LONER: LiDAR Only Neural Representations for Real-Time SLAM

Seth Isaacson*,1, Pou-Chun Kung*,1, Mani Ramanagopal[1], Ram Vasudevan[1], and Katherine A. Skinner[1]

***Abstract*—This paper proposes *LONER*, the first real-time LiDAR SLAM algorithm that uses a neural implicit scene representation. Existing implicit mapping methods for LiDAR show promising results in large-scale reconstruction, but either require groundtruth poses or run slower than real-time. In contrast, LONER uses LiDAR data to train an MLP to estimate a dense map in real-time, while simultaneously estimating the trajectory of the sensor. To achieve real-time performance, this paper proposes a novel information-theoretic loss function that accounts for the fact that different regions of the map may be learned to varying degrees throughout online training. The proposed method is evaluated qualitatively and quantitatively on two open-source datasets. This evaluation illustrates that the proposed loss function converges faster and leads to more accurate geometry reconstruction than other loss functions used in depth-supervised neural implicit frameworks. Finally, this paper shows that LONER estimates trajectories competitively with state-of-the-art LiDAR SLAM methods, while also producing dense maps competitive with existing real-time implicit mapping methods that use groundtruth poses.**

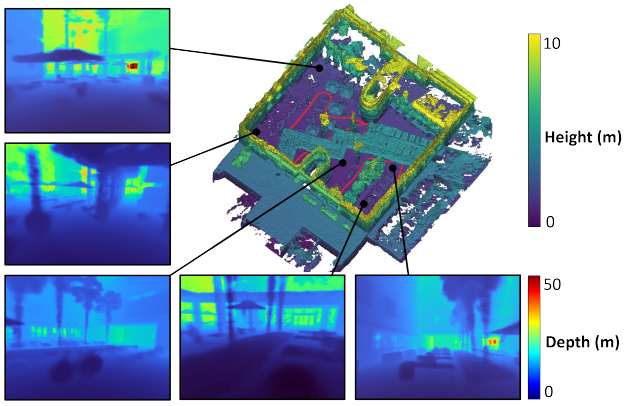

Fig. 1. LONER reconstruction on a courtyard scene [2]. The top-right is a mesh reconstruction with the estimated trajectory in red. The surrounding images are rendered depth images from novel views outside of the training trajectory, demonstrating LONER's ability to reconstruct dense novel views of an environment.

I. Introduction

NEURAL implicit scene representations, such as Neural Radiance Fields (NeRFs), offer a promising new way to represent maps for robotics applications [1]. Traditional NeRFs employ a Multi-Layer Perceptron (MLP) to estimate the radiance and volume density of each point in space, enabling dense scene reconstruction and novel view synthesis. This paper advances neural implicit scene representations for robotics applications. Specifically, we introduce the first real-time LiDAR-only SLAM algorithm that achieves accurate pose estimation and map reconstruction while learning a neural implicit representation of a scene.

Several recent papers have proposed real-time NeRF-based visual SLAM systems using monocular or RGB-D cameras [3, 4, 5]. These systems demonstrate impressive performance on indoor scenes. For outdoor environments, prior work has focused on using neural implicit representations for LiDAR to enable dense 3D reconstruction and novel view synthesis for large-scale scenes [6, 7, 8]. Recent methods have even shown promising results for LiDAR localization and mapping with neural implicit frameworks in large-scale outdoor scenes [9, 10]. Still, these LiDAR-supervised algorithms do not operate in real-time, which is necessary for robotics applications.

The contributions of this paper are as follows:

1) We propose the first real-time neural implicit LiDAR SLAM method, which adapts to outdoor environments and provides accurate online state estimation.
2) We introduce a novel loss function that leads to faster convergence and more accurate reconstruction than existing loss functions.

We demonstrate that our proposed method, LONER, runs in real-time and estimates both trajectories and maps more accurately than baselines. Figure 1 shows the reconstruction results on the Fusion Portable dataset [2]. A project page and code are available at https://umautobots.github.io/loner. An extended paper with additional experiments has recently been published in RA-L.

