Make it Dense: Self-Supervised Geometric Scan Completion of Sparse 3D LiDAR Scans in Large Outdoor Environments

Ignacio Vizzo Benedikt Mersch Rodrigo Marcuzzi Louis Wiesmann Jens Behley Cyrill Stachniss

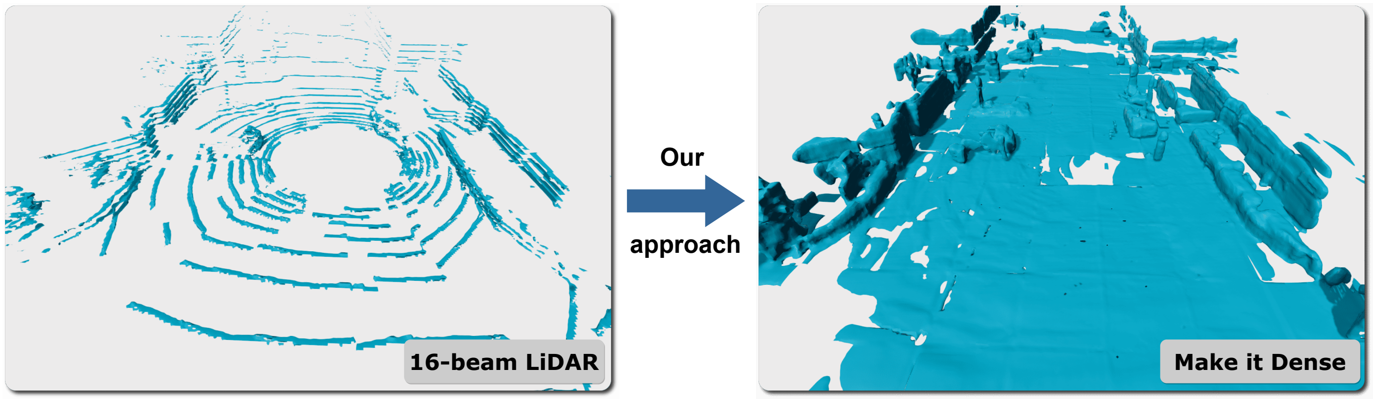

Fig. 1: A TSDF-based surface model from a single 16-beam LiDAR scan (left) turned into a denser, completed TSDF-based surface model (right) by the learning-based approach proposed in this paper.

***Abstract*—Mapping systems that turn sensor data into a model of the environment are standard components in mobile robotics. Outdoor robots are often equipped with 3D LiDAR sensors to obtain accurate range measurements at a high frame rate. The price for a robotic LiDAR sensor scales roughly linearly with the number of beams and thus the vertical resolution of the scanner. In general, the cheaper the sensors, the sparser the point cloud. In this paper, we address the problem of building dense models from sparse range data. Instead of requiring the vehicle to move slowly through the environment or to traverse the scene multiple times to cover the space densely, we investigate geometric scan completion through a learning-based approach. We revisit the traditional volumetric fusion pipeline based on truncated signed distance fields (TSDF) and propose a neural network to aid the 3D reconstruction on a frame-to-frame basis by completing each scan towards a dense TSDF volume. We propose a geometric scan completion network that is trained in a self-supervised fashion without labels. Our experiments illustrate that such frame-wise completion leads to maps that are on-par or even better compared to maps generated using a higher resolution LiDAR sensor. We additionally show that our system can be used to improve the performance of SLAM systems.**

***Index Terms*—Mapping; AI-Enabled Robotics**

Manuscript received: February 23, 2022; Revised: May 16, 2022; Accepted: June 11, 2022. This paper was recommended for publication by Editor Javier Civera upon evaluation of the Associate Editor and Reviewers' comments.

This work has partially been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy, EXC-2070 – 390732324 – PhenoRob.

All authors are with the University of Bonn, Germany. C.S. is additionally with the Dept. of Engineering Science at the University of Oxford, UK.

Digital Object Identifier (DOI): see top of this page.

I. Introduction

MOST autonomous vehicles rely on some form of mapping. Modern outdoor robots and self-driving cars are equipped with 3D sensors such as RGB-D cameras or LiDARs to perceive their surroundings. Volumetric mapping pipelines have shown to be an effective approach to mapping [7], [20], [36]. Most of the existing volumetric mapping works rely on the fact that they integrate dense 3D data at a rather high frame rate into a model, often at 10 to 30 frames per second (fps).

