Vision-Only Robot Navigation in a Neural Radiance World

Michal Adamkiewicz, Timothy Chen, Adam Caccavale, Rachel Gardner, Preston Culbertson, Jeannette Bohg, and Mac Schwager, *Member, IEEE*

***Abstract*—Neural Radiance Fields (NeRFs) have recently emerged as a powerful paradigm for the representation of natural, complex 3D scenes. Neural Radiance Fields (NeRFs) represent continuous volumetric density and RGB values in a neural network, and generate photo-realistic images from unseen camera viewpoints through ray tracing. We propose an algorithm for navigating a robot through a 3D environment represented as a NeRF using only an onboard RGB camera for localization. We assume the NeRF for the scene has been pre-trained offline, and the robot's objective is to navigate through unoccupied space in the NeRF to reach a goal pose. We introduce a trajectory optimization algorithm that avoids collisions with high-density regions in the NeRF based on a discrete time version of differential flatness that is amenable to constraining the robot's full pose and control inputs. We also introduce an optimization based filtering method to estimate 6DoF pose and velocities for the robot in the NeRF given only an onboard RGB camera. We combine the trajectory planner with the pose filter in an online replanning loop to give a vision-based robot navigation pipeline. We present simulation results with a quadrotor robot navigating through a jungle gym environment, the inside of a church, and Stonehenge using only an RGB camera. We also demonstrate an omnidirectional ground robot navigating through the church, requiring it to reorient to fit through a narrow gap.**

***Index Terms*—Collision avoidance, localization, motion and path planning, vision-based navigation, neural radiance fields.**

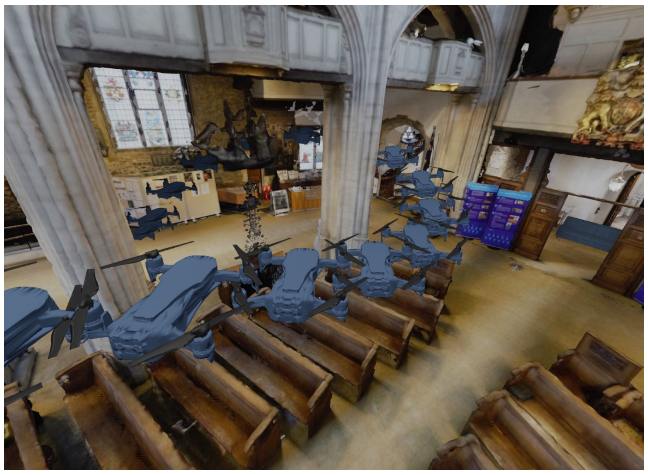


Fig. 1. A drone navigating through the interior space of a church using a monocular camera. The environment is modeled as a Neural Radiance Field (NeRF), a deep-learned geometry representation. The trajectory, which is optimized to minimize a NeRF-based collision metric, can be continually replanned as the drone updates its state estimate based on captured images.

I. Introduction

Planning and executing a trajectory with onboard sensors is a fundamental building block of many robotic applications, from manipulation to autonomous driving or drone flight. Robot navigation methods depend on properties of the underlying environment representation, whether it is a voxel grid, a point cloud, a mesh model, or a Signed Distance Field (SDF). Recently there has been an explosion of interest in a deep-learned geometric representation called NeRFs due to their ability to compactly encode detailed 3D geometry and color [1]. NeRFs take a collection of camera images and train a neural network to give a function relating each 3D point in space with a density and a vector of RGB values (called a "radiance"). This representation can then generate synthetic photo-realistic images through a differentiable ray tracing algorithm. In this paper, we propose a navigation pipeline for a robot given a pre-trained NeRF of its environment. A block diagram is shown in Fig. 2 We use the density of the NeRF to plan dynamically feasible, collision-free trajectories for a differentially flat robot model. We also build a filter to estimate the dynamic state of the robot given an onboard RGB image, using the image synthesis capabilities of the NeRF.

We combine the trajectory planner and the filter in a receding horizon loop to provide a full navigation pipeline for a robot to dynamically maneuver through an environment (an example is shown in Fig. 1) using only an RGB camera for feedback. While some existing vision-only navigation systems [2] have seen recent success with end-to-end approaches, others such as [3] have advocated for a modularization of learned perception and

Manuscript received September 9, 2021; accepted January 20, 2022. Date of publication February 11, 2022; date of current version March 1, 2022. This letter was recommended for publication by Associate Editor Youcef Mezouar and Editor Eric Marchand upon evaluation of the reviewers' comments. The work of Preston Culbertson was supported by NASA Space Technology Research Fellowship under Grant NSSC18K1180. This work was supported in part by NSF NRI under Grant 1830402, and in part by ONR under Grant N00014-18-1-2830, Siemens and the Stanford Data Science Initiative. *(Michal Adamkiewicz and Timothy Chen contributed equally to this work.) Corresponding author: Michal Adamkiewicz.)*

Michal Adamkiewicz, Rachel Gardner, and Jeannette Bohg are with the Department of Computer Science, Stanford University, Stanford, CA 94305 USA (e-mail: mikadam@stanford.edu; rachel0@stanford.edu; bohg@stanford.edu).

Timothy Chen and Mac Schwager are with the Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305 USA (e-mail: chengine@stanford.edu; schwager@stanford.edu).

Adam Caccavale and Preston Culbertson are with the Department of Mechanical Engineering, Stanford University, Stanford, CA 94305 USA (e-mail: adam.w.caccavale@gmail.com; pculbertson@stanford.edu).

Videos of this work can be found at https://mikh3x4.github.io/nerf-navigation/.

Digital Object Identifier 10.1109/LRA.2022.3150497

control and achieved impressive results. While their perception system aims to generalize over variations of drone race tracks it requires very specific training data and labels. We take a similar approach, and focus on NeRFs as a geometric environment representation that enables any robot, e.g. drones or ground robots, to navigate through it.

NeRFs present a range of potential advantages as an environment representation for robots. Unlike voxel models, point clouds, or mesh models, they are trained directly on dense photographic images without needing precise feature extraction, matching, and alignment [4], [5]. They inherently represent the geometry as a continuous density field, and so they are well-suited to robot motion planning and trajectory optimization using gradient methods, or used with differentiable simulators. They can also produce photo-realistic synthetic images, giving a mechanism for a robot to "hallucinate" what it would expect to see if it were to take different actions. Additionally, NeRFs are able to store geometry efficiently in memory as neural network weights rather than as dense point clouds or voxel grids, making them ideal for use in memory-constrained systems like robots.