II. Related Works

A. Neural Implicit Representations for LiDAR

While neural implicit representations were initially developed for visual applications, several works have introduced neural implicit representations for LiDAR to improve outdoor 3D reconstruction performance [6, 11, 7]. These methods all use ground-truth poses, while ours also estimates the camera trajectory.

*These two authors contributed equally to this work.

This work was supported by the Ford Motor Company via the Ford-UM Alliance under award N028603.

[1]S. Isaacson, P. Kung, M. Ramanagopal, R. Vasudevan, and K. A. Skinner are with the Department of Robotics, University of Michigan, Ann Arbor, MI 48109. {sethgi, pckung, srmani, ramv, kskin}@umich.edu.

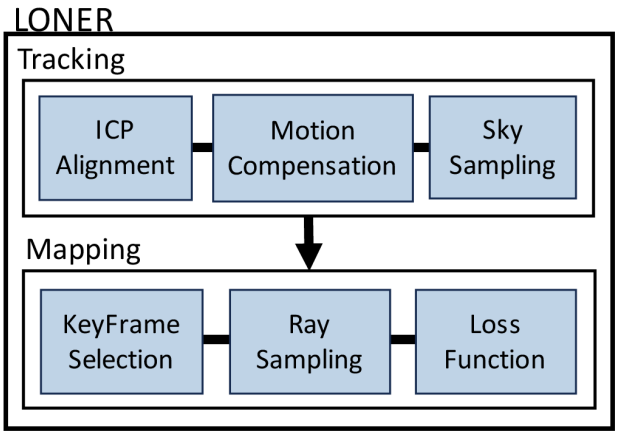

Fig. 2. LONER system overview. Scans are tracked with ICP, then the map is updated using our novel loss function.

B. Loss for Depth-supervised NeRF

Depth-supervised NeRF frameworks, such as those that use RGB-D sensors, typically use the difference between rendered and sensed depth as a loss to learn geometry from 2D images by volumetric rendering [3, 4]. Other works use depth measurements directly in 3D space to perform depth-supervision [11, 6, 7, 10]. The Line-Of-Sight (LOS) loss introduced by URF [6] approximate each LiDAR ray's termination depth as a normal distribution centered at the measured depth. The variance of the distribution is correlated with a margin parameter ϵ. The loss functions encourage the network to predict weights along a ray equal to the PDF of the normal distribution. As training progresses, ϵ is decayed. While uniformly decaying a margin is successful offline, using a single margin for all rays is unsuitable for real-time SLAM, which has incremental input and limited training samples. Using a uniform margin can force the NeRF model to forget learned geometry when adding a new LiDAR scan and can cause slower convergence. Therefore, this paper proposes a novel dynamic margin loss that applies a different margin for each ray.

III. Method

This section provides a high-level overview of our proposed system, LONER, before explaining each component in detail.

A. System Overview

An overview of LONER is shown in Fig. 2. As is common in the SLAM literature [3, 4], the system comprises parallel threads for tracking and mapping. The tracking thread processes incoming scans and estimates odometry using ICP. In parallel and at a lower rate, the mapping thread uses the current scan and selected prior scans as KeyFrames, which are used to update the training of the neural scene representation.

B. Implicit Map Representation

The scene is represented as an MLP with the hierarchical feature grid encoding from [12]. During online training, the parameters Θ of the MLP and the feature grid are updated to predict the volume density σ of each point in space. To train the network and estimate depths, we follow the standard volumetric rendering procedure [1]. In particular, for a LiDAR ray $\vec{r}$ with origin $\vec{o}$ and direction $\vec{d}$, we choose distances $t_i \in [t_{near}, t_{far}]$ to create N_S samples $s_i = \vec{o} + t_i\vec{d}$. LiDAR intrinsics dictate t_{near}, while t_{far} depends on the scale of the scene. The feature grid and MLP, collectively $\mathcal{F}(s_i; \Theta)$, are queried to predict the occupancy state σ_i. Then, weights transmittances T_i and weights w_i are computed according to:

