# Towards Open World NeRF-Based SLAM

Daniil Lisus[1] and Connor Holmes[1]

***Abstract*— Neural Radiance Fields (NeRFs) have taken the machine vision and robotics perception communities by storm and are starting to be applied in robotics applications. NeRFs offer versatility and robustness in map representations for Simultaneous Localization and Mapping. However, computational difficulties of multilayer perceptrons (MLP) have lead to reductions in robustness in the state-of-the-art of NeRF-based SLAM algorithms in order to meet real-time requirements. In this report, we seek to improve accuracy and robustness of NICE-SLAM, a recent NeRF-based SLAM algorithm, by accounting for depth measurement uncertainty and using IMU measurements. Additionally, extend this algorithm by providing a model that can represent backgrounds that are too distant to be modeled by NeRF.**

## I. INTRODUCTION

Starting with the landmark paper by Mildenhall et al. just over two years ago [1], Neural Radiance Fields (NeRFs) have taken the machine vision and robotics perception communities by storm. The central idea behind NeRF is to combine classical graphics rendering techniques with a multilayer perceptron (MLP) trained on image data to learn an *implicit* representation of a given scene. The scene can then be rendered from novel viewpoints (i.e., view synthesis). Within the context of robotics, this approach holds promise to address shortcomings of classical dense SLAM algorithms. In particular, a NeRF is well suited to estimate portions of the map for unobserved regions [2]. Additionally, they can be leveraged to view maps from perspectives that may be interesting to users, but which have not been directly visited by the robot. Since they are fundamentally based on MLPs, NeRF maps can also be trained to be robust to changes in map environment conditions such as lighting [3] or time of the year.

The original formulation of NeRF, which used one large MLP, required hours of training, was slow to render, and required exact knowledge of input camera poses. However, a flurry of advancements since have considerably improved on all of these issues. Several authors have shown that encoding the NeRF in a spatial data structure leads to considerable improvements in speed and accuracy, often using smaller MLPs to decoding spatial features during view synthesis [4]–[6]. In particular, NGLOD [4] proposed the use of small MLPs in a volumetric grid while Plenoxels [5] used a spatial octree and bypassed MLP use altogether. These ideas were further improved upon in [6] by using spatial hash tables to attain NeRFs that can be trained in real-time. BARF [7] and NeRF- - [8] have also shown that a priori exact pose knowledge is unnecessary to reproduce both accurate pose estimates and NeRFs.

These advancements have opened the door for the use of NeRFs to represent maps for Simultaneous Localization and Mapping (SLAM) in robotics applications. One of the first papers in this area was iMAP [9], which builds an MLP-based NeRF in real-time using RGB-D data by cleverly downsampling the number of pixels used to generate the NeRF. NICE-SLAM improved significantly on this framework by leveraging the key insight that the *entire* NeRF map need not be updated at every iteration. By introducing a spatial, voxel-based NeRF, NICE-SLAM only updates parts of the NeRF that are specially relevent to a given camera view. At the start of this project, NICE-SLAM represented the current state-of-the-art in the field of NeRF-based SLAM[1].

Although NICE-SLAM produces moderately robust and complete dense maps of the environment, it fails to perform competitively in the generated pose estimates compared to classical SLAM approaches. Additionally, the produced volumetric grid can become very expensive to maintain if the algorithm aims to function in a large environment. Looking forward to potential open world deployment of NeRF-based SLAM, all current approaches make use of a predefined, finite volumetric grid, without a clear way to handle visual information contained far from the camera or to dynamically expand the area of operation.

This project begins to tackle these problems in order to leverage the special mapping approach of NeRF-based SLAM without sacrificing the quality of the state estimates or requiring excessive computational resources. More specifically, this project improves on the robustness of NICE-SLAM by considering depth uncertainty from RGB-D images and by including IMU information to constrain the camera pose estimates. Additionally, the project begins to expand the capabilities of NICE-SLAM towards the open world by splitting visual information into a foreground and background. The background, modelled as a sphere infinitely far away from the foreground, is then able to contribute to the colour information of the map without requiring an explicit grid to be generated all the way to the visual range of the image.

## II. BASELINE ALGORITHM

Similar to its predecessor, iMAP, and other modern SLAM algorithms [11], NICE-SLAM separates the phases of SLAM into to two parallel threads: a *tracking* thread to localize the

[1]Authors are associated with the University of Toronto Robotics Institute, University of Toronto, Canada. Authors contributed equally to this work. [FIRSTNAME].[LASTNAME]@mail.utoronto.ca

[1]NeRFs and NeRF-based SLAM are currently rapidly evolving fields. NeRF-SLAM [10] has further developed the concepts presented in NICE-SLAM and extended them to monocular SLAM

current camera frame against the current map and a *mapping* thread which jointly optimizes the parameters of the NeRF map and a set of stored *keyframes*.

NICE-SLAM uses a series of 3 fixed-size, voxel grids of encoded features with different grid resolutions to represent the map. When evaluating pixels for a given RGB-D image, NICE-SLAM uses the same ray-casting technique as [1], but evaluates the rays by interpolating sample points the voxel grid (similar to [5]), decoding each sample feature with a (smaller) MLP and aggregating the result into an estimated pixel color and depth. Additional detail on mapping and tracking threads is given in the subsections below.

### A. Mapping

The mapping thread is responsible for updating the voxel-grid NeRF representation of the map. It does this by continually optimizing the voxel-grid features and the stored poses of relevant subset of *keyframes*. Similar to ORB-SLAM, keyframes are selected on the basis of the level of information gain that they provide and consist of an RGB-D image as well as the corresponding estimate of camera pose. For each map update, a set of keyframes is selected based on predicted *overlap* with the current frame and are used, along with the current frame, to construct a loss function.