Extensions of NeRFs have been developed to handle transparent objects [6], segment and recompose objects in a scene [7]–[9], and render moving and deformable objects [10], including humans [10] and human faces [11]. Variants of NeRFs can also incorporate a prior over objects or scenes, in order to quickly adapt to new environments with a handful of images [12], [13]. While the standard NeRF training and image rendering pipeline is slow, recent developments have accelerated image synthesis from a NeRF to 200 frames per second on a GPU [14], fast enough for use in a real-time robot control loop. We envision a future where all these improvements could be leveraged to create a fully NeRF based environment representation that combines complex geometry, semantic understanding, and real-time performance. However, for robots to harness the advantages of the NeRF representation for navigation, a trajectory planner and pose filter designed to work specifically with the NeRF machinery are needed.

We address this need in this paper by proposing:

1) a new trajectory planning method, based on differential flatness, that plans full, dynamically feasible trajectories to avoid collisions with a NeRF environment,
2) an optimization-based filter to obtain a posterior estimate of the full dynamic state, balancing a dynamics prediction loss and a NeRF-based photometric loss, and
3) an online replanning controller that combines both of the above algorithms in feedback to robustly navigate a robot through its environment using a NeRF model.

We demonstrate results in a variety of high fidelity simulation environments, and perform ablation studies to showcase the advantages provided by each part of our navigation framework. We run our navigation pipeline with custom-trained NeRF models of a playground, a church, and Stonehenge. We then evaluate the performance of our trajectory planner and pose estimator on the underlying ground truth mesh models, not the trained NeRF models, thereby demonstrating robustness to model mismatch between the real-world scene and the trained NeRF.

II. Related Work

A. Neural Implicit Representations

Neural implicit representations use a neural network to represent the geometry (and sometimes the color and texture) of a complex 3D scene. Generally, neural implicit representations take a labeled data set and learn a function of the form $f_\theta(p) = \sigma$, where f is a neural network parameterized by the weights θ, p is a low-dimensional query point such as an (x, y, z) coordinate, and σ is some (usually scalar) quantity of interest. Aside from NeRFs, there are several other approaches to implicit representations including learned SDFs [15]–[17] and Occupancy Networks [18].

However, there currently exists little work studying how to leverage NeRFs for applications beyond novel view synthesis. Recent work [4] has treated the problem of mapping and online NeRF construction from visual data; the authors demonstrate competitive accuracy with traditional SLAM pipelines, and realtime performance. This work's state estimator builds on [19], which presents a method for single-image camera pose estimation using a pre-trained NeRF representation of the environment. The method we present here for state estimation also uses maximum likelihood estimation (MLE), but we instead treat the problem as recursive Bayesian estimation, which incorporates system dynamics and must propagate uncertainty between timesteps.

B. Trajectory Optimization

Optimal control remains a fundamental tool in robotic motion planning. Of particular interest is the problem of trajectory optimization [20], which seeks a system trajectory $\mathbf{x}(t)$ and open-loop inputs $\mathbf{u}(t)$ that optimize a control objective, subject to state and input constraints. While there exists a vast literature on trajectory optimization for robot motion planning [21], our discussion here will focus specifically on collision avoidance in trajectory optimization, which remains unstudied for environments represented as NeRFs.

One approach to model an environment is as an SDF, which represents obstacles as the zero-level set of a nonlinear function $d(\mathbf{x})$, which takes negative values inside the obstacle, positive values outside the obstacle, and has magnitude equal to the distance between $\mathbf{x}$ and the obstacle boundary. Collision avoidance is typically imposed as a constraint in the trajectory optimization, requiring the SDF for all obstacles to be non-negative at all points on the robot body along the trajectory. This formalism has received particular interest as a map representation following the success of KinectFusion [22], which constructs truncated SDFs using RGB-D data. Works such as [23] and [24] present methods for incrementally constructing SDF-like map representations and using them for online motion planning.

Perhaps closest to this work's trajectory optimizer is CHOMP [25], [26], a family of gradient-based methods which optimizes a finite sequence of poses, with an objective which encourages the trajectory to be smooth and to avoid collision. Specifically, CHOMP represents obstacle geometry by pre-computing each obstacle's SDF on a finite grid, and approximates SDF gradients using finite differences or interpolation. In [27], the authors present a similar gradient-based method which optimizes quadrotor position trajectories to minimize a perception-aware collision metric based on an SDF-like map. In contrast, the NeRF geometry representation used here is continuous in itself, and of arbitrary resolution, with continuous gradients that can be efficiently computed using automatic differentiation. Further, our method generates trajectories that are constrained to be dynamically feasible rather than imposing the system dynamics via a cost.

III. Problem Formulation

This paper proposes a method for navigating a robot through an environment represented by a NeRF. A NeRF ($N : \mathbb{R}^3 \times \mathbb{R}^2 \mapsto \mathbb{R}^3 \times \mathbb{R}_+$) maps a 3D location $\mathbf{p} = (x, y, z)$ and view direction (θ, ϕ) to an emitted color $\mathbf{c} = (r, g, b)$ and scalar density ρ. For notational convenience, we define $\rho(\mathbf{p})$ as the density output of the NeRF evaluated at position $\mathbf{p}$ (note ρ depends only on position). Similarly, we define $C_i : \mathrm{SE}(3) \mapsto \mathbb{R}^3$ as the expected color of pixel i when rendering the NeRF from the camera pose $T \in \mathrm{SE}(3)$, where SE denotes the special Euclidean group.

In this paper, we consider the problem of a mobile robot, equipped only with a monocular camera, which seeks to navigate an environment. Specifically, the robot seeks to plan and track a collision-free path from its initial state $\mathbf{x}_0$ to a goal state $\mathbf{x}_f$. The robot has access to a NeRF representation of the environment which it can use for both planning (i.e., for evaluating the probability of collision for a given trajectory) and localization.

