \frac{\rho(x)}{wh} dx\right] \forall R \in \mathbb{SO}(3), \mathbf{p} \in v_{ijk}.
\end{aligned} \tag{15}$$

Finally, the PURR $\mathbb{P}$ is generated by thresholding the *robot* intensity grid by the collision probability threshold σ,

$$\mathbb{P}(v_{ijk}) = \mathbb{1}\left[\mathbb{I}_r(v_{ijk}) > \sigma\right]. \tag{16}$$

The resulting PURR is visualized for different probability thresholds σ in Fig. 5 for the Stonehenge NeRF environment. As shown in the bottom row of Fig. 5, the PURR gives a similar effect as computing a voxel map based on thresholding the NeRF density directly, but the threshold for the PURR is calibrated to a precise probability of collision, and gives more regular voxelized maps, while thresholding on the density directly is not interpretable in terms of safety, and gives more irregular voxelized maps.

[3]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.

Finally, we note that the trilinear interpolation of the density (14) to compute the integral in (13) 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 (14),*

$$\begin{aligned}
\alpha \geq \max_{i,j,k} \Big\{ & \exp\left[-\mathbb{I}_r(v_{ijk})\right] \\
& - \min_{\mathbf{p}\in v_{ijk}, R\in\mathbb{SO}(3)} \exp\left[-\int_{B(\mathcal{T})} \lambda(x) dx\right] \Big\}.
\end{aligned}$$

Finally, if α exists, then (16) and consequently our PURR are inflated with this collision offset factor to give the PURR a rigorous collision probability guarantee

$$\mathbb{P}(v_{ijk}) = \mathbb{1}\left[1 - \exp[-\mathbb{I}_r(v_{ijk})] > \sigma - \alpha\right]. \tag{17}$$

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 (15), 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 (9). □

Remark 1. *If the discretizations match between the PURR and a voxel-based NeRF architecture, as in [23] then α is*

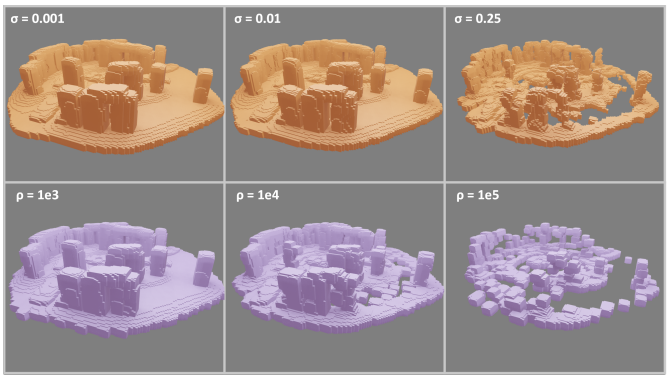

Fig. 5: (Top Row) PURR with varying σ values. (Bottom Row) density-thresholded voxel maps with varying ρ values. The PURR is precisely calibrated to give a desired probability of collision, while thresholding the NeRF density directly offers no particular safety guarantee.

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.

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.*

Remark 3. *The creation of a PURR with state uncertainty $\hat{\mathbb{P}}$ is located in Appendix B. The state uncertain PURR $\bar{\mathbb{P}}$ retains the original probabilistic collision guarantee.*

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 continuous path 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 point-wise 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 complement 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). 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 complement. 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 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. To accelerate computation, we use an efficient A* implementation [42] and pre-compute the connectivity graph of the PURR to be reused for path queries online.

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 will first generate a "tube" around our A* path that is both large (so we minimally constrain our trajectory optimization) and lies in the complement 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.

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

$$\mathbf{p}(t;\mathbf{s}_k) = \sum_{k=0}^{N} \binom{N}{k} (1-t)^{N-k} t^k \mathbf{s}_k, \tag{18}$$

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 [43]. 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 a common method for enforcing safety constraints in the path planning literature [44].