The loss function has both depth-based and colour-based components. The depth-based loss for each grid resolution in the mapping thread based on $N$ pixels is given by

$$\mathcal{L}_{\text{depth}}^{\text{map}} = \sum_{r=f,c} \frac{1}{N} \sum_{n=1}^{N} \left\| d_n - \hat{d}_{r,n} \right\|_1, \tag{1}$$

where $d_n$ is the measured pixel depth and $\hat{d}_{r,n}$ is the NeRF-predicted pixel depth for the fine ($f$) and coarse ($c$) grid resolutions.

The color-based loss is only generated at the fine grid resolution and is given by

$$\mathcal{L}_{\text{colour}}^{\text{map}} = \frac{1}{N} \sum_{n=1}^{N} \left\| I_n - \hat{I}_n \right\|_1, \tag{2}$$

where $I_n$ is the measured pixel colour and $\hat{I}_n$ is the NeRF-predicted pixel colour.

The overall optimization for the mapping portion is given by:

$$\min_{\boldsymbol{C}_i, \boldsymbol{r}_i, \boldsymbol{\Theta}} \quad \sum_{i=1}^{M} \mathcal{L}_{\text{depth}}^{\text{map,i}} + \lambda_m \mathcal{L}_{\text{colour}}^{\text{map,i}}, \tag{3}$$

where $i$ represents the $i^{th}$ keyframe of $M$ frames, $\boldsymbol{\Theta}$ represents the voxel grid features, and $\lambda_m$ is a tuning parameter.

### B. Tracking

The tracking thread performs a similar optimization to the mapping thread, but only optimizes the pose of the *current* frame (i.e. does not modify the map).

The colour component of the loss, $\mathcal{L}_{\text{colour}}^{\text{track}}$ is computed in the same way as for the mapping, but is only for the current frame. On the other hand, the depth-based loss in the tracking thread is weighted based on depth uncertainty in the network. For each grid resolution, the loss is computed for $N$ pixels as follows:

$$\mathcal{L}_{\text{depth}}^{\text{track}} = \sum_{r=f,c} \frac{1}{N} \sum_{n=1}^{N} \frac{\left\| d_n - \hat{d}_{r,n} \right\|}{\hat{\sigma}_{r,n}^{d}}, \tag{4}$$

where variables are defined in the same way as for (1) with $\hat{\sigma}_{r,n}^{d}$ corresponding to the standard deviation on $\hat{d}_n$. This standard deviation is extracted from the NeRF voxel grid by computing the variance from all points that contribute to the final $\hat{d}_n$ value along a pixel ray.

The overall tracking optimization is given by:

$$\min_{\boldsymbol{C}, \boldsymbol{r}} \quad \mathcal{L}_{\text{depth}}^{\text{track}} + \lambda_t \mathcal{L}_{\text{colour}}^{\text{track}}, \tag{5}$$

where $\boldsymbol{C}$ and $\boldsymbol{r}$ are the pose parameters for the current frame and $\lambda_t$ is a tuning parameter.

## III. IMPROVEMENTS

### A. Depth Uncertainty

The standard NICE-SLAM algorithm does not consider depth uncertainty from the measured RGB-D depth. As a result, when considering $N$ pixels, each pixel contributed equally to the computed depth loss in (1) and (4). While (4) does down-weight the contributions according to a proxy measure of the uncertainty on the NeRF-generated depth value, it still treats the RGB-D depth values as equally valid. This equal valuation is contrary to not only potential sensor-specific uncertainty fluctuations or biases, for example due to artifacts such as vignetting, but also to the fact that most depth sensors measurements increase in uncertainty as a function of the depth. Although this uncertainty negligable in relatively small and constant depth environments, it can become considerable in a large environment.

In order to account for this potentially varying uncertainty, a modification to the depth loss in (1) and (4) is proposed as follows:

$$\mathcal{L}_{\text{depth}}^{\text{map}} = \frac{1}{N} \sum_{n=1}^{N} \frac{\left\| d_n - \hat{d}_n \right\|_1}{\sigma_n^d}, \tag{6}$$

$$\mathcal{L}_{\text{depth}}^{\text{track}} = \frac{1}{N} \sum_{n=1}^{N} \frac{\left\| d_n - \hat{d}_n \right\|_1}{\sqrt{\sigma_n^d + \hat{\sigma}_n^d}}, \tag{7}$$

where $\sigma_n^d$ is the standard deviation of $d_n$, the depth measured in pixel $n$.

### B. Including IMU Data

Motion data can serve as a valuable source of information for state estimation. In this report, we include this data in both the tracking and mapping optimization to make use of temporal relationship between camera poses.

*1) $SO(3)$ Preliminaries:* The camera orientation in this report is represented by the $SO(3)$ matrix Lie group (MLG). For brevity, overall MLG theory is not covered in this report, with a general reference to everything found in [12]. However, this section deals with various $SO(3)$ operators, which need to be defined. The $\log(\cdot)$ and $\exp(\cdot)$ logarithmic and exponential operators respectively convert $SO(3)$ element to and from the Lie algebra. The $(\cdot)^\vee$ and $(\cdot)^\wedge$ vee and wedge operators convert elements of the Lie algebra respectively to and from the $\mathbb{R}^3$ representation, for the 3 degrees of freedom of $SO(3)$. As a shorthand, the capital logarithmic and exponential operators are defined as

$$\mathbf{C} = \exp(\boldsymbol{\xi}^\wedge) \triangleq \mathrm{Exp}(\boldsymbol{\xi}) \in SO(3), \tag{8}$$

$$\boldsymbol{\xi} = \log(\mathbf{C})^\vee \triangleq \mathrm{Log}(\mathbf{C}) \in \mathbb{R}^3. \tag{9}$$

Additionally, the right-invariant error definition for $SO(3)$ is adopted here arbitrarily. With this definition, the difference between a nominal $\bar{\mathbf{C}}$ and $\mathbf{C}$ is computed as $\delta\mathbf{C} = \bar{\mathbf{C}}\mathbf{C}^\mathsf{T}$.