We approximate the robot body using a finite collection of points $\mathcal{B}$ at which collision is checked. Typically this will be a 3 d grid of points representing the robot's bounding box, however it can also be an arbitrarily complex model. However, it is not obvious how the NeRF density at a point relates to its occupancy. Specifically, the NeRF density represents the differential probability of a given spatial point stopping a ray of light [1]. We assume the probability of terminating a light ray is a strong proxy for the probability of terminating a mass particle. Thus, the collision probability at time t is given by

$$p_t^{\text{coll}} = P\left(\bigcup_{\mathbf{b}_t \in \mathcal{B}} \mathbf{b}_t \in \mathcal{X}_{\text{coll}}\right) \le \sum_{\mathbf{b}_t \in \mathcal{B}} \rho(\mathbf{b}_t)\, s(\mathbf{b}_t), \qquad (1)$$

where $\mathcal{X}_{\text{coll}}$ denotes the collision set, $s(\mathbf{b}_t)$ is the distance traveled by a body-fixed point $\mathbf{b}$ in timestep t, and the bound follows from Boole's inequality. In this work, we include the collision probability as a cost to be minimized during trajectory optimization; an alternative approach would be to impose a chance constraint on the optimization, which would require more sophisticated optimization techniques.

Given a Gaussian estimate of its current state, $\mathcal{N}(\mu_t, \Sigma_t)$, the robot plans a series of waypoints that avoid regions of high density in the NeRF. After taking a control action, the robot receives an image of the environment, and updates its belief about its current state. Finally, the robot replans the trajectory using the latest estimate as the first state.

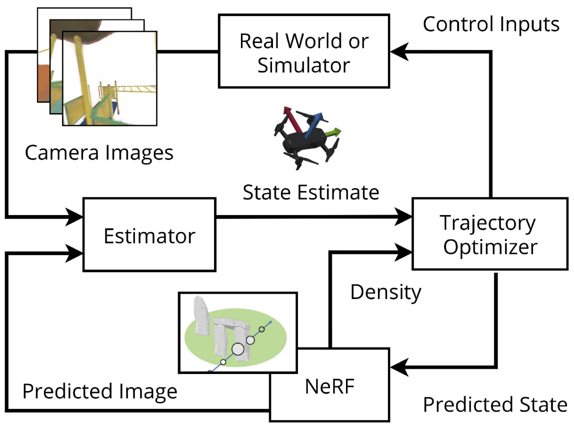


Fig. 2. Block diagram of the proposed pipeline. Our method consists of a trajectory optimizer and state estimator which use a NeRF representation of the environment for planning and localization. At each timestep, the planner optimizes a trajectory from the current mean state estimate which minimizes a NeRF-based collision metric. The robot then applies the first control action of this trajectory, and receives a noisy image from its onboard camera. Finally, the state estimator, using the NeRF as a nonlinear measurement model, uses this image to generate a posterior belief over the new state.

IV. Trajectory Planning in a NeRF

This paper addresses the unique challenges that prevent common trajectory planning methods from working with NeRF environment representations. Querying a NeRF at a point in space gives a density, not an absolute occupancy, which prevents the use of hard constraints and instead suggests a method that seeks to minimize the integrated density over the volume of the robot.

A. Differential Flatness

To speed up planning, our system leverages "differential flatness," a particular property of some dynamical systems which allows their inputs and states to be represented using a (smaller) set of "flat outputs," and their derivatives. Notably, quadrotors are known to be differentially flat, with their position and yaw angle as flat outputs [28].

Traditional planning pipelines for differentially flat systems [29], [30] seek polynomial trajectories for the flat outputs which minimize an objective functional (such as snap or jerk) subject to waypoint constraints. This problem can be expressed as a quadratic program, which can be solved efficiently. Collision avoidance can also be included in this formulation, but in order for the problem to remain convex, the designer must hard-code decisions about how obstacles will be passed.

Our approach differs from the traditional pipeline since we do not describe the obstacles in closed form (e.g., as polytopes), but instead represent them implicitly using the NeRF density. Additionally, while prior methods only optimize the trajectory between static, hand-designed waypoints, our method uses a denser set of waypoints whose location can be optimized directly. Because our trajectory optimization is fundamentally nonconvex, we instead perform our optimization using first-order methods (in particular, the Adam optimizer) with gradients computed efficiently using automatic differentiation. Our decision variables thus are a set of flat output waypoints that we optimize to minimize a combined objective of collision probability and control effort. One advantage of our approach is that the cost can be an arbitrary differentiable functional of the trajectory or robot state; further, our planned trajectory can be naturally combined with differential flatness-based feedback controllers for low-level tracking. Note that while this paper uses quadrotors as an example, this property is known to hold for numerous other vehicle types, such as omnidirectional or differential drive ground robots.

B. Optimization Formulation

The trajectory optimizer seeks a set of flat output waypoints $W = \{\boldsymbol{\sigma}_0, \ldots, \boldsymbol{\sigma}_h\}$ that minimizes multi-objective cost given

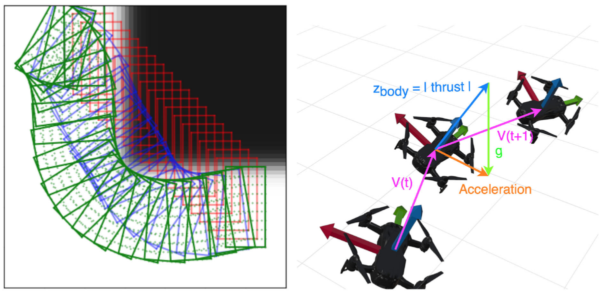


Fig. 3. (a) Overhead view showing the planned trajectory as the optimization progresses on a toy example. The initial trajectory (red) goes straight through the high density regions of the NeRF (black area). The blue is an intermediate trajectory which clips the corner. The final trajectory (green) avoids the obstacle. (b) Visualization of how the force balance on the quadrotor leads to a differential flatness formulation.

by

$$J=\sum_{\tau=0}^{h}\overbrace{\sum_{\mathbf{b}_i\in\mathcal{B}}\rho\left(\mathbf{R}_\tau\mathbf{b}_i+\mathbf{p}(\boldsymbol{\sigma}_\tau)\right)s(\mathbf{b}_i)}^{\text{collision penalty}}+\overbrace{\mathbf{u}_\tau^T\boldsymbol{\Gamma}\mathbf{u}_\tau}^{\text{control penalty}} \qquad (2)$$

where $\mathbf{p}(\boldsymbol{\sigma}_\tau)$ is the position component of a differentially flat state $\boldsymbol{\sigma}_\tau$, $\mathbf{R}_\tau$ is the rotation matrix from the robot body frame to the world frame, $s(\cdot)$ is a function that returns the distance traveled by the point in the robot's point cloud, and $\boldsymbol{\Gamma}$ is the positive definite matrix of of weights penalizing control effort. The first objective seeks to minimize the upper bound on collision probability defined in (1), and the second seeks to minimize control effort. Note that $\mathbf{R}_\tau$, $s(\cdot)$, and $\mathbf{u}_\tau$ are derived from the surrounding waypoints using the robot's dynamics, and therefore are functions of the decision variables $\{\boldsymbol{\sigma}_1,\ldots\boldsymbol{\sigma}_h\}$.