To find this trajectory, suppose we have L bounding boxes $\{B_1, \ldots, B_L\}$ 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 B_i. Since the Bézier curves are linear functions of the control points, we can in

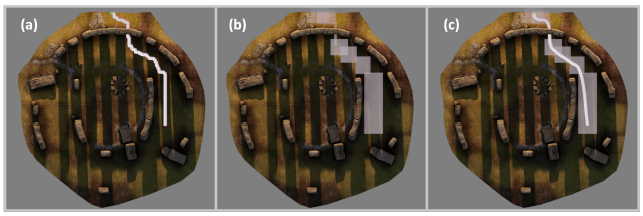

Fig. 6: (a) Discrete path returned by A*, (b) Union of bounding boxes containing subsets of the A* path whilst remaining strictly in the complement of the PURR, (c) Smooth path represented as the union of Bézier curves whose control points lie within a particular bounding box.

turn represent the i^{th} curve as $\mathbf{p}_i(t) \equiv \beta(t)\mathbf{s}^i$, where $\mathbf{s}^i$ a concatenated vector of all the control points for curve i, and $\beta(t) : [0,1] \mapsto \mathbb{R}^{3\times 3(N+1)}$ is a coefficient matrix that only depends on the curve parameter t. We can similarly represent higher derivatives of the curve as $\mathbf{p}_i^{(d)}(t) = \beta^{(d)}(t)\mathbf{s}^i$. We refer the reader to [43] for a more detailed treatment of Bézier curves and splines.

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

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

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 [45].

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

$$\begin{aligned} \min_{\mathbf{s}^1,\dots,\mathbf{s}^L} \quad & J(\mathbf{s}^1, \dots, \mathbf{s}^L) && (19) \\ \text{s.t.} \quad & s_j^i \in B^i, && \forall i \le L,\ j \le N \\ & \beta^{(d)}(1)\mathbf{s}^i = \beta^{(d)}(0)\mathbf{s}^{i+1}, && \forall i \le L,\ d \le D \\ & \beta(0)\mathbf{s}^1 = \mathbf{p}_0, \\ & \beta(1)\mathbf{s}^L = \mathbf{p}_f \end{aligned}$$

In particular, we constrain the control points of every segment so that $\mathbf{s}^i$ must lie in the corresponding bounding box 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 $N = 5$, 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 (19) is a quadratic program (QP) that can be solved in real time.

Corollary 1. *The trajectory given by the solution of the QP* (19) *is probabilistically safe.*

Proof. The QP (19) 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 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* (19) *(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 [45]) 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. Using our proposed trajectory optimizer, we generate trajectories for a quadrotor flying through the scene, and study the safety and efficiency 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]. We demonstrate that both voxel methods are more computationally efficient than NeRF-Nav, and also generate trajectories that are safer (with fewer collisions) and more efficient (requiring less control effort). Further, we find that our method, CATNIPS, allows the user to set a clearly defined probability threshold 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. 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 randomized start and goal locations within Stonehenge are shown in Figure 7. The PURR was generated using a resolution of 200 voxels per side and $\sigma = 0.01$. Bézier curves of degree 8 were used in planning through the PURR free space. 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 [45].

All paths are visually collision-free, smooth, and non-degenerate. Figure 5 shows the resulting PURR of Stonehenge at various values of σ, and the baseline occupancy map computed for various of the NeRF density threshold, ρ. We

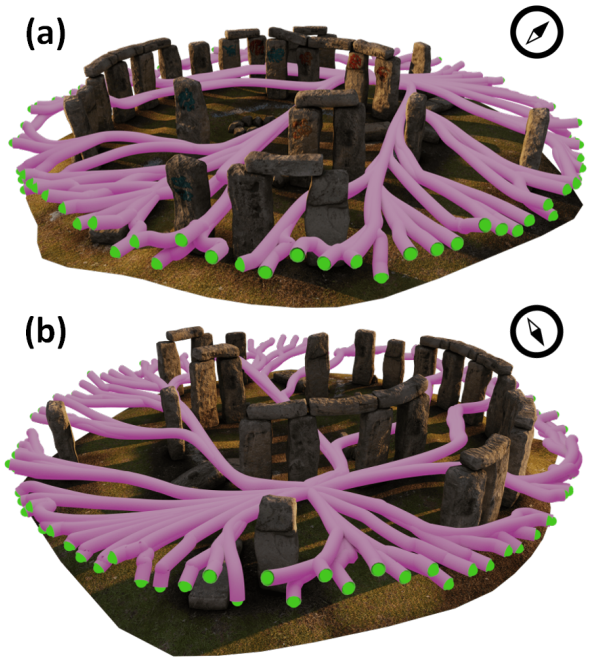

Fig. 7: Generated safe paths across 100 configurations within Stonehenge.

