# Vision-Only Robot Navigation in a Neural Radiance World

Michal Adamkiewicz,*[1] Timothy Chen,*[2] Adam Caccavale,[3] Rachel Gardner,[1] Preston Culbertson,[3]
Jeannette Bohg,[1] Mac Schwager[2]

***Abstract*— Neural Radiance Fields (NeRFs) have recently emerged as a powerful paradigm for the representation of natural, complex 3D scenes. 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 on-board 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 the narrow gap. Videos of this work can be found at mikh3x4.github.io/nerf-navigation/.**

## I. Introduction

Planning and executing a trajectory with on-board sensors is a fundamental building block of many robotic applications, from manipulation, to autonomous driving, to autonomous 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 Neural Radiance Fields (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. 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 on-board RGB image, using the image synthesis capabilities of the NeRF.

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.

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 using only an RGB camera for feedback. Existing vision-only navigation systems such as [2] have also advocated for a modularization of learned perception and control and achieved impressive results. Their perception system aims to generalize over variations of drone race tracks and requires very specific training data and labels. We 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 the traditional feature extraction, matching, and alignment pipeline. They inherently represent the geometry as a continuous density field, and so they are well-suited to robot motion planning and trajectory optimization. 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. However, for robots to harness the advantages of the NeRF representation for navigation, we need a trajectory planner and pose filter designed to work specifically with the NeRF machinery.

We address this need in this paper with several contri-

*These authors contributed equally.
[1]Department of Computer Science, Stanford University, Stanford, CA, 94305, USA {mikadam, rachel0, bohg}@cs.stanford.edu
[2]Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305, USA, {chengine, schwager}@stanford.edu
[3]Department of Mechanical Engineering, Stanford University, Stanford, CA 94305, USA, {awc11, pculbertson}@stanford.edu
This work was supported in part by NSF NRI grant 1830402, ONR grant N00014-18-1-2830, Siemens and the Stanford Data Science Initiative. P. Culbertson was supported by a NASA Space Technology Research Fellowship.

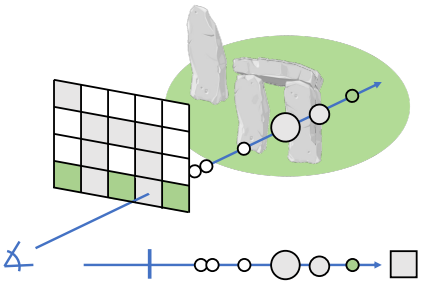


Fig. 2. Schematic of the differentiable rendering process for a NeRF. **Top:** For each pixel in the synthetic image, a ray is cast into the environment. At sample points along the ray, the NeRF is queried for its density (shown by the circle size) and radiance (shown by the circle fill color). **Bottom:** The sampled densities are converted to termination probabilities. The color of the pixel is then calculated as the expected color given these termination probabilities (e.g., this pixel is rendered as gray).

butions. We propose a new trajectory optimization method built from a discrete variation of differential flatness. Unlike typical differential flatness planners, we do not rely on fixed waypoints, and condsider the full robot trajectory, including position and orientation, while imposing penalties to enforce control limits, velocity and acceleration limits, and to avoid collisions with regions of high density in the NeRF. Due to the NeRF environment representation, the resulting trajectory optimization problem is highly non-convex, with some cost terms represented as deep neural networks. We therefore optimize the trajectory using stochastic gradient descent with auto-differentiation (e.g. in PyTorch or TensorFlow), harnessing powerful deep learning tools for our trajectory optimization problem.

Additionally, we propose an optimization based recursive filtering algorithm to estimate the posterior over the full dynamic state of the robot. At each time step, the filter minimizes a loss balancing a photometric error and a dynamics prediction error. The photometric error penalizes the difference between the on board image from the robot, and a synthetic image from the NeRF predicting what the robot should see from its estimated pose. This optimization is again highly nonlinear due to the incorporation of the NeRF in the photometric loss. Hence, we solve it also using stochastic gradient descent. Finally, we combine the trajectory optimization and state filter in an online replanning framework to provide feedback, creating a robust vision-only navigation pipeline.

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 $\theta$, $p$ is a low-dimensional query point such as an $(x, y, z)$ coordinate, and $\sigma$ is some (usually scalar) quantity of interest. Aside from the previously discussed NeRFs, there are several other approaches to implicit representations [3], [4], [5], [6].