*2) IMU Modelling:* Since IMU data is not considered in NICE-SLAM, it is not included with any of the provided datasets. Therefore, we generate IMU measurements from the provided ground truth poses. Specifically, linear and angular velocities are generated based on a fixed timestamp and corrupted with white Gaussian noise. In this project, "IMU" measurements refer to linear and angular velocity measurements. This is done in order to simplify derivations, since camera velocity estimates are not included, while still evaluating the benefit of motion data.

A linear velocity measurements $\mathbf{v}_k$ at time $t_k$ is generated from ground truth camera poses according to

$$\mathbf{v}_k = \bar{\mathbf{v}}_k + \mathbf{w}_k^\mathbf{v} = \mathbf{C}_k^\mathsf{T}\left(\frac{\mathbf{r}_{k+1} - \mathbf{r}_k}{T_k}\right) + \mathbf{w}_k^\mathbf{v}, \tag{10}$$

where $\bar{\mathbf{v}}_k$ is the true linear velocity, $\mathbf{C}_k \in SO(3)$ is the camera orientation at $t_k$, $\mathbf{r}_\imath \in \mathbb{R}^3$ is the camera position at $t_\imath, \imath \in [k, k+1]$, $T_k = t_{k+1} - t_k$ is the time increment, and $\mathbf{w}_k^\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_k^\mathbf{v})$ is the measurement noise with covariance $\boldsymbol{\Sigma}_k^\mathbf{v}$. Note, this model assumes that the velocity measurement is constant over $T_k$.

An angular velocity measurement $\mathbf{u}_k$ at time $t_k$ is generated from ground truth camera poses according to

$$\mathbf{u}_k = \bar{\mathbf{u}}_k + \mathbf{w}_k^\mathbf{u} = \frac{\mathrm{Log}(\mathbf{C}_k^\mathsf{T}\mathbf{C}_{k+1})}{T_k} + \mathbf{w}_k^\mathbf{u}, \tag{11}$$

where $\bar{\mathbf{u}}_k$ is the true angular velocity, $\mathbf{w}_k^\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \Sigma_k^\mathbf{u})$ is the measurement noise with covariance $\Sigma_k^\mathbf{u}$, and the assumption that the measurement is constant over $T_k$ is again made.

*3) Tracking with IMU:* We add IMU-based loss to the tracking optimization that provides a motion prior for the current camera pose based on its immediate predecessor. Since there is no uncertainty quantification in NICE-SLAM, the previous camera pose is treated as fixed, with the IMU-based loss being weighted by a factor corresponding to the IMU noise.

Consider the current camera pose $\mathbf{X}_k = (\mathbf{C}_k, \mathbf{r}_k)$ as being at time $t_k$ and the previous camera pose $\mathbf{X}_{k-1} = (\mathbf{C}_{k-1}, \mathbf{r}_{k-1})$ as being at time $t_{k-1}$. The IMU loss is composed of an orientation and position component.

The orientation component is computed as

$$\mathbf{e}_\mathrm{IMU}^\mathbf{C} = \mathrm{Log}\left(\mathrm{Exp}(\mathbf{u}_{k-1}T_{k-1})\mathbf{C}_k^\mathsf{T}\mathbf{C}_{k-1}\right). \tag{12}$$

For the final loss, this loss is weighted by the uncertainty from $\mathbf{u}_{k-1}$. This weight is computed by linearizing $\mathbf{e}_\mathrm{IMU}^\mathbf{C}$ with respect to $\mathbf{u}_{k-1}$ and propagating $\boldsymbol{\Sigma}_{k-1}^\mathbf{u}$ through the linearization as

$$\boldsymbol{\Sigma}_\mathrm{IMU}^\mathbf{C} = \left(\frac{\partial \mathbf{e}_\mathrm{IMU}^\mathbf{C}}{\partial \mathbf{w}_{k-1}^\mathbf{u}}\right)\boldsymbol{\Sigma}_{k-1}^\mathbf{u}\left(\frac{\partial \mathbf{e}_\mathrm{IMU}^\mathbf{C}}{\partial \mathbf{w}_{k-1}^\mathbf{u}}\right)^\mathsf{T}, \tag{13}$$

where $\mathbf{w}_{k-1}^\mathbf{u}$ enters $\mathbf{e}_\mathrm{IMU}^\mathbf{C}$ through (11). The final Jacobian, with the error $\mathbf{u} = \bar{\mathbf{u}} - \delta\mathbf{u}$ and the full derivation omitted for brevity, is

$$\frac{\partial \mathbf{e}_\mathrm{IMU}^\mathbf{C}}{\partial \mathbf{u}_{k-1}} = -\mathbf{J}_\ell^{-1}\left(\mathbf{e}_\mathrm{IMU}^\mathbf{C}\right)\mathbf{J}_\ell(\mathbf{w}_{k-1}^\mathbf{u}T_{k-1})T_{k-1}, \tag{14}$$

where $\mathbf{J}_\ell$ is the left group Jacobian of $SO(3)$.