C. Initialization

Our method is initialized by calculating a series of preliminary waypoint poses between the current pose and goal pose via a heuristic, such as a straight line or A* on a coarse grid overlaid on the scene. We optimize these initial guesses via gradient decent to balance multiple objectives such as avoiding collisions, and minimizing control effort. Fig. 3(a) shows a trajectory moving towards areas of low NeRF density as it its optimized from an initial straight line.

V. Vision-Only Pose Filtering in a NeRF

After executing an action from the planned trajectory, the robot must close the loop and estimate its pose using its onboard sensors (e.g. a monocular camera). In this section we address the problem of how a robot can update its pose belief given a measurement and its most recent control action.

Our method is most closely related to [19] where an initial pose estimate is optimized by minimizing the photometric loss between the pixels in the image and the predicted pixels via the projected NeRF scene. However, this method is a single-shot estimator and is highly dependent on the initialization. We formulate a state estimation filter that adds a process loss to the same photometric loss. This additional loss term provides benefits beyond the prior work by estimating a pose and its derivatives. Additionally, the state estimation should be more robust when the robot travels through regions of low photometric gradient information, relying more on the dynamics model. Lastly, the filter produces a state covariance which can be useful for other robotics algorithms running in parallel, such as collision avoidance with dynamic agents [31].

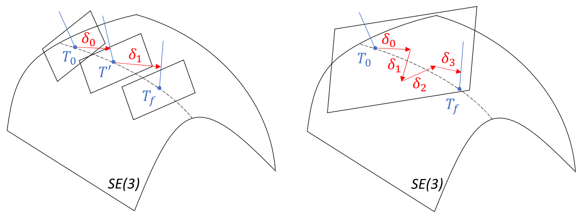


Fig. 4. Recursive SE(3) optimization (left) vs. Optimization in tangent plane (right).

A. Optimization Formulation

At each timestep, the estimator is provided a new image I_t, the previous action taken $\mathbf{u}_t$, and a Gaussian belief over the state $\mathbf{x}_{t-1}$ with mean $\boldsymbol{\mu}_{t-1}$ and covariance $\boldsymbol{\Sigma}_{t-1}$. Using this information, we update the belief as follows:

$$\boldsymbol{\mu}_{t|t-1}=f(\boldsymbol{\mu}_{t-1},\mathbf{u}_t) \qquad (3)$$

$$\mathbf{A}_{t-1}=\left.\frac{\partial f(\mathbf{x},\mathbf{u}_t)}{\partial\mathbf{x}}\right|_{\mathbf{x}=\boldsymbol{\mu}_{t-1}} \qquad (4)$$

$$\boldsymbol{\Sigma}_{t|t-1}=\mathbf{A}_{t-1}\boldsymbol{\Sigma}_{t-1}\mathbf{A}_{t-1}^T+\mathbf{Q}_{t-1} \qquad (5)$$

where the dynamics are modeled as $\mathbf{x}_t=f(\mathbf{x}_{t-1},\mathbf{u}_t)$ with process noise covariance $\mathbf{Q}_t$.

As in [19], a subset of pixels $\mathcal{I}$ are selected for evaluation using existing image feature detectors (e.g. ORB) to identify points of interest and bias the sampling around these areas of higher gradient information. The pose of the robot $\mathbf{T}_t\in SE(3)$ can be constructed from the position and rotation elements of $\boldsymbol{\mu}_t$. With this information, the cost function to be minimized is

$$J(\boldsymbol{\mu}_t)=\overbrace{\|C_{\mathcal{I}}(\mathbf{T}_t)-I_t(\mathcal{I})\|^2_{\mathbf{S}_t^{-1}}}^{\text{photometric loss}}+\overbrace{\|\boldsymbol{\mu}_{t|t-1}-\boldsymbol{\mu}_t\|^2_{\boldsymbol{\Sigma}^{-1}_{t|t-1}}}^{\text{process loss}} \qquad (6)$$

where $\mathbf{S}_t$ is the measurement noise covariance and the notation $\|\mathbf{x}\|^2_{\mathbf{M}}=\mathbf{x}^T\mathbf{M}\mathbf{x}$ is the weighted ℓ_2 norm. Minimizing this equation gives the updated mean $\boldsymbol{\mu}_t$. Outlier rejection is performed on the per-pixel loss to reduce variance.

Finally, we leverage the known relationship between the Hessian of a Gaussian loss function and the covariance [32] to yield the posterior covariance,

$$\boldsymbol{\Sigma}_t=\left(\left.\frac{\partial^2 J(\mathbf{x})}{\partial\mathbf{x}^2}\right|_{\mathbf{x}=\boldsymbol{\mu}_t}\right)^{-1}. \qquad (7)$$

B. Performance Enhancing Optimization Details

The approach in [19] optimizes for the state by taking gradient steps with respect to a reference pose and projecting onto the SE(3) manifold after the optimization to recover the state estimate. Instead, we project back onto the manifold after every gradient step. This is illustrated in Fig. 4. These two methods are mathematically distinct, as the multiplication of