However, there currently exists little work studying how to leverage NeRFs for applications beyond novel view synthesis. Recent work [7] 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 [8], 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 [9], 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 [10], 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 positive values inside the obstacle, negative 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 negative at all points on the robot body, along the trajectory. This approach is common in robotic trajectory optimization [11], [12], especially when performing contact-implicit trajectory optimization [13], which seeks to reason about contact geometry when optimizing contact sequences for manipulation or locomotion.

Perhaps closest to this work's trajectory optimizer is CHOMP [14], a gradient-based method which optimizes a finite sequence of poses, with an objective which encourages the trajectory to be smooth and to avoid collision. Obstacle geometry is represented using an SDFs, with a penalty incurred for any states with positive signed distance. Specifically, CHOMP represents obstacle geometry by pre-computing each obstacle's SDF on a finite grid, and approximates SDF gradients using finite differences. In [15], the authors combine a new geometric trajectory optimizer [16] with an extension of this SDF obstacle representation that interpolates between grid cells, thereby observing improved optimization [17]. 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 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 $(\theta, \phi)$ to an emitted color $\mathbf{c} = (r, g, b)$ and scalar density $\rho$. For notational convenience, we define $\rho(\mathbf{p})$ as the density output of the NeRF evaluated at position $\mathbf{p}$ (note $\rho$ 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 3d 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

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

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.

## 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.

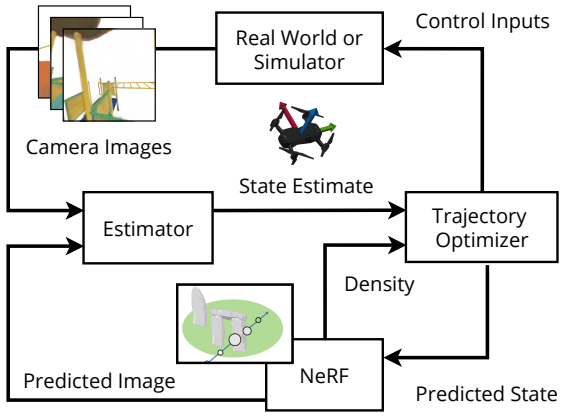



Fig. 3. Block diagram of the proposed pipeline. Our method consists of a trajectory optimizer and state estimator which use a NeRF representation of the environment for planning and localization. At each timestep, the planner optimizes a trajectory from the current mean state estimate which minimizes 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.

### A. Differential flatness

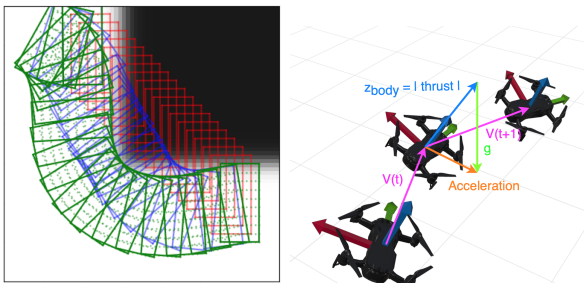
To speed up planning, our system leverages "differential flatness," a particular property of some dynamical systems which allows their inputs and states to be represented using a (smaller) set of "flat outputs," and their derivatives. Notably, quadrotors are known to be differentially flat, with their position and yaw angle as flat outputs [18].

Traditional planning pipelines for differentially flat systems [19], [20] 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. 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 $\{\sigma_0, \ldots, \sigma_h\}$, which we optimize to minimize a combined objective of collision probability and control effort. One advantage of our approach is that costs on the full robot state and inputs can be included in our formulation easily; further, our planned trajectory can be naturally combined with differential flatness-based feedback controllers for low-level tracking. Note that while we describe the dynamics derivation for quadrotors, this same property holds for other vehicle types such as omnidirectional or differential drive ground robots.