The position component is computed as

$$\mathbf{e}_\mathrm{IMU}^\mathbf{r} = \mathbf{v}_{k-1} - \mathbf{C}_{k-1}^\mathsf{T}\left(\frac{\mathbf{r}_k - \mathbf{r}_{k-1}}{T_{k-1}}\right), \tag{15}$$

with a weighting resulting from $\mathbf{v}_{k-1}$ being $\boldsymbol{\Sigma}_\mathrm{IMU}^\mathbf{r} = \boldsymbol{\Sigma}_{k-1}^\mathbf{v}$.

The final IMU tracking loss is then

$$\mathcal{L}_\mathrm{IMU}^\mathrm{track} = \mathbf{e}_\mathrm{IMU}^{\mathbf{C}^\mathsf{T}}\boldsymbol{\Sigma}_\mathrm{IMU}^{\mathbf{C}^{-1}}\mathbf{e}_\mathrm{IMU}^\mathbf{C} + \mathbf{e}_\mathrm{IMU}^{\mathbf{r}^\mathsf{T}}\boldsymbol{\Sigma}_\mathrm{IMU}^{\mathbf{r}^{-1}}\mathbf{e}_\mathrm{IMU}^\mathbf{r}. \tag{16}$$

*4) Mapping with IMU:* In order to make use of IMU data in the mapping optimization, IMU preintegration is used. Introduced in [13], IMU preintegration allows to group multiple IMU measurements into a relative motion increment (RMI). The RMI then functions as a single "measurement", defining a probabilistic motion constraint between two poses separated by any number of IMU measurements. Since the keyframe used for each mapping step are not predetermined a potentially different number of IMU measurements can be connected to each consecutive keyframe. IMU preintegration allows to greatly simplify the computation the corresponding motion constraints between these keyframes, preventing the need to keep track of every IMU measurement received.

The RMIs are computed incrementally between keyframes and are saved whenever a new keyframe is added. Consider the current camera pose $\mathbf{X}_k = (\mathbf{C}_k, \mathbf{r}_k)$ as being at time $t_k$, the previous keyframe camera pose $\mathbf{X}_i = (\mathbf{C}_i, \mathbf{r}_i)$ as being at time $t_i$, and the RMI $\Delta\mathbf{X}_{ik} = (\Delta\mathbf{C}_{ik}, \Delta\mathbf{r}_{ik})$ connecting the two. The orientation component $\Delta\mathbf{C}_{ik}$ and position component $\Delta\mathbf{r}_{ik}$ are incrementally updated with the IMU measurement that arrived at $t_{k-1}$ according to

$$\Delta\mathbf{C}_{ik} = \Delta\mathbf{C}_{ik-1}\,\mathrm{Exp}(\mathbf{u}_{k-1}T_{k-1}), \tag{17}$$

$$\Delta\mathbf{r}_{ik} = \Delta\mathbf{r}_{ik-1} + \Delta\mathbf{C}_{ik-1}(\mathbf{v}_{k-1}T_{k-1}), \tag{18}$$

with full derivation omitted for brevity. Note, when a new keyframe is created, the RMI is reset to $\Delta\mathbf{X}_{ik} = (\mathbf{1}, \mathbf{0})$. The

uncertainty on the RMI can also be updated incrementally according to

$$\boldsymbol{\Sigma}_{ik}^{\mathrm{RMI}} = \left(\frac{\partial \Delta\mathbf{X}_{ik}}{\partial \Delta\mathbf{X}_{ik-1}}\right) \boldsymbol{\Sigma}_{ik-1}^{\mathrm{RMI}} \left(\frac{\partial \Delta\mathbf{X}_{ik}}{\partial \Delta\mathbf{X}_{ik-1}}\right)^{\mathsf{T}} + \left(\frac{\partial \Delta\mathbf{X}_{ik}}{\partial \mathbf{w}_{k-1}}\right) \boldsymbol{\Sigma}_{k-1}^{\mathbf{w}} \left(\frac{\partial \Delta\mathbf{X}_{ik}}{\partial \mathbf{w}_{k-1}}\right)^{\mathsf{T}}, \quad (19)$$

where $\mathbf{w}_{k-1} = \begin{bmatrix} \mathbf{w}_{k-1}^{\mathbf{u}\,\mathsf{T}} & \mathbf{w}_{k-1}^{\mathbf{v}\,\mathsf{T}} \end{bmatrix}^{\mathsf{T}}$ enters $\Delta\mathbf{X}_{ik}$ through (11) and (10), and $\boldsymbol{\Sigma}_{k-1}^{\mathbf{w}} = \mathrm{diag}(\boldsymbol{\Sigma}_{k-1}^{\mathbf{u}}, \boldsymbol{\Sigma}_{k-1}^{\mathbf{v}})$. These Jacobians are derived, however, as discussed in Section IV-A, are not used in the end. For brevity, their explicit form is thus omitted.

The final computed RMI is defined between two keyframes. In the case that, during optimization, involved keyframes are separated by one or more other not involved keyframes, the total RMI can be computed by multiplying the individual RMIs through. For example, if keyframes 2 and 4 are involved in the optimization but 3 is not, the RMI between the involved keyframes can be computed as $\mathbf{X}_{24} = \mathbf{X}_{23}\mathbf{X}_{34}$. Typically, the uncertainty on the RMIs also need to be combined, but this was not considered for this project.