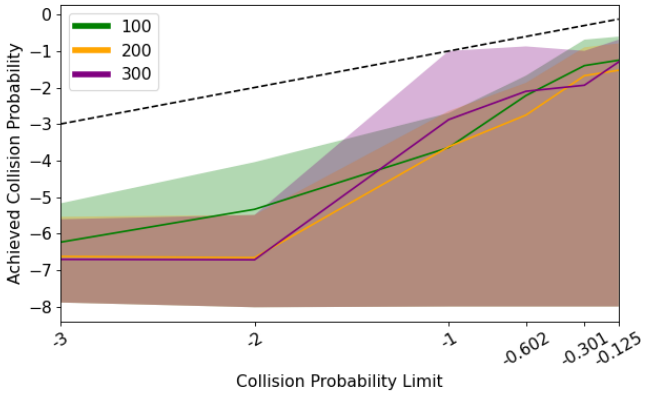

Fig. 8: Ablation of collision probability satisfaction (with max/min spread) for grid sizes of $(100)^3$, $(200)^3$, and $(300)^3$ for the same environment. Results are shown on a log10-log10 plot. The dashed-line has a slope of one, representing perfect agreement between the threshold and the empircal collision probabilities. Below the line is safe, above is in violation of the probability bound. Satisfaction of the collision probability is not very sensitive to the grid discretization. All lines follow the expected trend.

find that the PURRs are well-behaved across a wide range of σ values, in contrast to the density threshold map. Visually, the density-based map is typically more conservative at low ρ (e.g. no gap exists in the front pillars for $\rho = 10^3$, but exists in $\sigma = 10^{-3}$) yet degenerates very quickly with large ρ. For example, the floor begins to disappear at $\rho = 10^4$ and is largely gone at $\rho = 10^5$ compared to the PURR at $\sigma = 0.25$. Therefore, we argue that the well-behavedness of the PURR contributes to the safety and non-conservatism of the generated paths.

First, we validate the safety of the same 100 randomized PURR paths in Fig. 8 over various voxel grid resolutions. The figure has the empirical collision rate on the horizontal axis and the desired collision probability on the vertical, so curves below the dotted line meet the collision probability bound, while curves above violate the bound. We see that all curves for CATNIPS are below the dotted line across all scene discretizations, indicating that the planned trajectories are probabilistically safe regardless of descritization level. We also see a trend toward less conservatism as the collision probability bound, σ, is increased. In fact, some trajectories reach just under the dotted line, illustrating that our paths are not overly conservative. Moreover, we note that the collision probability does not vary significantly based on the discretization resolution; thus, we can use rather coarse and more computationally efficient PURRs while still achieving safety.

Finally, we compare our method to two baselines: the "baseline grid" which computes occupancy from a sensity threshold, and NeRF-Nav [12], a gradient-based trajectory planner for NeRFs in Fig. 9. We analyze their performance on two metrics: the minimum distance of the path to the ground-truth mesh (negative is within the mesh) to quantify safety, and the difference in lengths between the generated path and the shortest straight line path to quantify efficiency. We evaluate the algorithms on the same 100 randomized configurations used to evaluate the collision probability in Figure 8. Again, we note that NeRF-Nav and the baseline paths have no safety guarantees and require parameter tuning to get a desired safety performance. Meanwhile, our method works out of the box by simply choosing the collision probability bound, σ.

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). As we increase the weighting on the collision penalty, we do see that the algorithm can perform safely on average. However, such a high collision penalty (10^9) will typically cause numerical issues in the trajectory optimizer. The trajectories also deviate from the shortest path the farthest, illustrating conservatism and non-smooth paths.

We see that our method, CATNIPS, is safer in the worst-case trajectories (higher bottom whisker) for low $\sigma = 10^{-3}$ compared to the baseline paths with low $\rho = 10^3$, and is safer on average at high values ($\sigma = 0.25, \rho = 10^5$). Even though the baseline voxel grid can visually approximate the ground-truth (Fig. 5, $\rho = 10^3$), we see that the generated trajectories can still be unsafe, rendering the baseline unreliable. In particular, for $\sigma = 0.001$, we can see that our proposed method is safer