### B. Optimization formulation

The trajectory optimizer seeks a set of flat output waypoints $W = \{\sigma_0, \ldots, \sigma_h\}$ that minimizes multi-objective




(a) (b)

Fig. 4. (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

cost given by

$$J(W) = \sum_{\tau=0}^{h} \Bigg[ \overbrace{\sum_{b_i \in \mathcal{B}} \rho(R_\tau b_i + \hat{\sigma}_\tau) s(b_i)}^{\text{collision penalty}} + \overbrace{\gamma u_\tau}^{\text{control penalty}} \Bigg] \quad (2)$$

where $\hat{\sigma}_\tau$ is the position component of a differentially flat state $\sigma_\tau$, $R_\tau$ 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, $\gamma$ is the vector of weights penalizing control effort. The first objective seeks to minimize the probability of collision, as defined in (1), and the second seeks to minimize control effort. Note that $R_\tau$, $s(\cdot)$, and $u$ are derived from the surrounding waypoints using the robot's dynamics, and therefore are functions of the decision variables $\{\sigma_1, \dots \sigma_h\}$.

### 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. 4(a) shows a trajectory moving towards areas of low NeRF density as it its optimized from an initial straight line.

### D. Quadrotor discrete differential flatness

In this subsection we will look at the discrete differential flatness formulation for the quadrotor robot, where we calculate the dependent terms in Eqn. 2 given a set of differentally flat waypoints.

The target velocity $v_\tau$ and acceleration $a_\tau$ are calculated via finite differencing, and Newton's law gives the thrust $F_\tau^z$

$$v_\tau = (\hat{\sigma}_{\tau+1} - \hat{\sigma}_\tau)/dt \quad (3)$$
$$a_\tau = (v_{\tau+1} - v_\tau)/dt \quad (4)$$
$$F_\tau^z = m(a_\tau - g) \quad (5)$$

where $m$ is the mass of the robot.

Next, the full orientation represented as a rotation matrix $R_\tau$ at each waypoint can be determined by

$$\bar{z} = F_\tau^z / \|F_\tau^z\|_2 \quad (6)$$
$$\bar{x} = \bar{z} \times [\sin(\psi_\tau),\ -\cos(\psi_\tau),\ 0]^T \quad (7)$$
$$\bar{y} = \bar{z} \times \bar{x} \quad (8)$$
$$R_\tau = [\bar{x}\ \bar{y}\ \bar{z}]. \quad (9)$$

In the above equations $\psi$ is the yaw angle (the last element of the differentially flat state $\sigma_\tau$), and $\bar{x}$, $\bar{y}$, and $\bar{z}$ are the target principle body directions of the quadrotor.

Finally, the control inputs required to achieve these desired waypoints are calculated as

$$\omega_\tau = Log(R_{\tau+1} R_\tau^T)/dt \quad (10)$$
$$\alpha_\tau = (\omega_{\tau+1} - \omega_\tau)/dt \quad (11)$$
$$u_\tau = [\|F_\tau^z\|_2,\ \bar{J}\alpha_\tau]^T \quad (12)$$

where $Log(\cdot)$ is the logrithmic mapping between the SO(3) manifold and the vector space [21], $\omega_\tau$ is the angular velocity, $\alpha_\tau$ is the angular acceleration, $\bar{J}$ is the robot's moment of inertia, and $u_\tau$ is the control input vector consisting of the thrust and torques.

## 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 [8] 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 first 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 [22].

### A. Interest region sampling

Interest region sampling is a sample-efficient heuristic used in [8] to significantly increase the speed of pose estimation without having to render a full image. The method leverages existing image feature detectors (e.g. SIFT) to identify points of interest and biases the sampling around these areas of higher gradient information.

### B. Optimization formulation

At each timestep, the estimator is provided a new image $I_t$, the previous action taken $u_t$, and a state prior $\mu_{t-1}$ and

covariance $\Sigma_{t-1}$ of the previous time step. We propagate both the uncertainty and estimate as follows:

$$\mu_{t|t-1} = f(\mu_{t-1}, u_t) \tag{13}$$

$$A_{t-1} = \frac{\partial f(x, u_t)}{\partial x}\Big|_{x=\mu_{t-1}} \tag{14}$$

$$\Sigma_{t|t-1} = A_{t-1}\Sigma_{t-1}A_{t-1}^T + Q_{t-1} \tag{15}$$

where the dynamics are modeled as $x_t = f(x_{t-1}, u_t)$ with process noise covariance $Q_t$.