The RMI loss between keyframe $i$ and $j$ is

$$\mathbf{e}_{ij}^{\Delta\mathbf{C}} = \mathrm{Log}\left(\Delta\mathbf{C}_{ij}\mathbf{C}_j^{\mathsf{T}}\mathbf{C}_i\right), \quad (20)$$

$$\mathbf{e}_{ij}^{\Delta\mathbf{r}} = \Delta\mathbf{v}_{ij} - \mathbf{C}_i^{\mathsf{T}}\left(\mathbf{r}_j - \mathbf{r}_i\right), \quad (21)$$

$$\mathbf{e}_{ij}^{\mathrm{RMI}} = \begin{bmatrix} \mathbf{e}_{ij}^{\Delta\mathbf{C}^{\mathsf{T}}} & \mathbf{e}_{ij}^{\Delta\mathbf{r}^{\mathsf{T}}} \end{bmatrix}^{\mathsf{T}}, \quad (22)$$

with a corresponding weight $\boldsymbol{\Sigma}_{ij}^{\mathrm{RMI}}$.

The total final RMI loss for some subset of keyframes $z$, for example $z \in [2, 4]$ in the example above, is written

$$\mathcal{L}_{\mathrm{RMI}}^{\mathrm{map}} = \sum_{q=2}^{\mathrm{len}(z)} \mathbf{e}_{z[q-1]z[q]}^{\mathrm{RMI}^{\mathsf{T}}} \boldsymbol{\Sigma}_{z[q-1]z[q]}^{\mathrm{RMI}} \mathbf{e}_{z[q-1]z[q]}^{\mathrm{RMI}}. \quad (23)$$

### C. Background Model

In order to extend NICE-SLAM to *unbounded* environments without required infinite memory resources, we encode the background of given scene using a spherical grid. This representation is similar to the Multi-Sphere Images (MSI) proposed in [14] and used in [5].

*1) Modeling A Sphere at Infinity:* In order to preserve real-time capability by keeping the representation computationally light, we have opted to model the background with only one background sphere. This sphere is modeled as if it is infinitely far away from the central world frame and, by extension, any given camera frame. This allows to represent distant objects on the horizon, which can be very helpful for camera/robot orientation localization.

This representation simplifies some of our computations, since the only the direction of the pixel ray is required to evaluate the contribution of the background model. To see why this is true, consider a point on the background sphere along a given pixel ray in the camera frame, $\boldsymbol{x}_c = \beta\hat{\boldsymbol{x}}_c$, where $\hat{\boldsymbol{x}}_c$ is the normalized direction of the ray and $\beta$ represents the magnitude or the ray. The normalized ray in the world frame can be expressed as

$$\hat{\boldsymbol{x}}_w = \frac{\boldsymbol{x}_w}{\|\boldsymbol{x}_w\|_2} = \frac{\boldsymbol{C}_{wc}\boldsymbol{x}_c + \boldsymbol{r}_w^{cw}}{\|\boldsymbol{x}_c + \boldsymbol{r}_w^{cw}\|_2} = \frac{\boldsymbol{C}_{wc}\beta\hat{\boldsymbol{x}}_c + \boldsymbol{r}_w^{cw}}{\beta\left\|\hat{\boldsymbol{x}}_c + \boldsymbol{r}_w^{cw}/\beta\right\|_2}. \quad (24)$$

Now, as $\beta \to \infty$, it is clear that $\hat{\boldsymbol{x}}_w \simeq \boldsymbol{C}_{wc}\hat{\boldsymbol{x}}_c$, that is, we only need to evaluate the sphere along the rotated camera frame ray.

*2) Background NeRF Model:* Consider the formula for a pixel colour corresponding to a given ray, $\boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d}$, as given in the original NeRF paper [1]:

$$\hat{I}_n = \int_{t_n}^{t_f} T(t)\sigma(\boldsymbol{r}(t))\boldsymbol{c}(\boldsymbol{r}(t))dt, \quad (25)$$

$$\text{where } T(t) = \mathrm{Exp}\left(-\int_{t_n}^{t_f} \sigma(\boldsymbol{r}(s))ds\right) \quad (26)$$

where $\sigma(\boldsymbol{x})$ represents the scene volume density, $T(t)$ represents the accumulated transmittance (probability that a ray travels from $t$ to $t_n$), $\boldsymbol{c}(\boldsymbol{r}(t))$ represents the colour at a given point and $t_n$, $t_f$, represent the near and far limits of the depth parameter, $t$, respectively. Now, suppose we allow the far limit to extend to infinity, $t_f \to \infty$. We can rewrite the integral as follows:

$$\begin{aligned}\hat{I}_{n,\infty} &= \int_{t_n}^{\infty} T(t)\sigma(\boldsymbol{r}(t))\boldsymbol{c}(\boldsymbol{r}(t))dt \\ &= \int_{t_n}^{t_f} T(t)\sigma(\boldsymbol{r}(t))\boldsymbol{c}(\boldsymbol{r}(t))dt + \int_{t_f}^{\infty} T(t)\sigma(\boldsymbol{r}(t))\boldsymbol{c}(\boldsymbol{r}(t))dt \\ &= \hat{I}_n + T(t_f)\boldsymbol{c}_\infty(\boldsymbol{r}(t)).\end{aligned}$$

where $\hat{I}_n$ is the original NICE-SLAM *foreground* NeRF and $\boldsymbol{c}_\infty(\boldsymbol{r}(t)) = \int_{t_f}^{\infty} T(t)\sigma(\boldsymbol{r}(t))\boldsymbol{c}(\boldsymbol{r}(t))$ represents the evaluation from $t_f$ to $\infty$ of a separate *background* NeRF. We have made use of the fact that

$$\begin{aligned}T(c) &= \mathrm{Exp}(-\int_a^c \sigma(\boldsymbol{r}(s))ds) \\ &= \mathrm{Exp}(-\int_a^b \sigma(\boldsymbol{r}(s))ds)\,\mathrm{Exp}(-\int_b^c \sigma(\boldsymbol{r}(s))ds) \\ &= T(b)T(b,c).\end{aligned}$$

We see that we can represent the background of the scene by simply adding a defined *background* colour, $\boldsymbol{c}_\infty(\boldsymbol{r}(t))$, multiplied by the final transmittance value at the boundary foreground NeRF, $T(t_f)$.