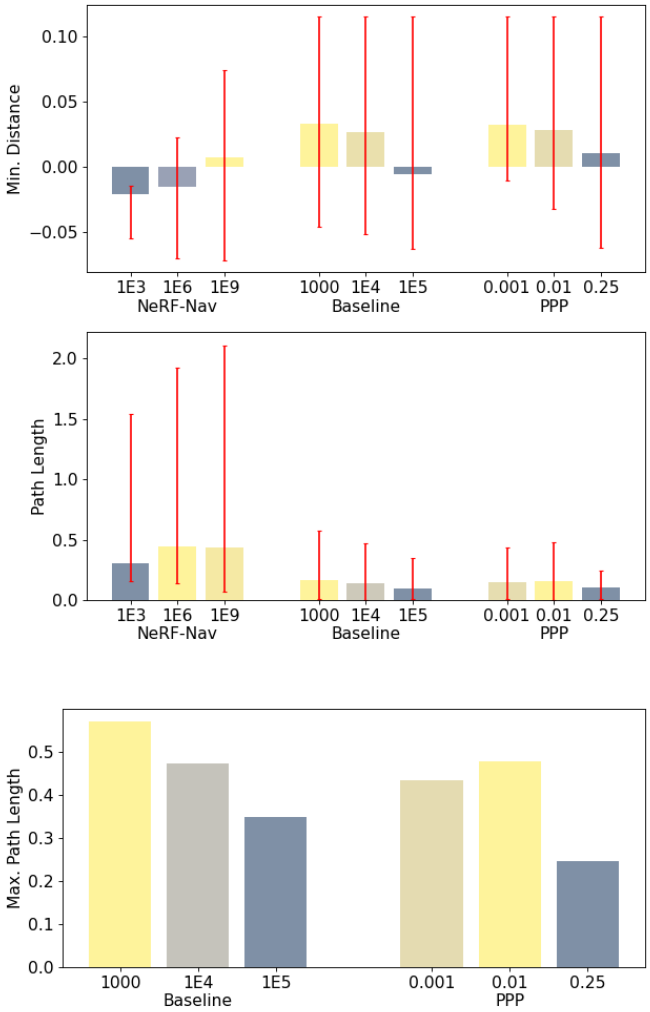

Fig. 9: Statistics over 100 runs at each of three parameter choices for our PPP method, the baseline grid, and NeRF-Nav [12] in the Stonehenge environment. Whiskers indicate max/min. The minimum distance in the top plot is to the ground-truth mesh (not known to the planner). Top: our PPP method gives an interpretable probability parameter (we can tune the lower whisker) without being overly conservative (lower path length). The parameters of the baseline and the NeRF-Nav algorithms do not have a probability interpretation, and can lead to large collision violations (low whiskers) and overly-conservative paths. Mid and Bottom: NeRF-Nav gives longer paths, while in the worst case, the baseline is more conservative than our method for similar safety performance.

than the baseline method (with smaller interpenetration in both the mean and worst-case) with ρ thresholds that generate paths of similar lengths.

We can also see the irregularity of the baseline grid come into play when analyzing the most conservative paths (i.e. paths corresponding to top whisker in the path length metric). For paths with similar safety performance between the PURR and baseline grid, baseline paths can be more conservative.

B. Computation Times

The implementation of the above algorithms are performed in Pytorch on an RTX 3090Ti and Intel i9 10900. Our method is built on top of the Pytorch Instant-NGP implementation [46], which we did not alter. Almost no code optimizations were performed to achieve real-time performance. Under an online NeRF training scenario, the entire pipeline consisting of PURR creation (0.1 s), A* (0.1 s), and Bézier curve generation (0.02 s) operates at 4 Hz (Table I). If we assume a static NeRF scenario, the PURR can be created offline and the A* paths can be cached. More specifically, the A* implementation [42] can simultaneously solve A* from the goal cell to all the other cells. Online replanning then consists of a lookup of the A* path from the current position (5×10^{-4} s) and solving for Bézier curves. Additional optimizations can be done in reducing PURR creation time, as well as investigating methods of creating a trajectory initialization quicker by processing on GPU.

Note that the proposed method produces smooth trajectories from the current position to the goal. It is important to note that in an online replanning 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 much longer to converge (if at all) to a reasonable tolerance, without any safety guarantees. We believe this is primarily due to difficulty of optimizing its highly nonconvex objective and the many queries required to the density neural network.

Computation Time		
Operation	PURR	NeRF-Nav
Offline	0.95	4.92
Voxel Grid	0.1	0.1
A* Graph	0.85	0.85
Gradients	N/A	3.97⋆
Online	0.02	2.0
Querying Path	5e-4	N/A
B-Spline	0.015	N/A
Gradients	N/A	2.0⋆⋆

⋆ 200 gradient steps.
⋆⋆ 100 gradient steps.

TABLE I: Timing results between our voxel methods (PURR and level-set) and NeRF-Nav. Because both voxel methods use similar operations, they have identical times.