$$T_i = \exp(-\sum_{j=1}^{i-1} \sigma_j \delta_j) \tag{1}$$

$$w_i = T_i(1 - \exp(-\sigma_i \delta_i)) \tag{2}$$

where $\delta_j = t_{j+1} - t_j$, and σ_i is the density at sample s_i predicted by the MLP. The weights w_i are used by the loss function and represent the probability that the ray terminates at each point. Therefore, the expected termination depth of a ray $\hat{D}(\vec{r})$ can be estimated as

$$\hat{D}(\vec{r}) = \sum_{i=1}^{N} w_i t_i. \tag{3}$$

C. Mapping

The mapping thread receives LiDAR scans from the tracking thread and determines whether to form a KeyFrame. If 3 seconds have passed since the previous KeyFrame, the scan is accepted as a KeyFrame.

1) Optimization: Each time a KeyFrame is accepted, the map is updated. Eight total KeyFrames are used in the update, including the current KeyFrame and seven random selected past KeyFrames. The map is jointly optimized with the poses of KeyFrames in the optimization window. The poses are used to parameterize the rays, and 512 rays are sampled at random from the LiDAR scan. In the backward pass, gradients are computed for MLP and feature grid parameters, as well as the poses. At the end of the optimization, the final poses are sent to the tracking thread, such that future tracking is performed relative to the optimized poses.

D. JS Dynamic Margin Loss Function

The primary loss function in our system is a novel dynamic margin loss. This is combined with terms for depth loss and sky loss as follows:

$$\mathcal{L}(\Theta) = \mathcal{L}_{JS} + \lambda_1 \mathcal{L}_{depth} + \lambda_2 \mathcal{L}_{sky}. \tag{4}$$

Each of these terms is explained below.

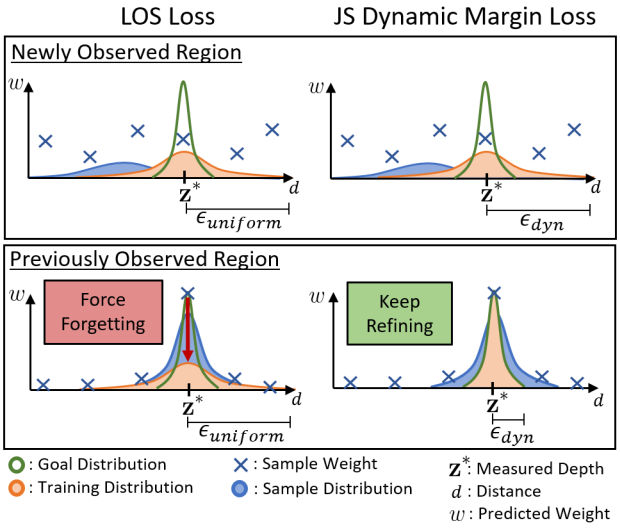

Fig. 3. Illustration of the difference between the JS loss and the LOS loss. The LOS loss sets a uniform margin ϵ for rays pointing to both learned and unobserved regions. This strategy corrupts the learned information by forcing learned regions to predict higher variances. In contrast, the proposed JS loss sets the dynamic margin ϵ for each ray depending on the similarity between goal distribution and predicted sample distribution. The JS loss sets higher margins for rays in unobserved regions to improve convergence, and sets lower margins for rays in learned regions to refine learned geometry.