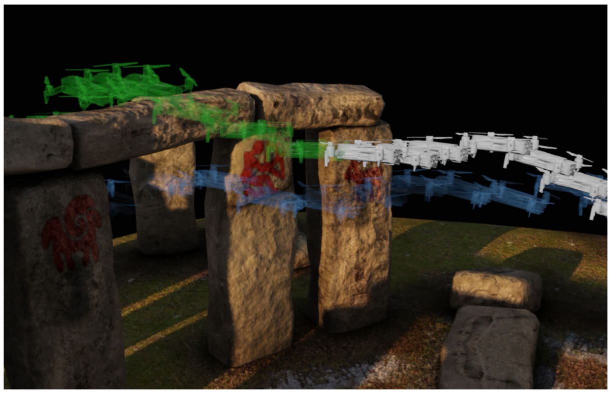


Fig. 5. Results for the proposed trajectory optimizer navigating through Stonehenge. The blue trajectory is the initial plan returned by our optimizer. We then roll out the dynamics noisily for a number of timesteps (white) and re-optimize the trajectory (green). The planner responds to a vertical disturbance by opting to fly above the arch, rather than below as initially planned. Changing homotopy classes around obstacles is quite difficult for existing differential flatness-based planners .

skew-symmetric matrices in the exponential map do not commute. Explicitly,

$$\exp(\boldsymbol{\delta}_1)\exp(\boldsymbol{\delta}_2)\cdots\mathbf{T} \neq \exp(\boldsymbol{\delta}_1+\boldsymbol{\delta}_2+\cdots)\mathbf{T}, \qquad (8)$$

where $\mathbf{T}$ is the reference homogeneous transformation matrix, and $\exp(\cdot)$ is the exponential map between the tangent space and manifold. Qualitatively we observe that the recursive SE(3) gradient descent converges quicker and more smoothly than the method in [19], which we attribute to the noisy photometric loss landscape over the SE(3) manifold. Please see [33] for further Lie theory details. Optimization on the manifold is implemented using the LieTorch library [34].

VI. Online Replanning for Vision-Only Navigation

We combine the trajectory planner from Section IV and the state estimator from Section V in an online replanning formulation. The robot begins with an initial prior of its state belief, a final desired state, as well as the trained NeRF.

The robot first plans a trajectory as described in Section IV. The robot then executes the first action (in this case inside a simulator), and the state filter takes in a new image and updates its belief. The mean of this posterior is used in the trajectory planner as a new starting state and along with the rest of the previous waypoints at a hot start, re-optimizes the trajectory taking into account any disturbances. This continues until the robot has reached the goal state. This process is described in Algorithm 1. This allows the robot to create new updated plans that take into account disturbances. Fig. 5 show an example of a trajectory being reoptimised given new information.

Algorithm 1: Receding Horizon Planner.

```
1: Inputs: $(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$ initial state prior, $\mathbf{x}_{goal}$ desired final
   state, $N$ trained NeRF model of environment.
2: $W \leftarrow \mathtt{A}^*(\boldsymbol{\mu}_0, \mathbf{x}_{goal})$
3: while not at $\mathbf{x}_{goal}$ do
4:   $W \leftarrow \mathtt{trajOpt}(W)$ [Sec. IV]
5:   $\mathbf{x}, \mathbf{u} \leftarrow \mathtt{getStatesActions}(W)$
6:   $I \leftarrow \mathtt{getCameraImage}()$
7:   $\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t \leftarrow \mathtt{poseFilter}(I, \boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}, \mathbf{u}_t)$
     [Sec. V]
8:   $W \leftarrow [\boldsymbol{\mu}_t, W\mathtt{[2:end]}]$
```

VII. Experiments

We demonstrate the performance of our method using a variety of high-fidelity simulated mesh environments (scene meshes by Sketchfab users Ahmad Azizi, artfletch, & ruslans3d). Since our method assumes a trained NeRF model, we first render a sequence of images from the mesh. These images are used to train a NeRF model using an off-the-shelf PyTorch implementation [35]. Rendering images from a mesh with [36] (rather then images taken with a camera in the real world) provides a ground truth reference for the scene geometry with which to evaluate our method. The robot's sensor images are similarly rendered from the ground truth environment, but the trajectories are different so any query of the NeRF model differs from the training data.

The experiment section is divided into 3 parts: We first evaluate the performance of our trajectory planner on its own, then the state estimator on it own, before demonstrating the complete online replanning pipeline.

We first study the performance of our trajectory optimizer alone on a number of benchmark scenes. We demonstrate that our trajectory optimizer can generate aggressive, dynamically feasible trajectories for a quadrotor and an omni-directional robot which avoid collision.

A. Planner - Ground Truth Comparison

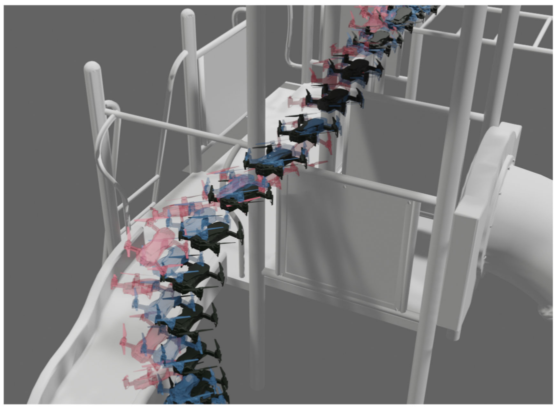
In order to use a NeRF to reason about collisions, we need to show that the learned optical density is a good proxy for real world collisions. We compare the NeRF predicted collisions during various stages of trajectory optimization with the ground truth mesh intersecting volume, during planning of a quadrotor path through a playground environment. The trajectories are shown in Fig. 6(a). Fig. 6(b) shows the relationship between the collision loss term from (2) and overlap between the robot volume and the ground truth mesh over time. In addition to planner finding a smooth, collision-free (i.e., zero mesh intersection volume) trajectory, we can see that throughout training the NeRF density and mesh overlap are closely matched.

B. Planner - Comparison to Prior Work

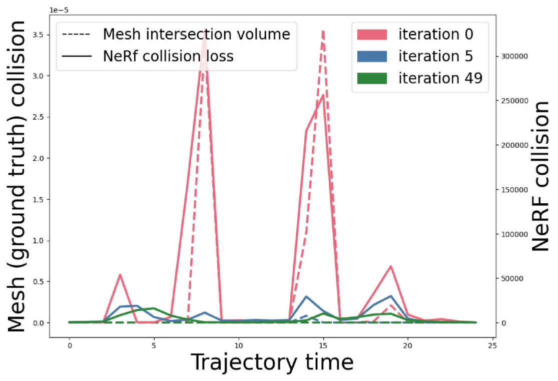
Since this method is, to our knowledge, the first method to operate on NeRFs, direct comparisons are difficult. We compare to two widely used techniques that we have adapted to work on a NeRF environment representation: minimum-snap trajectory planning and Rapidly-exploring Random Trees.