VIII. Conclusion

In this paper, we present a novel method for chance-constrained trajectory optimization through NeRF scenes. We present an interpretation of the NeRF density channel as 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 more efficient paths than a state-of-the-art method [12] for trajectory planning through NeRFs, and also give more well-behaved and more user interpretable results than a baseline planer 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 work. 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., [17], [15]) to perform active exploration or "next-best-view planning" on NeRFs, allowing a robot to autonomously explore a novel environment using only onboard vision. Another interesting direction is to consider modifications to the NeRF rendering using insights gleaned from the PPP interpretation introduced in this work. In particular, considering pyramidal or conic "beams" instead of rectangular prisms could prove helpful for improving NeRF training on unbounded scenes or to reduce aliasing. Finally, the probabilistic collision framework developed here could have interesting applications to problems like differentiable simulation of rigid bodies represented as NeRFs [47] or planning for problems like contact-rich manipulation and locomotion.

Appendix A
Integration over Trilinearly Interpolated Cells

For a trilinearly interpolated density over a cell v_{ijk} given by (14) 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:

$$\begin{aligned}\int_{a_x}^{b_x}\int_{a_y}^{b_y}\int_{a_z}^{b_z}\rho(x)dxdydz &= (b_x-a_x)(b_y-a_y)(b_z-a_z)\\ &+\frac{c_2}{2}(b_y-a_y)(b_z-a_z)(b_x^2-a_x^2)\\ &+\frac{c_3}{2}(b_x-a_x)(b_z-a_z)(b_y^2-a_y^2)\\ &+\frac{c_4}{2}(b_x-a_x)(b_y-a_y)(b_z^2-a_z^2)\\ &+\frac{c_5}{4}(b_z-a_z)(b_x^2-a_x^2)(b_y^2-a_y^2)\\ &+\frac{c_6}{4}(b_x-a_x)(b_y^2-a_y^2)(b_z^2-a_z^2)\\ &+\frac{c_7}{4}(b_y-a_y)(b_x^2-a_x^2)(b_z^2-a_z^2)\\ &+\frac{c_8}{8}(b_x^2-a_x^2)(b_y^2-a_y^2)(b_z^2-a_z^2)\end{aligned} \tag{20}$$

where c_i are the coefficients of the trilinear interpolation.

Appendix B
Probabilistic Collision with State Uncertainty

The results of V-A can be extended to account for state uncertainty in the agent body. The extent of $B(\mathcal{T})$ is a function of the pose random variable. The collision probability with state uncertainty is a marginalization over the support of the pose distribution. Explicitly,

$$\begin{aligned}Pr(N(B(\mathcal{T}))=0) &= \int_S Pr(N(B(\mathcal{T}))=0 \mid \mathcal{T})\; p(\mathcal{T})d\mathcal{T}\\ &= \int_S \exp\left[-\int_{B(\mathcal{T})}\frac{\rho(x)}{wh}\,dx\right]p(\mathcal{T})d\mathcal{T},\end{aligned} \tag{21}$$

where S is the support.

Due to the voxel nature of the PURR, state uncertainty can be incorporated into the representation cheaply. We first assume that the state uncertainty distribution is rotationally invariant (i.e. $\mathbf{p}$ is the only random variable) and that the support S is parametrized by its mean μ. We voxelize our state uncertainty equation (21)

$$\begin{aligned}\bar{\mathbb{I}}_r(v_{ijk},\mu) &= 1-\sum_{v_{lmn}\in\mathbb{S}(v_{ijk})}\exp\left[-\mathbb{I}_r(v_{lmn})\right]\bar{p}(v_{lmn})\\ &\geq 1-\int_S\exp\left[-\int_{B(\mathcal{T})}\frac{\rho(x)}{wh}dx\right]p(\mathbf{p})d\mathbf{p},\end{aligned} \tag{22}$$

where $\mathbb{S}$ is the voxelized support of the state and $\bar{p}(v_{lmn})$ is the accumulated density (i.e. discrete probability) within voxel v_{lmn}

$$\bar{p}(v_{lmn}) = \int_{v_{lmn}} p(\mathbf{p})d\mathbf{p}. \tag{23}$$

This operation is similar to the creation of the robot kernel, replacing the binary values with values of discrete probability. The operation is illustrated in Fig. 10c. Although in theory $\mathbf{p}$ can have unbounded support, in practice a probability cutoff is necessary so that the kernel used in `Conv3D` is finite.