We assume that the background is mostly empty except for some points that are very far away. By the arguments presented in Section III-C.1, this allows us to represent the background colour only in terms of the ray direction: $\boldsymbol{c}_\infty(\boldsymbol{r}(t)) \simeq \boldsymbol{c}_\infty(\boldsymbol{d}) = c_{bg}(\boldsymbol{d})$.

Similar to the encoded color grid for NeRF-SLAM, we generate a 2D grid of background features that are warped onto a sphere. At render time, we use the grid and a pixel ray, $d$, to interpolate a feature, then decode the feature and scale by the boundary transmittance of the foreground NeRF, $T(t_f)$[2].

[2]Note that NICE-SLAM already computes the rotated rays, $d$, and boundary transmittances, $T(t_f)$, so we incur no additional cost to use them

## IV. RESULTS

### A. Implementation Details

We implemented the changes outlined in this paper using the existing NICE-SLAM codebase [2], making modifications to the rendering and loss functions as appropriate. To test our changes, we initially tested on the "Demo" dataset provided in [2], but used scenes from the Replica dataset [15] for depth uncertainty and IMU testing and TUM RGB-D dataset [16] for final testing for background sphere testing.

In order to test the impact of the listed changes on robustness, the algorithm is tested with the full, default number of iterations suggested by the authors of NICE-SLAM and with a heavily reduced number of iterations to approach real-time performance. Specifically, for the Replica datasets the "full number of iterations" corresponds to 10 and 60 iterations for tracking and mapping respectively, whereas the "low number of iterations" corresponds to 5 and 10 iterations for tracking and mapping respectively. In some circumstances, reducing the number of iterations is required to ensure real-time capability. Indeed, we found that this reduction in number of iterations reduced the overall runtime of the NICE-SLAM from 48 minutes to 10 minutes.

In practice, because the tracking and mapping threads are run in parallel, it was found to be challenging to compute RMIs at non-keyframe frequency. As a result, an RMI connection to the most recent state, which is involved in mapping but not guaranteed to be a keyframe, is not included.

### B. Depth Uncertainty and IMU Results

The main results for experiments with depth uncertainty and IMU data are shown in Table I and Figure 1.

Depth uncertainty parameters were set based on the specified depth uncertainty values given for the cameras used in each dataset. Typically, this uncertainty increases linearly with the depth measurement and depends on the stereo baseline.

It was initially found that the depth uncertainty was less impactful on the results of NICE-SLAM. However, it turns out that this was only true in smaller environments for which the depth uncertainty does not vary greatly ("Demo" dataset from [2]). When larger environments were considered (i.e., from Replica dataset [15]), it was found that depth uncertainty moderately improved most metrics and considerably improved metrics for both rooms when used in combination with IMU data.

Across all experiments, the standard deviation of the additive noise was 0.01 rad/s and 0.01 m/s for rotation and translation measurements, respectively. In general, integration of IMU data lead to drastic improvement in both tracking error and map reconstruction error. Most notably, for the Room 1 test with reduced number of iterations, an order of magnitude improvement can be seen in Table I for the RMSE and worst case error for tracking as well as the average accuracy (Acc.) of the reconstructed map.

The improvement at low iterations can also be seen in Figure 1 (b) and (d). Note that in (b) the localization and room reconstruction approaches catastrophic failure when the number of iterations is reduced to improve computation time. In contrast, even with reduced number of iterations, (d) shows that the IMU and depth uncertainty measurements lead to a consistent representation. This highlights an important conclusion: addition of depth uncertainty and IMU data can reduce the number of iterations required for convergence, thereby increasing the runtime frequency of NICE-SLAM.

### C. Background Representation

The background NeRF representation outlined in Section III-C was implemented and tested on a subset of images from the Freiburg2 RPY scene from the TUM RGB-D dataset. This scene was selected because it involved mostly only rotations of the camera and because the scene itself consisted of distinct foreground and background elements.

To simplify this experiment, the tracking part of NICE-SLAM was disabled and the ground truth camera positions were used. Additionally, the IMU measurements and depth uncertainty measurements were not included in this experiment. The bounding box of the foreground NeRF was intentionally set to a small value (1.5 m) to observe the effect of the including the background. Figure 2 shows the background model and resulting image reconstruction (with and without background) for three scenes in the dataset. By comparing columns c) and d) with a) it is clear that the background model allows the details in the background to be represented more accurately. For this experiment, the average colour loss was decreased by 18.32% when the background model was included.

## V. CONCLUSIONS

We have shown how the addition of depth uncertainty and IMU data can improve the accuracy of NICE-SLAM, especially when reducing the number of allowed iterations for each timestep. Additionally, we have demonstrated that spherical background model can improve the image reconstruction of NICE-SLAM when the scene is too large to be modeled by the foreground NeRF.

In the future, the goal would be to test the implemented changes in a large environment where background visual information is significantly far away from the robot. Additionally, in order to make NeRF-based SLAM truly open world, it is required to remove the dependency on a predefined grid. Potential work in this area would involve switching to a dynamically expanding octree. This would allow a robot to explore new environments while maintaining reasonable memory requirements by only forming finer grid resolutions around objects.

## ACKNOWLEDGMENT

Thank you Alec Krawciw for taking the time out of your weekend to go to UTIAS and turn on the lab computer so that we could continue running tests for this project.