Minimum-snap trajectory planning [29], similarly to our method, uses differential flatness to compute trajectories that pass through a series of waypoints. However, this method typically uses hand-placed waypoints, whereas our method is capable of optimizing the locations of those waypoints based on the NeRF. In this comparison, we generate the waypoints for the minimum-snap planner using the same A* algorithm our method uses for initialization.

Rapidly-exploring Random Trees (RRT) is a sampling based method that generates a space-filling tree used to find a collision free trajectory. Since it requires a binary collision metric, we

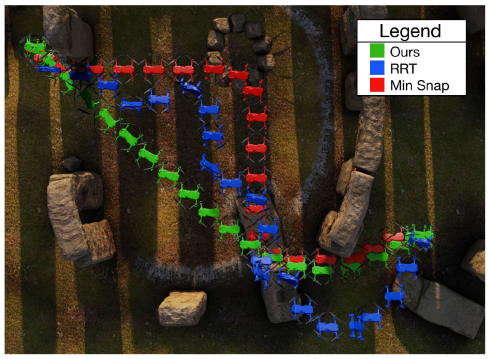


(a)

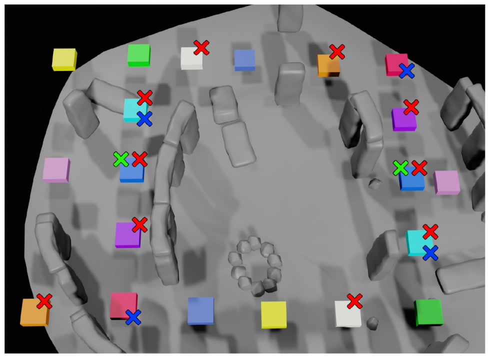


(b)

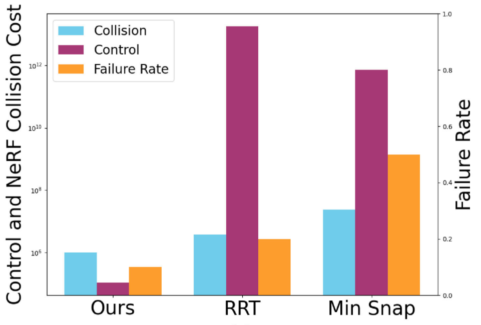
Fig. 6. Results of our proposed trajectory optimizer planning a path through a playground. (a) Visualization of the optimized trajectory generated by our planner. The initialization provided is shown in red and a partially-optimized trajectory in blue. We see the optimizer converge to a trajectory that both avoids collision and is smooth by observation. (b) Plot of the NeRF collision loss (solid lines), and the intersecting volume of the ground-truth meshes (dashed lines). Lower is better. We see the NeRF collision loss is clearly correlated with the intersection volume, showing that minimizing our proposed objective (2) indeed minimizes collision. Note that by iteration 49 optimizer converges to a trajectory that has zero intersections with the ground truth meshes.



(a)



(b)



(c)

Fig. 7. A comparison between our planner and the minimum-snap and RRT baselines. (a) Example trajectories the planners take though the Stonehenge environment. Our planner can move the waypoints to result in a smooth trajectory compared to minimum-snap, which exactly follows the A* initialization. Further, while our planner's trajectory does not collide with the ground truth mesh the minimum-snap trajectory clips the column on the right. While RRT generates a collision-free trajectory, its erratic shape leads to a high control effort. (b) Color-matched start and endpoints of the trajectories along with an indication if they were successful for a given planner (crosses use the same coloring as in (a)). (c) Each planner's mean NeRF collision metric and control effort per time step, averaged over the 20 initializations. We can see that our method yields trajectories with low control effort, low NeRF collision cost, as well as a low failure rate.

first convert the NeRF into a mesh using marching cubes, as in. When generating the RRT, we use a spherical collision model, as we cannot know the robot's orientation at the planning stage, since the RRT only selects positions. Finally, in order to extract the control actions required to follow the RRT trajectory, we use a a differential flatness-based controller [29].

To evaluate performance we run all 3 methods on 10 trajectories with a range of obstacles, speed and complexities inside the Stonehenge environment. Fig. 7(c) shows the mean costs associated with each planner, along with the failure rate (defined as an collision with the ground truth mesh) in the trajectories. Additionally Fig. 7(a) shows the 3 planners' qualitative performance.

C. Planner – Omnidirectional Robot in Tight Space

Our method is not limited to quadrotors, but can handle any robot with differently flat dynamics. Fig. 8 presents an omnidirectional, couch-shaped mobility robot navigating through a narrow space. This scenario presents a difficult kinematic planning problem, commonly called the "piano movers' problem" [37], which requires the robot to turn to fit its body through the narrow gap. The trajectory optimizer, using the proposed NeRF-based collision penalty, is able to generate the desired behavior, which turns the robot to fit through the gap.

D. Estimator - Comparison to Prior Work

We evaluate two methods of estimating the robot state in the NeRF Stonehenge environment. We anticipate that an iNeRF

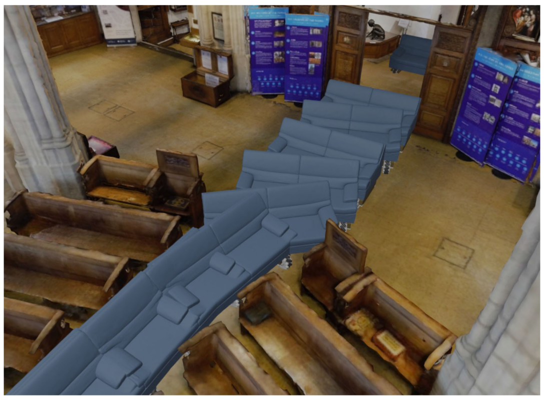


Fig. 8. A planned trajectory for a couch-shaped mobility robot through a narrow gap. The proposed NeRF-based collision penalty results in a trajectory which turns the robot to fit through the gap and avoid collision.