1) JS Loss Formulation: The LOS loss used by [6, 7] uses a single margin for all rays; we use a similar formulation but introduce a novel strategy based on the Jensen-Shannon Divergence to assign a unique margin to each ray. For a given LiDAR ray $\vec{r}$, the samples along the ray are $s_i = \vec{o} + t_i \vec{d}$, and z^* denotes the measured depth along the ray. t_i denotes the distance of individual training samples along the ray, and w_i represents a corresponding weight prediction from an MLP, as defined in Equation 2. We define a truncated Gaussian distribution $\mathcal{K}_\epsilon$ that has a bounded domain parameterized by margin ϵ, with $\mathcal{K}_\epsilon = \mathcal{N}(0, (\epsilon/3)^2)$ as the training distribution. Thus, target weights are given by $w_i^* = \mathcal{K}_\epsilon(t_i - z^*)$. The JS loss is defined as

$$\mathcal{L}_{JS}(\Theta) = \underbrace{\|w_i^* - w_i\|_1}_{\text{Primary Loss}} + \underbrace{\|1 - \sum_i w_i\|_1}_{\text{Opacity Loss}}, \tag{5}$$

where the opacity loss (explained in more detail by [7]) ensures weights along each ray sum to one and thus form a probability distribution. Note that while URF [6] uses an L2 loss to compute the LOS loss, we follow [7] and use an L1 loss. The effect of this is discussed in Section IV-B.

In [6, 7], the margin decays exponentially throughout training and, at each iteration, a single margin is shared by all of the rays. In contrast, we present a JS divergence-based dynamic margin that computes a unique margin for each ray to improve the training convergence and reconstruction accuracy.

In a SLAM application, continuous optimization, sparse sampling, and incremental input lead to different regions of the map being learned to varying degrees during online training. As shown in Fig. 3, using a uniform ϵ in the LOS loss causes forgetting in regions that have already been learned. The idea of the JS dynamic margin is to use a larger margin for rays pointing toward regions of the map with unknown geometry while using a smaller margin for rays pointing toward well-learned regions. This allows the system to learn new regions while preserving and refining learned geometry. We use the JS divergence to measure the dissimilarity between the goal distribution and the sample distribution for each ray, which represents how well the map has learned along the ray. Learned regions have similar goal and sample distributions, which lead to smaller JS divergence. We define a goal distribution $G = \mathcal{N}(z^*, \sigma^*)$, where $\sigma^* = \epsilon_{min}/3$. Further, we define the sample distribution $S = \mathcal{N}(\bar{\mu}_w, \bar{\sigma}_w)$, where $\bar{\mu}_w$ and $\bar{\sigma}_w$ denote mean and standard deviation of the predicted weights along a particular ray. The dynamic margin is then defined as

$$\epsilon_{dyn} = \epsilon_{min}(1 + \alpha \mathbf{J}^*) \tag{6}$$

$$\mathbf{J}^* = \begin{cases} 0 & JS(G||S) < JS_{min} \\ JS_{max} & JS(G||S) > JS_{max} \\ JS(G||S) & \text{otherwise,} \end{cases} \tag{7}$$

where α is a constant scaling parameter. JS_{max} denotes the upper bound of the JS score, and JS_{min} denotes a threshold for scaling. Once the JS score is smaller than JS_{min}, ϵ_{dyn} is equal to ϵ_{min}. In practice, we set $JS_{min} = 1$, $JS_{max} = 10$, and $\epsilon_{min} = 0.5$.

2) Depth Loss: As in [6], we use the depth loss as an additional term in the loss function. The depth loss is the error between rendered depth and LiDAR-measured depth along each ray. The loss is defined as

$$\mathcal{L}_{depth}(\Theta) = \|\hat{D}(\vec{r}) - z^*\|_2^2 \tag{8}$$

We found the depth loss contributes to blurry reconstruction with limited training time, but still provides good hole-filling. Hence, unlike [6] which weights depth loss and LOS loss equally, we down-weight the depth loss by setting $\lambda_1 = 5 \times 10^{-6}$.

Finally, similar to [6], we add an additional loss to force weights on rays pointing at the sky to be zero. We identify holes in the LiDAR scan and assume them to correspond to sky regions, then add a loss forcing weights along those corresponding rays toward zero.