(a) **NICE-SLAM:** Full number of iterations Default

(b) **NICE-SLAM:** Low number of iterations Default

(c) **Ours:** Full number of iterations IMU + Dep.

(d) **Ours:** Low number of iterations IMU + Dep.

Fig. 1: Reconstruction results for NICE-SLAM and our proposed modifications run at a full and low number of iterations.

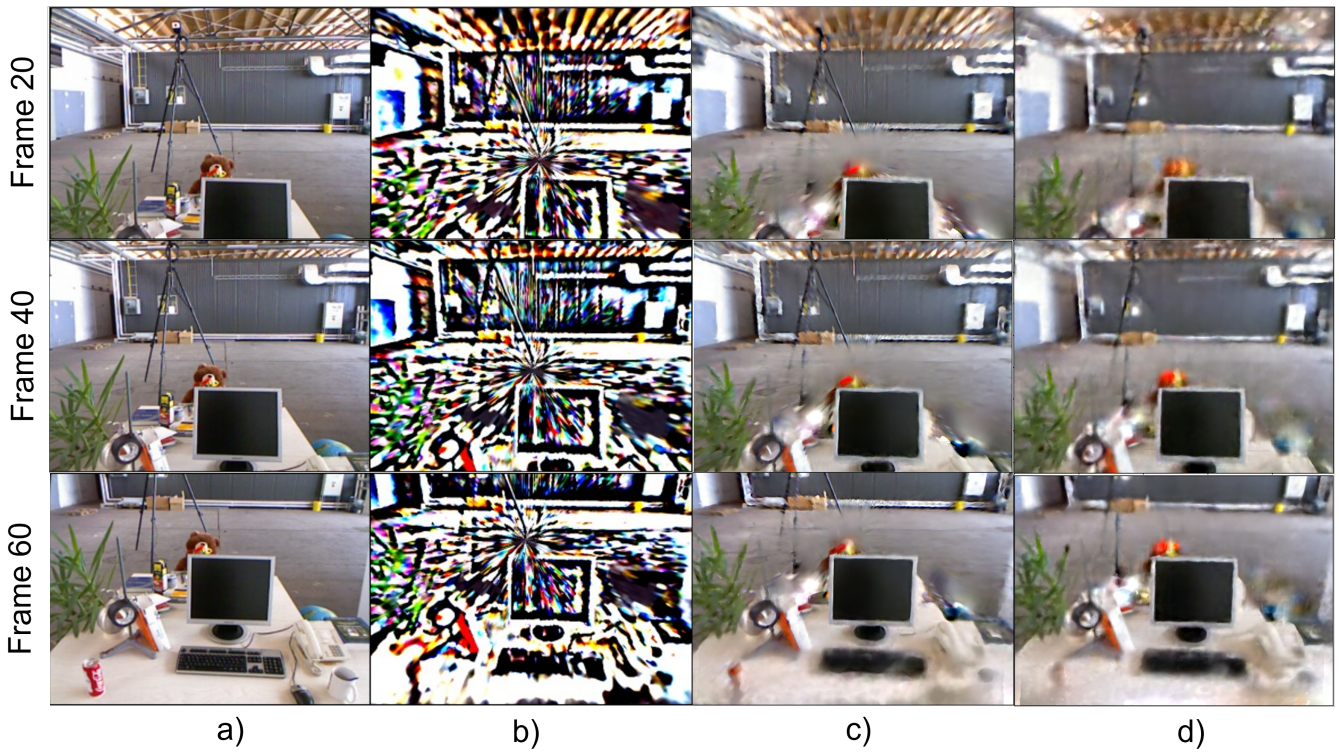


Fig. 2: Experiment demonstrating the effect of the background sphere across different frames in the Freiburg2 RPY scene from the TUM RGB-D dataset. **a)** Input image; **b)** Background sphere only; **c)** NICE-SLAM with background active; **d)** Original NICE-SLAM (no background model).

## 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," Aug. 2020, arXiv:2003.08934 [cs].

| | Set Up | Room 0 | | | | | | Room 1 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Tracking | | Mapping | | | | Tracking | | Mapping | | | |
| | | RMSE [cm] ↓ | Max. [cm] ↓ | Acc. [cm] ↓ | Comp. [cm] ↓ | C.R. [%] ↑ | Dep. L1 [cm] ↓ | RMSE [cm] ↓ | Max. [cm] ↓ | Acc. [cm] ↓ | Comp. [cm] ↓ | C.R. [%] ↑ | Dep. L1 [cm] ↓ |
| Full Iter. | Default | 0.020 | 0.072 | 2.744 | 3.000 | **91.03** | 1.922 | 0.026 | 0.103 | 2.796 | 2.376 | **92.43** | 1.732 |
| | Depth | **0.018** | 0.074 | 2.646 | **2.840** | **91.25** | **1.763** | **0.018** | **0.048** | 2.430 | **2.257** | **93.54** | 1.488 |
| | IMU | 0.019 | **0.053** | **2.442** | **2.756** | **91.10** | **1.752** | 0.019 | 0.061 | **2.300** | **2.298** | **93.12** | 1.493 |
| | IMU + Dep. | **0.019** | 0.057 | 2.796 | **2.855** | **90.71** | **1.700** | 0.020 | 0.057 | 2.671 | **2.293** | **93.22** | **1.378** |
| Low Iter. | Default | 0.156 | 0.627 | 5.083 | 5.030 | 67.29 | 5.677 | 0.531 | 1.148 | 15.55 | 16.67 | 35.68 | 24.98 |
| | Depth | 0.211 | 0.539 | 5.641 | 6.330 | 66.13 | 7.937 | 0.3429 | 0.8199 | 18.17 | 18.57 | 28.79 | 25.89 |
| | IMU | **0.029** | **0.079** | **3.357** | **3.646** | **84.61** | **3.258** | **0.031** | 0.104 | 3.116 | 3.360 | **84.51** | 3.189 |
| | IMU + Dep. | **0.029** | 0.085 | **3.217** | **3.595** | **84.98** | **3.293** | **0.030** | **0.089** | **2.635** | **2.968** | **88.35** | **2.373** |