Using interest region sampling, a subset of pixels are selected for evaluation $\mathcal{I}$. The pose of the robot $T_t$ can be constructed from the position and rotation elements of $\mu_t$. With this information, the cost function to be minimized is

$$J(\mu_t) = \overbrace{\|C_{\mathcal{I}}(T_t) - I_t(\mathcal{I})\|^2_{S_t^{-1}}}^{\text{photometric loss}} + \overbrace{\|\mu_{t|t-1} - \mu_t\|^2_{\Sigma^{-1}_{t|t-1}}}^{\text{process loss}} \tag{16}$$

where $S_t$ is the measurement noise covariance and the notation $\|x\|^2_M = x^T M x$ is the weighted $\ell_2$ norm. Minimizing this equation gives the updated mean $\mu_t$. Finally, using the known relationship between the hessian of a Gaussian loss function and the covariance [23], the covariance is

$$\Sigma_t = \left(\frac{\partial^2 J(x)}{\partial x^2}\Big|_{x=\mu_t}\right)^{-1}. \tag{17}$$

### C. Performance enhancing optimization details

To improve estimation performance, we optimize $J(\cdot)$ over a vector of deviations from the predicted state $\mu_{t|t-1}$. 9 of the 18 elements of the quadrotor state vector are the rotation matrix elements representing the orientation of the robot. Since we want to optimize directly along the $SO(3)$ manifold, the rotation delta is a 3 element axis-angle vector denoted $\delta$, which exists in the tangent plane of the $SO(3)$ manifold. After the optimization, the exponential map denoted $\exp(\cdot)$ [21] can be used to update the $SO(3)$ element

$$R_t = R_{t-1}\exp(\delta). \tag{18}$$

## VI. Online Replanning for Vision-Only Navigation

We combine the trajectory planner from Sec. IV and the state estimator from Sec. 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 Sec. IV. The robot then executes the first action (in this case 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.

## VII. Experiments

We demonstrate the performance of our method using a variety of high-fidelity simulated mesh environments [1]. 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 [24]. Rendering images from a mesh with [25] (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.

[1]Scene meshes by Sketchfab users Ahmad Azizi, artfletch, & ruslans3d.

**Algorithm 1** Receding Horizon Planner

Inputs: ($\mu_0$, $\Sigma_0$) initial state prior, $x_{goal}$ desired final state, $N$ trained NeRF model of env.
$W \leftarrow \mathrm{A}^*(\mu_0, x_{goal})$
**while** not at $x_{goal}$ **do**
  $W \leftarrow \texttt{trajOpt}(W)$ [Sec. IV]
  $\mathbf{x}, \mathbf{u} \leftarrow \texttt{getStatesActions}(W)$ [Eqn. 3-12]
  $I \leftarrow \texttt{getImage}()$
  $\mu_t, \Sigma_t \leftarrow \texttt{poseFilter}(I, \mu_{t-1}, \Sigma_{t-1}, \mathbf{u}[0])$ [Sec.V]
  $W \leftarrow [\mu_t, W\texttt{[2:end]}]$

### A. Trajectory Planning

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 omni-directional robot which avoid collision. To evaluate how well the robot avoids collisions with the environment, we use compare the same collision penalty metric used in our loss function in Eqn. 2 with the intersection volume between the robot mesh and the true underlying mesh along the same trajectory.

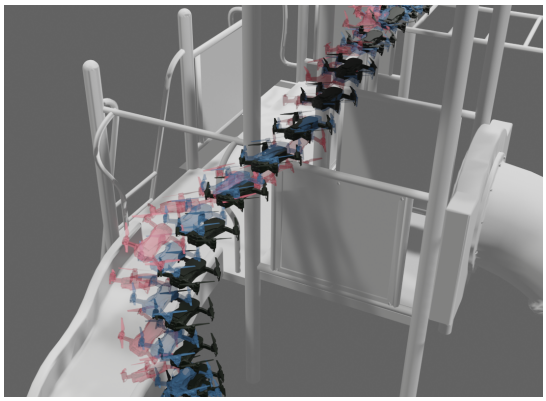
### B. Quadrotor trajectory optimization

A trajectory of a quadrotor through a playground environment is shown in Fig. 5(a). The solid robot figures represent the optimized waypoint poses, while the translucent images show a snapshot of the partially optimized trajectory. Fig. 5(b) shows the relationship between the collision loss term from (2) and overlap between the robot volume and the ground truth mesh. As desired, minimizing the collision loss results in trajectories that do not intersect the mesh. The proposed trajectory optimizer indeed converges to a trajectory which is collision free (zero mesh intersection volume).