Note that we can account for bounded uncertainty without reasoning about the distribution by simply selecting the cell with the maximum *robot* intensity in $\mathbb{S}$. This is a `MaxPool3D` operation using $\mathbb{S}$ as the kernel (Fig. 10b).

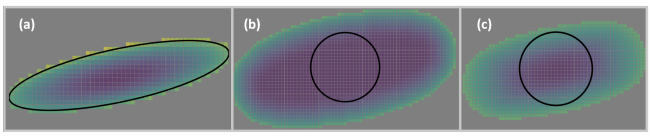

Fig. 10: (a) Bounded state uncertainty ellipse, (b) Dilation of obstacle reasoning only on the uncertainty bounds (equivalent to `MaxPool3D` operation), (c) Inflation of obstacle by reasoning over the distribution (equivalent to `Conv3D` operation). The black circle in (b) and (c) denotes the true obstacle.

Acknowledgements

We would like to thank 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," *arXiv:2003.08934 [cs]*, Aug. 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," 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," 2022.

[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*, 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] M. Tong, C. Dawson, and C. Fan, "Enforcing safety for vision-based controllers via Control Barrier Functions and Neural Radiance Fields," Sep. 2022.

[15] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022.

[16] 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.

[17] D. Maggio, M. Abate, J. Shi, C. Mario, and L. Carlone, "Loc-nerf: Monte carlo localization using neural radiance fields," 2022. [Online]. Available: https://arxiv.org/abs/2209.09050

[18] X. Pan, Z. Lai, S. Song, and G. Huang, "Activenerf: Learning where to see with uncertainty estimation," 2022. [Online]. Available: https://arxiv.org/abs/2209.08546

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

[20] N. Sünderhauf, J. Abou-Chakra, and D. Miller, "Density-aware nerf ensembles: Quantifying predictive uncertainty in neural radiance fields," 2022. [Online]. Available: https://arxiv.org/abs/2209.08718

[21] J. Shen, A. Ruiz, A. Agudo, and F. Moreno-Noguer, "Stochastic neural radiance fields: Quantifying uncertainty in implicit 3d representations," 2021. [Online]. Available: https://arxiv.org/abs/2109.02123

[22] 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

[23] Sara Fridovich-Keil and Alex Yu, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa, "Plenoxels: Radiance fields without neural networks," in *CVPR*, 2022.

[24] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa, "PlenOctrees for real-time rendering of neural radiance fields," in *ICCV*, 2021.

[25] 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," *ICCV*, 2021.

[26] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, "Mip-nerf 360: Unbounded anti-aliased neural radiance fields," *CVPR*, 2022.

[27] X. Wu, S. Chen, K. Sreenath, and M. W. Mueller, "Perception-aware receding horizon trajectory planning for multicopters with visual-inertial odometry," 2022. [Online]. Available: https://arxiv.org/abs/2204.03134

[28] 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.

[29] L. Han, F. Gao, B. Zhou, and S. Shen, "Fiesta: Fast incremental euclidean distance fields for online motion planning of aerial robots," *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4423–4430, 2019.

[30] 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.

[31] 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.

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

[33] B. Luders, M. Kothari, and J. How, "Chance constrained rrt for probabilistic robustness to environmental uncertainty," *AIAA Guidance, Navigation, and Control Conference*, 08 2010.

[34] J. L. Schonberger and J.-M. Frahm, "Structure-from-motion revisited," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4104–4113.

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

[36] 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.

[37] P. S. Heckbert and P. Hanrahan, "Beam tracing polygonal objects," in *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '84. New York, NY, USA: Association for Computing Machinery, 1984, p. 119–127. [Online]. Available: https://doi.org/10.1145/800031.808588

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

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

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

[41] P. L. Williams and N. Max, "A volume density optical model," in *Proceedings of the 1992 Workshop on Volume Visualization*, ser. VVS '92. New York, NY, USA: Association for Computing Machinery, 1992, p. 61–68. [Online]. Available: https://doi.org/10.1145/147130.147151

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

[43] K. I. Joy, "BERNSTEIN POLYNOMIALS."

[44] 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.

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

[46] J. Tang, "Torch-ngp: a pytorch implementation of instant-ngp," 2022, https://github.com/ashawkey/torch-ngp.

[47] S. L. Cleac'h, H.-X. Yu, M. Guo, T. A. Howell, R. Gao, J. Wu, Z. Manchester, and M. Schwager, "Differentiable Physics Simulation of Dynamics-Augmented Neural Objects," Oct. 2022.