TABLE I: Results from two trials on "Replica" dataset. Best results (within 5% of pre-rounded values) are in bold. **RMSE** refers to Root Mean Squared Error; **Max.** refers to worst cases error; **Comp.** refers to the average distance to the nearest reconstructed mesh from ground truth mesh; **C.R.** refers to the completion ratio, or percentage of points with completion under 5 cm; **Dep. L1** refers to sum of absolute depth errors.


[2] Z. Zhu, S. Peng, V. Larsson, W. Xu, H. Bao, Z. Cui, M. R. Oswald, and M. Pollefeys, "NICE-SLAM: Neural Implicit Scalable Encoding for SLAM," Apr. 2022, arXiv:2112.12130 [cs].

[3] V. Rudnev, M. Elgharib, W. Smith, L. Liu, V. Golyanik, and C. Theobalt, "NeRF for Outdoor Scene Relighting," Jul. 2022, arXiv:2112.05140 [cs].

[4] T. Takikawa, J. Litalien, K. Yin, K. Kreis, C. Loop, D. Nowrouzezahrai, A. Jacobson, M. McGuire, and S. Fidler, "Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Nashville, TN, USA: IEEE, Jun. 2021, pp. 11 353–11 362.

[5] A. Yu, S. Fridovich-Keil, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa, "Plenoxels: Radiance Fields without Neural Networks," Dec. 2021, arXiv:2112.05131 [cs].

[6] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Transactions on Graphics*, vol. 41, no. 4, pp. 1–15, Jul. 2022.

[7] C.-H. Lin, W.-C. Ma, A. Torralba, and S. Lucey, "BARF: Bundle-Adjusting Neural Radiance Fields," Aug. 2021, arXiv:2104.06405 [cs].

[8] Z. Wang, S. Wu, W. Xie, M. Chen, and V. A. Prisacariu, "NeRF–: Neural Radiance Fields Without Known Camera Parameters," Apr. 2022, arXiv:2102.07064 [cs].

[9] E. Sucar, S. Liu, J. Ortiz, and A. J. Davison, "iMAP: Implicit Mapping and Positioning in Real-Time," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. Montreal, QC, Canada: IEEE, Oct. 2021, pp. 6209–6218.

[10] A. Rosinol, J. J. Leonard, and L. Carlone, "NeRF-SLAM: Real-Time Dense Monocular SLAM with Neural Radiance Fields," Oct. 2022, arXiv:2210.13641 [cs].

[11] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, Oct. 2015. [Online]. Available: https://ieeexplore.ieee.org/document/7219438/

[12] J. Solà, J. Deray, and D. Atchuthan, "A Micro Lie Theory for State Estimation in Robotics," `arXiv:1812.01537v8`, 2018.

[13] T. Lupton and S. Sukkarieh, "Visual-Inertial-Aided Navigation for High-Dynamic Motion in Built Environments without Initial Conditions," *IEEE Trans. Robot.*, vol. 28, no. 1, pp. 61–76, 2012.

[14] K. Zhang, G. Riegler, N. Snavely, and V. Koltun, "NeRF++: Analyzing and Improving Neural Radiance Fields," Oct. 2020, arXiv:2010.07492 [cs]. [Online]. Available: http://arxiv.org/abs/2010.07492

[15] J. Straub, T. Whelan, L. Ma, Y. Chen, E. Wijmans, S. Green, J. J. Engel, R. Mur-Artal, C. Ren, S. Verma, A. Clarkson, M. Yan, B. Budge, Y. Yan, X. Pan, J. Yon, Y. Zou, K. Leon, N. Carter, J. Briales, T. Gillingham, E. Mueggler, L. Pesqueira, M. Savva, D. Batra, H. M. Strasdat, R. De Nardi, M. Goesele, S. Lovegrove, and R. Newcombe, "The replica dataset: A digital replica of indoor spaces," no. arXiv:1906.05797, Jun 2019, arXiv:1906.05797 [cs, eess].

[16] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.


## VI. Supplementary Material

An example of a simplified (black and white) background rendering can be seen in Figure 3. This was used initially to debug the background model, but has been included in this section since it is instructive.

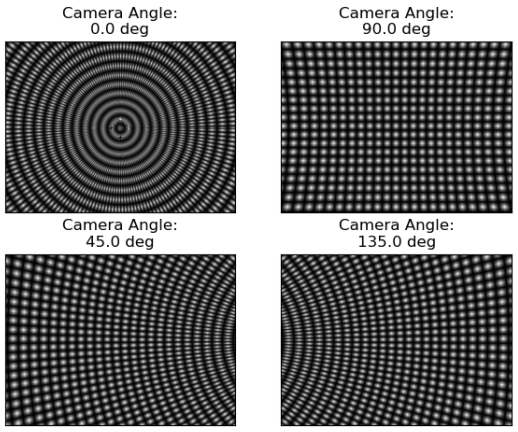


Fig. 3: Interpolated rendering at different angles of checkerboard background grid for angle divisions of 0.02 (rad). Zero (deg) camera angle corresponds to z axis in world frame.