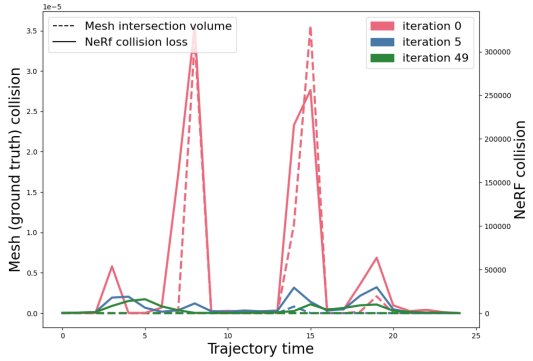
Figure 6 displays the trajectory generated as the quadrotor flies through Stonehenge. The dynamics are rolled out for a number of steps using the original open-loop inputs, with added noise (shown in white), and then re-optimize the trajectory from an intermediate state. Since the drone experienced a vertical disturbance during the dynamics rollout, the re-optimized trajectory converges to a solution which flies over the arch, instead of under, as originally planned. Changing homotopy classes around obstacles is quite difficult for existing differential flatness-based planners

### C. Omnidirectional robot in tight space

Our method isn't limited to quadrotors, but can handle any robot with differently flat dynamics. Figure 7 presents an omnidirectional robot (in this case a mobility device shaped like a sofa) navigating through a narrow space. The planner is aware of the robots shape and turns it sideways to fit through the gap. This is an example of the piano movers' problem [26]



(a)




(b)

Fig. 5. 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. Further, the optimizer converges to a trajectory which is collision free for the ground truth meshes.

### D. State Estimation

We evaluate three methods of estimating the robot state in the NeRF playground environment. The first is iNERF [8], which is the most closely related prior work. It uses the initial state as an initialization for every time step. We improve upon this method by combining it with a dynamics predicting initialization. While not a Bayesian filter, this method initializes by predicting the robot's motion and is more likely to converge to the correct state estimate. Finally, we present our full filter as described in Sec. V. We evaluate these methods by first executing a known trajectory to get a set of ground truth waypoints and actions. We assume $\Sigma_0 = 0.1I$, $Q_t = 0.01I$, $S_t = 0.03I$, and use $\mathcal{I} = 1024$ pixels. Gaussian white noise is added to the true dynamics with standard deviation $1cm$ for the translation, and $0.01rad$ to the angles. For comparison, scene area is scaled to be approximately $4m^2$ and the drone is $0.5cm^3$ in volume.

The results of running the three state estimator are shown in Fig. 8. Unsurprisingly, the vanilla implementation of iNERF is unable to compensate for the motion. Both the Bayesian filter and dynamics predicting initialization implementation of iNeRF perform similarly, which suggests a high-fidelity representation of the ground truth by the NeRF. The filter and dynamic iNeRF achieved a mean translational and rotational error of 0.004 and 0.005, respectively, while iNeRF averaged 0.7 and 0.3 for the translation and rotation. Out of the three, our method is the only one that can provide a Bayesian estimate on the velocity and angular rates.

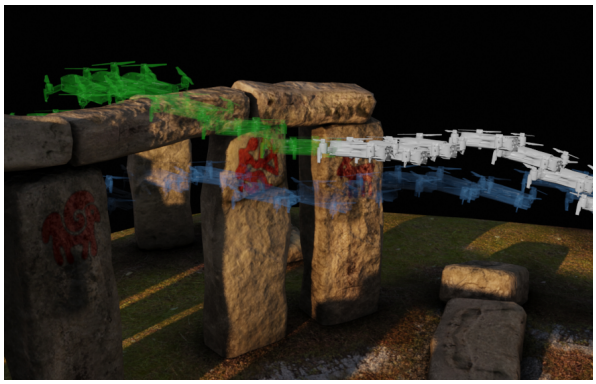


Fig. 6. Results for the proposed trajectory optimizer navigating through Stonehenge. The blue trajectory is the initial plan returned by our optimizer. We then roll out the dynamics noisily for a number of timesteps (white) and re-optimize the trajectory (green). We can see the planner respond to a vertical disturbance by opting to fly above the arch, rather than below as initially planned.