E. Meshing

To form a mesh from the implicit geometry, a virtual LiDAR is placed at estimated KeyFrame poses. We compute weights along LiDAR rays, then bucket the weights into a 3D grid. When multiple weights fall within the same grid cell, the maximum value is kept. Marching cubes is then used to form a mesh from the result. This process runs offline for visualization and evaluation, and is not a part of online training.

TABLE I
Pose tracking results on Fusion Portable and Newer College sequences. Reported metric is RMS APE (m). An ✗ indicates the algorithm failed.

	MCR	Canteen	Garden	Quad
LeGO-LOAM	0.052	0.129	0.161	**0.126**
NICE-SLAM	0.248	✗	✗	-
LONER w./ $\mathcal{L}_{\text{URF}}$	0.047	0.952	0.928	0.931
LONER w./ $\mathcal{L}_{\text{CLONeR}}$	0.034	0.071	0.073	0.306
LONER	**0.029**	**0.064**	**0.056**	0.130

IV. Experiments

This section evaluates the trajectory estimation and mapping accuracy of LONER against state-of-the-art baselines. Refer to the extended paper on the project website for additional comparisons and ablation studies.

A. Baselines

We evaluate against NICE-SLAM [4] and LeGO-LOAM [13], which represent state-of-the-art methods in neural-implicit SLAM and LiDAR SLAM respectively. Additionally, we evaluate our SLAM pipeline with the loss functions from CLONeR [7] and URF [6]. We refer to these approaches as "LONER w./ $\mathcal{L}_{\text{CLONeR}}$" and "LONER w./ $\mathcal{L}_{\text{URF}}$" respectively. Finally, mapping performance is compared to SHINE mapping, which is run with groundtruth poses [8].

We evaluate performance on two open source datasets, Fusion Portable [2] and Newer College [14]. Collectively, the chosen sequences represent a range of scales and difficulties. For testing the Fusion Portable RGB-D sequences with NICE-SLAM, we simulate RGB-D from stereo using RAFT [15].

B. Performance Analysis

1) Trajectory Tracking Evaluation: Trajectory estimates from each algorithm are evaluated using the open-source EVO package, which aligns the trajectories and then computes RMS APE (absolute pose error), denoted t_{APE}. Table I compares trajectory performance to state-of-the-art methods [13, 4]. Our method offers performance competitive with or better than existing state-of-the-art LiDAR SLAM, while outperforming the neural-implicit baselines.

2) Map Evaluation: To evaluate maps, point clouds are created by generating a mesh, then sampling a point cloud from the mesh. Point clouds are then downsampled to 5cm resolution for all scenes except MCR, and 1cm for MCR.

Map metrics include accuracy (mean distance from each point in the estimated map to each point in the groundtruth map) and completion (mean distance from each point in the groundtruth map to each point in the estimated map) [3, 4]. Additionally, precision and recall are computed with a 0.1m threshold. Table II shows quantitative evaluation for map reconstruction performance. LONER performs competitively with or better than the baselines in all tests. LONER and SHINE Mapping out-perform the other baselines. Note only SHINE was run with ground-truth poses.

TABLE II
Comparison of map Acccuracy (m), Completion (m), Precision, and Recall between proposed and baseline algorithms. A '-' indicates invalid configurations, while '✗' indicates that the algorithm failed.

		NICE SLAM	SHINE	LONER w./ $\mathcal{L}_{\text{CLONeR}}$	LONER w./ $\mathcal{L}_{\text{URF}}$	LONER
MCR	Acc.	0.621	0.164	**0.110**	0.153	0.186
	Cmp.	0.419	0.075	0.080	0.102	**0.069**
	Prec.	0.124	0.624	**0.665**	0.449	0.473
	Rec.	0.476	0.757	**0.940**	0.884	0.932
Canteen	Acc.	✗	-	-	-	-
	Cmp.		0.116	0.220	0.190	**0.105**
	Prec.		-	-	-	-
	Rec.		0.753	0.524	0.846	**0.878**
Garden	Acc.	✗	-	-	-	-
	Cmp.		**0.130**	0.333	0.539	0.157
	Prec.		-	-	-	-
	Rec.		0.657	0.469	0.623	**0.784**
Quad	Acc.	-	**0.301**	0.663	0.552	0.380
	Cmp.	-	**0.148**	0.543	0.895	0.373
	Prec.	-	**0.453**	0.150	0.127	0.327
	Rec.	-	0.717	0.602	0.484	**0.809**