[19] estimator initialized without a dynamics prior will very quickly diverge. Therefore, we propose a dynamically-informed iNeRF estimator as a baseline. The estimate is propagated through the dynamics model to provide an initialization to the estimator at the next time step. Only the photometric loss is optimized. This estimator is identical to a recursive Bayesian filter with infinite state covariance, hence the process loss is set to 0. The second method is our full filter proposed in Section V. We evaluate these methods on a identical set of actions and initial state. For our filter, we assume $\Sigma_0 = 0.1I$, $Q_t = 0.1I$, $S_t = I$, and use $\mathcal{I} = 256$ pixels. Zero-mean Gaussian white noise is added to the true dynamics with standard deviation $2cm$ for the translation and $0.02\,rd$ to the pose angles, while the standard deviation for their rates are half those values. For comparison, the scene area is scaled to be approximately $4m^2$ and the drone is $0.5cm^3$ in volume.

A comparison on the two methods over 100 trials conditioned on the same initial state, set of actions, and noise characteristics is shown in Fig. 9. Our method outperforms the dynamically-informed iNeRF baseline on almost every metric and does not under-perform. We again bring attention to the fact that our filter provides a finite state covariance, which may be useful in determining low-fidelity regions of the NeRF environment.

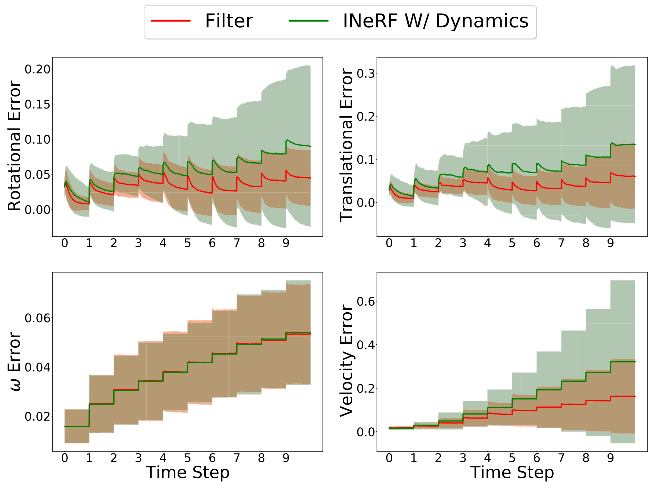


Fig. 9. Error comparison averaged over 100 trials. Rotational errors are the ℓ_2 norm of the angles in axis-angle representation required to rotate the estimated pose to the ground truth. Translational and rate errors are the ℓ_2 norm of the estimated and ground truth difference. Bounds indicate one standard deviation above and below the mean error. Regions between time steps are the gradient steps of the optimization, while spikes at the beginning of time steps indicate the forward propagation of the simulation, a new image observation, and a new optimization. Our method (red) outperforms the dynamically initialized iNeRF (green) in rotational, translational, and velocity estimates while sporting lower variance.

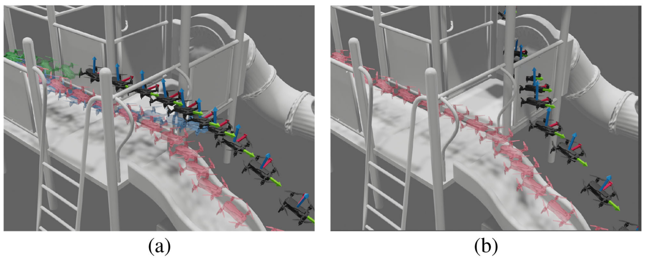


Fig. 10. (a) Quadrotor flight path execution with feedback. The originally planned trajectory is in red. However, when the state estimate deviates significantly from the planned trajectory, the robot re-plans and executes a collision-free path to the goal, as shown by the re-planned trajectories in blue and green. (b) Quadrotor flight path execution without feedback (open-loop). An external disurbance causes the trajectory to deviate from the original plan (red) with catastrophic results.

E. Online Replanning

We evaluate performance of the entire pipeline on planned trajectories in the playground and Stonehenge scenes. The ground truth dynamics are the finite difference drone equations in Section V with the same additive noise as in our estimator experiment. Although the executed trajectories incur a higher cost than the initial plan, the planner is still able to generate collision-free trajectories (Fig. 10(a)) and reach the goal, whereas an open-loop execution (Fig. 10(b)) of the initial planned actions causes collisions and divergence.

F. Performance and Timing

Experiments were run on a computer with an AMD Ryzen 9 5900X @ 3.7 GHz CPU and Nvidia RTX 3090 GPU. Both the trajectory planner and estimator computation time is dependent on number of iterations. Typically, an initial trajectory requires 20 s over 2500 iterations to optimize. In the online replanning experiments ($\Delta t = 0.1s$), subsequent trajectory updates occur in 2 s over 250 iterations. The state estimator typically runs for 4 s over 300 gradient steps, 0.25 s of which is the Hessian computation (7). However, NeRFs are a fast-evolving technique and extensions have seen orders-of-magnitude improvements in performance [14], [38], which we hope to leverage in the future.

Assuming those performance gains apply for our use case, we could expect to be able to run this type of algorithm in real time on a real robot, perhaps aided by off-board compute, in the near future.

VIII. Conclusions

In this work, we proposed a trajectory planner and pose filter that allow robots to harness the advantages of the NeRF representation for collision-free navigation. We presented a new trajectory optimization method based on discrete time differential flatness dynamics, and combined this with a new vision-based state filter to create a full online trajectory planning and replanning pipeline.

Ongoing work seeks to further integrate perception and control in an active planning manner, both by encouraging the trajectories to point the camera in directions with greater gradient information as well as use the uncertainty metrics calculated by the state estimator to reduce collision risk. Another direction for future work includes harnessing improvements in the underlying NeRF representation to improve execution speed [14], since this is the limiting factor for the proposed method.

We also seek to extend this work to utilize multiple NeRFs to represent scenes with movable objects, and explore how various robots such as manipulators could interact with such an environment. Lastly, further work could look to improve the pixel sub-sampling heuristic employed by the state filter. Finally, we would like to implement the proposed method on quadrotors in real scenes to demonstrate the performance beyond simulation.

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 *Proc. Euro. Conf. Comput. Vis.*, 2020, pp. 405–421.

[2] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Learning high-speed flight in the wild," *Sci. Robot.*, vol. 6, no. 59, 2021, Art. no. eabg5810.

[3] A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep drone racing: From simulation to reality with domain randomization," *IEEE Trans. Robot.*, vol. 36, no. 1, pp. 1–14, Feb. 2020.