Fig. 7. The planner understands how to plan with a robot shaped like a sofa.

### E. Model Predictive Control

We evaluate performance of the entire pipeline on planned trajectories in the playground and Stonehenge scenes. The ground truth dynamics are the finite differencing drone equations in Sec. 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 and reach the goal, whereas an open-loop execution of the initial planned actions causes collisions and divergence.

## 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

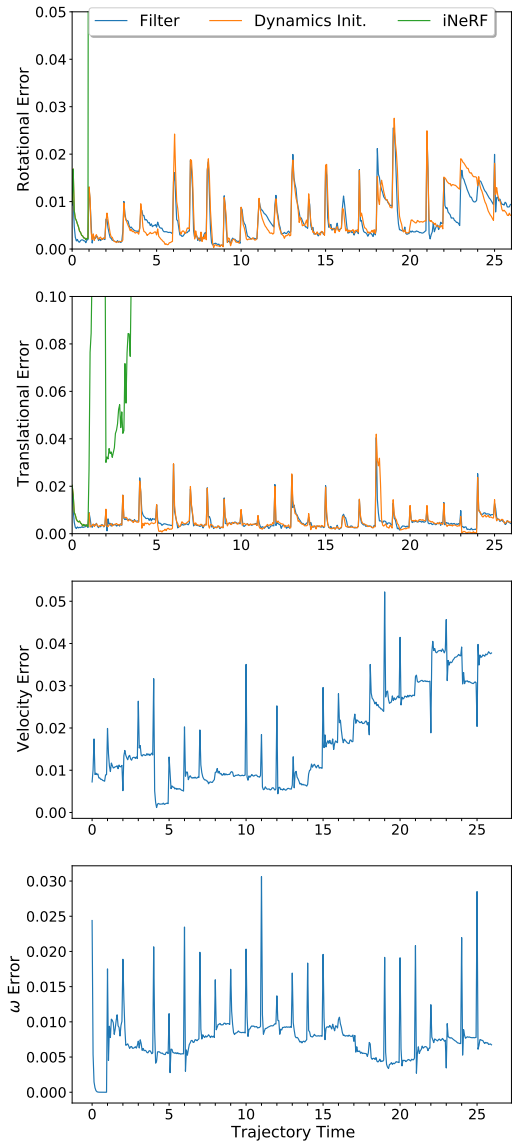



Fig. 8. Error comparison across three state estimators. Spikes are correlated with a new time step. Rotational errors are the angle in axis-angle representation required to rotate the estimated pose to the ground truth. Translational and rate errors are the $\ell_2$ norm of the estimated and ground truth difference. Our method performs similarly to the dynamically initialized iNeRF, but far outperforms vanilla iNeRF, and is also able to return a Bayesian estimate of the derivatives.

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 [27], since this is the limiting factor for the proposed method. Similarly, 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 *European conference on computer vision*. Springer, 2020, pp. 405–421.

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

[3] M. Michalkiewicz, J. K. Pontes, D. Jack, M. Baktashmotlagh, and A. Eriksson, "Implicit Surface Representations As Layers in Neural Networks," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. Seoul, Korea (South): IEEE, Oct. 2019, pp. 4742–4751.

[4] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, "Occupancy Networks: Learning 3D Reconstruction in Function Space," *arXiv:1812.03828 [cs]*, Apr. 2019, comment: To be presented at CVPR 2019. Supplementary material and code is available at http://avg.is.tuebingen.mpg.de/publications/occupancy-networks.

[5] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation," Jan. 2019.

[6] V. Sitzmann, J. N. P. Martel, A. W. Bergman, D. B. Lindell, and G. Wetzstein, "Implicit Neural Representations with Periodic Activation Functions," *arXiv:2006.09661 [cs, eess]*, Jun. 2020.

[7] E. Sucar, S. Liu, J. Ortiz, and A. J. Davison, "iMAP: Implicit Mapping and Positioning in Real-Time," *arXiv:2103.12352 [cs]*, Mar. 2021.