C. Runtime

Runtime performance was evaluated on a computer with an AMD Ryzen 5950X CPU and an NVidia A6000 GPU, which is similar to the platform used to benchmark NICE-SLAM [4]. Each tracking step takes an average of 14ms to compute, which is faster than is needed by the 5Hz configuration. In the mapping step, the system finishes processing a KeyFrame in under the 3 seconds allotted per KeyFrame. This ensures the system can keep up with input sensor data.

V. Conclusions and Future Work

This paper proposed LONER, the first real-time LiDAR SLAM algorithm with an implicit neural map representation. To achieve real-time SLAM, we presented a novel loss function for depth-supervised training. Results demonstrated that the JS loss outperforms current loss functions in both reconstruction accuracy and hole-filling while maintaining low computational costs. Tests demonstrated that LONER achieves state-of-the-art map and trajectory quality, while providing an implicit geometry representation to support novel view depth rendering.

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," *Commun. ACM*, vol. 65, no. 1, p. 99–106, Dec 2021.

[2] J. Jiao, *et al.*, "Fusionportable: A multi-sensor campus-scene dataset for evaluation of localization and mapping accuracy on diverse platforms," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3851–3856, 2022.

[3] E. Sucar, S. Liu, J. Ortiz, and A. J. Davison, "imap: Implicit mapping and positioning in real-time," *2021 IEEE/CVF International Conference on Computer Vision*, pp. 6209–6218, 2021.

[4] Z. Zhu, *et al.*, "Nice-slam: Neural implicit scalable encoding for slam," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[5] A. Rosinol, J. J. Leonard, and L. Carlone, "Nerf-slam: Real-time dense monocular slam with neural radiance fields," *ArXiv*, vol. abs/2210.13641, 2022.

[6] K. Rematas, *et al.*, "Urban radiance fields," *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[7] A. Carlson, M. S. Ramanagopal, N. Tseng, M. Johnson-Roberson, R. Vasudevan, and K. A. Skinner, "Cloner: Camera-lidar fusion for occupancy grid-aided neural representations," *IEEE Robotics and Automation Letters*, vol. 8, no. 5, pp. 2812–2819, 2023.

[8] X. Zhong, Y. Pan, J. Behley, and C. Stachniss, "Shine-mapping: Large-scale 3d mapping using sparse hierarchical implicit neural representations," *arXiv preprint arXiv:2210.02299*, 2022.

[9] J. Deng, *et al.*, "Nerf-loam: Neural implicit representation for large-scale incremental lidar odometry and mapping," *arXiv preprint arXiv:2303.10709*, 2023.

[10] D. Yan, X. Lyu, J. Shi, and Y. Lin, "Efficient implicit neural reconstruction using lidar," *arXiv preprint arXiv:2302.14363*, 2023.

[11] K. Deng, A. Liu, J.-Y. Zhu, and D. Ramanan, "Depth-supervised nerf: Fewer views and faster training for free," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12 882–12 891.

[12] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Trans. Graph.*, vol. 41, no. 4, pp. 102:1–102:15, July 2022.

[13] T. Shan and B. Englot, "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018, pp. 4758–4765.

[14] M. Ramezani, Y. Wang, M. Camurri, D. Wisth, M. Mattamala, and M. Fallon, "The newer college dataset: Handheld lidar, inertial and vision with ground truth," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020, pp. 4353–4360.

[15] Z. Teed and J. Deng, "Raft: Recurrent all-pairs field transforms for optical flow," in *European Conference on Computer Vision*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Springer, 2020, pp. 402–419.