[4] E. Sucar, S. Liu, J. Ortiz, and A. J. Davison, "iMAP: Implicit mapping and positioning in Real-Time," in *Proc. Int. Conf. Comput. Vision*, 2021. [Online]. Available: https://edgarsucar.github.io/iMAP/

[5] Z. Wang, S. Wu, W. Xie, M. Chen, and V. A. Prisacariu, "NeRF–: Neural radiance fields without known camera parameters," 2021, *arXiv:2102.07064*. [Online]. Available: https://nerfmm.active.vision

[6] "NERF-GTO: Using a neural radiance field to grasp transparent objects," in *Conf. Robot Learn. (CoRL)*, 2020. [Online]. Available: https://sites.google.com/view/dex-nerf

[7] W. Yuan, Z. Lv, T. Schmidt, and S. Lovegrove, "Star: Self-supervised tracking and reconstruction of rigid objects in motion with neural rendering," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 13144–13152.

[8] M. Niemeyer and A. Geiger, "GIRAFFE: Representing scenes as compositional generative neural feature fields," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 11453–11464.

[9] J. Ost, F. Mannan, N. Thuerey, J. Knodt, and F. Heide, "Neural scene graphs for dynamic scenes," in *Proc. IEEE/CVF Conf. Comput. Vision Pattern Recognit. (CVPR)*, Jun. 2021, pp. 2856–2865.

[10] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer, "D-NeRF: Neural radiance fields for dynamic scenes," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 10318–10327.

[11] G. Gafni, J. Thies, M. Zollhofer, and M. Nießner, "Dynamic neural radiance fields for monocular 4D facial avatar reconstruction," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 8649–8658.

[12] M. Tancik *et al.*, "Learned initializations for optimizing coordinate-based neural representations," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 2846–2855.

[13] A. Yu, V. Ye, M. Tancik, and A. Kanazawa, "Pixelnerf: Neural radiance fields from one or few images," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 4578–4587.

[14] S. J. Garbin, M. Kowalski, M. Johnson, J. Shotton, and J. Valentin, "Fastnerf: High-fidelity neural rendering at 200fps," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 14346–14355.

[15] M. Michalkiewicz, J. K. Pontes, D. Jack, M. Baktashmotlagh, and A. Eriksson, "Implicit surface representations as layers in neural networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 4742–4751.

[16] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "DeepSDF: Learning continuous signed distance functions for shape representation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 165–174.

[17] V. Sitzmann, J. N. P. Martel, A. W. Bergman, D. B. Lindell, and G. Wetzstein, "Implicit neural representations with periodic activation functions," in *Proc. Adv. Neural Inform. Process. Syst.*, vol. 33, 2020, pp. 7462–7473.

[18] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, "Occupancy networks: Learning 3D reconstruction in function space," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4460–4470.

[19] 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 *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 1323–1330. [Online]. Available: https://github.com/salykovaa/inerf/

[20] M. Kelly, "An introduction to trajectory optimization: How to do your own direct collocation," *SIAM Rev.*, vol. 59, no. 4, pp. 849–904, Jan. 2017.

[21] J. T. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. 2nd ed., Philadelphia, PA: Society for Industrial and Applied Mathematics, 2010.

[22] R. A. Newcombe *et al.*, "KinectFusion: Real-time dense surface mapping and tracking," in *Proc. 10th IEEE Int. Symp. Mixed Augmented Reality*, 2011, pp. 127–136.

[23] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3D euclidean signed distance fields for on-board MAV planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 1366–1373.

[24] L. Han, F. Gao, B. Zhou, and S. Shen, "FIESTA: Fast incremental euclidean distance fields for online motion planning of aerial robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 4423–4430.

[25] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2009, pp. 489–494.

[26] J. Mainprice, N. Ratliff, and S. Schaal, "Warping the workspace geometry with electric potentials for motion optimization of manipulation tasks," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 3156–3163.

[27] B. Zhou, J. Pan, F. Gao, and S. Shen, "RAPTOR: Robust and perception-aware trajectory replanning for quadrotor fast flight," *IEEE Trans. Robot.*, vol. 37, no. 6, pp. 1992–2009, Dec. 2021.

[28] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 2520–2525.

[29] D. Mellinger, "Trajectory generation and control for quadrotors," Ph.D. dissertation, Univ. Pennsylvania, Mech. Eng. Appl. Mech., 2012.

[30] M. J. V. Nieuwstadt and R. M. Murray, "Real-time trajectory generation for differentially flat systems," *Int. J. Robust Nonlinear Control*, vol. 8, no. 11, pp. 995–1020, 1998.

[31] G. Angeris, K. Shah, and M. Schwager, "Fast reciprocal collision avoidance under measurement uncertainty," in *Proc. Int. Symp. Robot. Res.*, 2019. [Online]. Available: https://arxiv.org/pdf/1905.12875.pdf.

[32] K.-V. Yuen, *Bayesian Methods for Structural Dynamics and Civil Engineering*. Hoboken, NJ, USA: Wiley, 2010, pp. 257–262. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470824566.app1

[33] J. Sola, J. Deray, and D. Atchuthan, "A micro lie theory for state estimation in robotics," 2018. [Online]. Available: https://arxiv.org/pdf/1812.01537.pdf

[34] Z. Teed and J. Deng, "Tangent space backpropagation for 3d transformation groups," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 10338–10347. [Online]. Available: http:github.com/princeton-vl/lietorch

[35] L. Yen-Chen, "Nerf-pytorch," 2020. [Online]. Available: https://github.com/yenchenlin/nerf-pytorch/

[36] A. Szot *et al.*, "Habitat 2.0: Training home assistants to rearrange their habitat," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2021. [Online]. Available: https://arxiv.org/pdf/2106.14405.pdf

[37] J. T. Schwartz and M. Sharir, "On the 'piano movers' problem I the case of a two-dimensional rigid polygonal body moving amidst polygonal barriers," *Commun. Pure Appl. Math.*, vol. 36, no. 3, pp. 345–398, 1983.

[38] C. Sun, M. Sun, and H.-T. Chen, "Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction," *CoRR*, vol. abs/2111.11215, 2021, [Online]. Available: https://arxiv.org/abs/2111.11215