[8] 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 *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021. [Online]. Available: https://github.com/salykovaa/inerf/

[9] M. Kelly, "An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation," *SIAM Review*, vol. 59, no. 4, pp. 849–904, Jan. 2017.

[10] J. T. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming, Second Edition*, 2nd ed. Society for Industrial and Applied Mathematics, 2010.

[11] T. A. Howell, B. E. Jackson, and Z. Manchester, "ALTRO: A Fast Solver for Constrained Trajectory Optimization," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Macau, China: IEEE, Nov. 2019, pp. 7674–7679.

[12] R. Bonalli, A. Cauligi, A. Bylard, and M. Pavone, "GuSTO: Guaranteed Sequential Trajectory optimization via Sequential Convex Programming," in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 6741–6747.

[13] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 69–81, Jan. 2014.

[14] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *2009 IEEE International Conference on Robotics and Automation*, May 2009, pp. 489–494.

[15] D. Kappler, F. Meier, J. Issac, J. Mainprice, C. G. Cifuentes, M. Wüthrich, V. Berenz, S. Schaal, N. Ratliff, and J. Bohg, "Real-Time Perception Meets Reactive Motion Generation," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1864–1871, Jul. 2018.

[16] N. Ratliff, M. Toussaint, and S. Schaal, "Understanding the geometry of workspace obstacles in motion optimization," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 4202–4209.

[17] J. Mainprice, N. Ratliff, and S. Schaal, "Warping the workspace geometry with electric potentials for motion optimization of manipulation tasks," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2016, pp. 3156–3163.

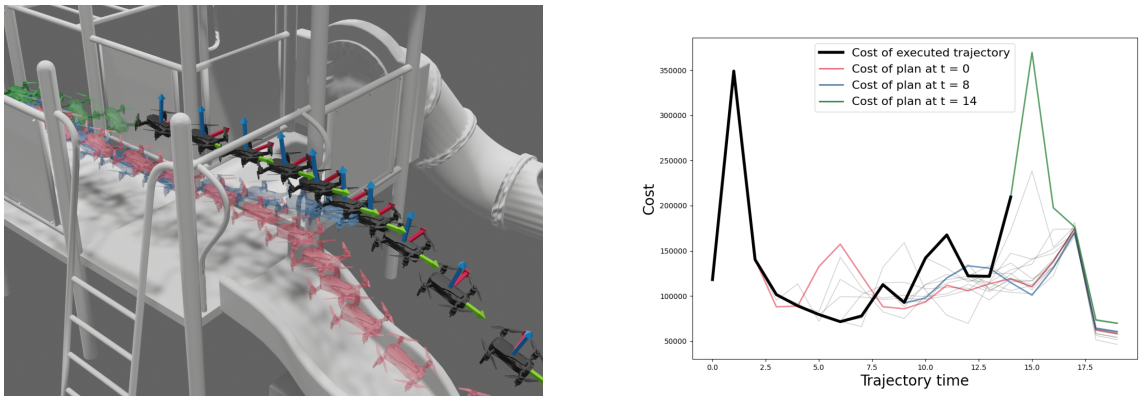
[18] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 2520–2525.

[19] D. Mellinger, "Trajectory Generation and Control for Quadrotors," Ph.D. dissertation, University of Pennsylvania, 2012.

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

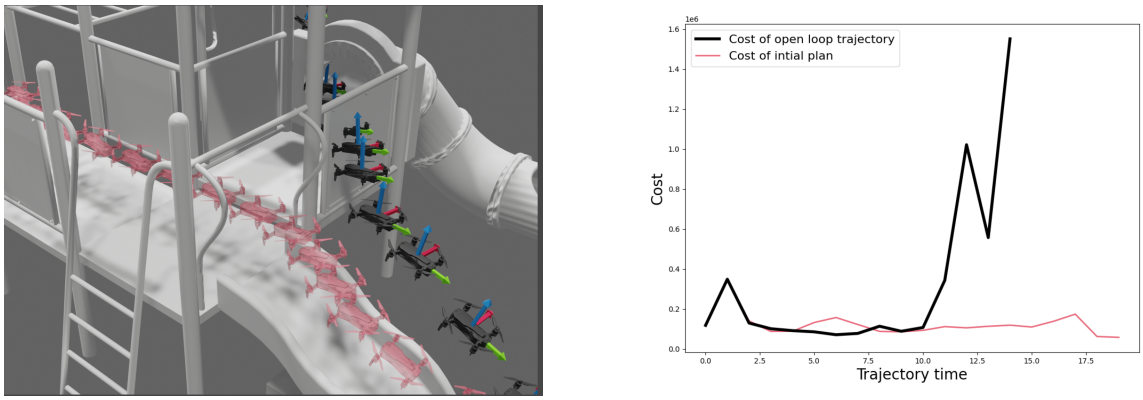
[21] J. Sola, J. Deray, and D. Atchuthan, "A micro lie theory for state estimation in robotics," *https://arxiv.org/pdf/1812.01537.pdf*, 2018.

[22] G. Angeris, K. Shah, and M. Schwager, "Fast Reciprocal Collision Avoidance Under Measurement Uncertainty," in *2019 International Symposium of Robotics Research (ISRR)*, Hanoi, Vietnam, 2019.



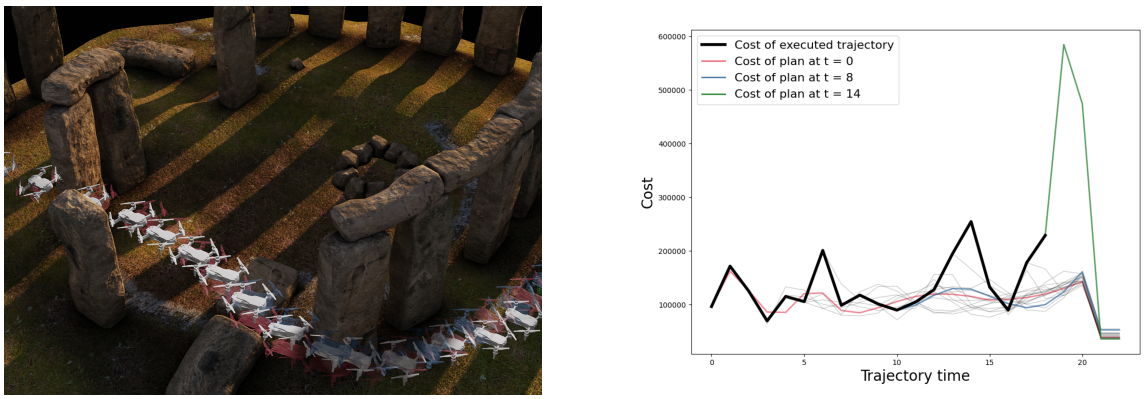

(a) (b)

Fig. 9. Quadrotor flight path execution with feedback. The originally planned trajectory is in red. However, once the system realises it has been blown of course, it corrects and continues towards its goal as shown by the re-planned trajectories in blue and green.




(a) (b)

Fig. 10. Quadrotor flight path execution without feedback (openloop). Noise makes the trajectory deviate from the red plan with catastrophic results.




(a) (b)

Fig. 11. Quadrotor flight path executed in a realistic outdoor environment with feedback. The final high cost (in green) is due to noise pushing the quadrotor downward and it needing a lot of thrust to reach the final state at the last moment.

[23] K.-V. Yuen, *Bayesian Methods for Structural Dynamics and Civil Engineering*. John Wiley & Sons, Ltd, 2010, ch. Appendix A: Relationship between the Hessian and Covariance Matrix for Gaussian Random Variables, pp. 257–262. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470824566.app1

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

[25] A. Szot, A. Clegg, E. Undersander, E. Wijmans, Y. Zhao, J. Turner, N. Maestre, M. Mukadam, D. Chaplot, O. Maksymets, A. Gokaslan, V. Vondrus, S. Dharur, F. Meier, W. Galuba, A. Chang, Z. Kira, V. Koltun, J. Malik, M. Savva, and D. Batra, "Habitat 2.0: Training home assistants to rearrange their habitat," *arXiv preprint arXiv:2106.14405*, 2021.

[26] 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," *Communications on Pure and Applied Mathematics*, vol. 36, no. 3, pp. 345–398, 1983.

[27] S. J. Garbin, M. Kowalski, M. Johnson, J. Shotton, and J. Valentin, "Fastnerf: High-fidelity neural rendering at 200fps," *arXiv preprint arXiv:2103.10380*, 2021.