Outdoor robots often use 3D LiDARs such as Velodyne or Ouster scanners, which have between 16 and 128 beams. Their price scales roughly linearly with their number of beams and thus the vertical resolution of the scans. The denser a single scan, the more expensive the scanner. Thus, it is a relevant research question whether we can build a mapping or SLAM system that generates dense models but only requires sensors with a low vertical (or horizontal) resolution. The left image of Fig. 1 depicts the TSDF model computed from a single 16-beam LiDAR with a state-of-the-art mapping system [36]. As can be seen, the data and thus the model is rather sparse. As a result, it can be challenging for standard mapping pipelines to build dense models from such spare measurements, especially if the vehicle is driving at high speeds or the LiDAR scanner is operating at a low frame rate. Additionally, this spareness of the sensor data can lead to wrong pose estimation results of the SLAM system.

Recent work has shown the importance of obtaining a dense observation of the environment without the need to accumulate multiple frames [37]. The ability to predict how the environment looks beyond the current observation can be exploited for robotic tasks such as navigation [37], thus, being able to complete the geometry of single observations is also a relevant (and interesting) research question.

This paper addresses the problem of completing sparse 3D LiDAR scans in a frame-to-frame fashion for TSDF mapping using a learning approach. Fig. 1 illustrates our central question: "Based on data obtained from a single scan of a 16-beam LiDAR, can we estimate and hallucinate how the scene looks like?" For investigating this, we revisit the traditional volumetric fusion idea [7], [20] and combine it with a learning approach. We aim at exploiting the best of both worlds. In contrast to recent work on scan completion [10], [19], [37], we target single geometric scan completion in large outdoor environments where existing completion approaches fail to operate due to memory limitations of commonly used GPUs.

The main contribution of this paper is a self-supervised approach for turning a sparse 3D LiDAR scan into a comparably dense TSDF representation of the local scene. We propose a 3D convolutional neural network (CNN) trained in a self-supervised manner that completes the reconstructed scene on a frame-to-frame basis. In contrast to recently published works [10], [19], [37], we aim at completing *single scans* instead of completing a scene created from aggregated scans offline. In our approach, we process the 3D LiDAR data and pass it to our CNN. The output of the network is a TSDF representation that encodes the most recent observation plus synthetically completed data, which is then fused into a global map. Our experiments show that our approach can complete sparse LiDAR scans improving the mapping results, as well as a state-of-the-art SLAM pipeline [3]. We see this as a step towards getting better representations with cheaper scanners and thus reducing hardware costs through smarter software. We also publish our code together with the pre-trained models[1].

II. Related Work

3D scene reconstruction and understanding have been an active area of research for the last three decades. The classical volumetric integration method introduced by Curles and Levoy [7], made the use of the truncated signed distance functions popular in computer graphics, computer vision, and robotics applications. With KinectFusion, Newcombe et al. [20] popularized the use of volumetric integration methods [22], [24], [39], [36].

With the recent advances in LiDAR technology, the use of TSDF for volumetric mapping also gained increased attention in the research community. However, the sparsity of the sensor data and the volume that one scan covers makes it challenging to use such representation in outdoor environments with basic data structures like dense voxel grids. Some works specifically address the memory consumption, e.g., voxel hashing [21], [22], rolling grids [39], octrees [15], [32], or more recently using VDBs [36].

[1]https://github.com/PRBonn/make_it_dense

With the recent developments in deep learning, many approaches have been developed to solve 3D reconstruction [17], [25], [26], [27] and scene completion [8], [10]. However, most approaches only consider relatively small objects (from ShapeNet [5]) and do not apply to the setting we target, namely large outdoor scenes.

More recently, learning-based approaches have been proposed for RBG-D sensors that improve the volumetric aggregation in a TSDF [38] or learn to complete or improve the appearance of the generated reconstruction [10], [18], [19]. Our work is related to the work of Dai et al. [10] as well as Atlas [19]. In contrast to these methods, which work on the aggregated volumes, we target the single scan setting to avoid the time-consuming buffering of scans.

With the availability of large, annotated LiDAR datasets [2], new approaches have been proposed for *semantic* scene completion. Typically, these approaches require a rather large amount of *training* data to produce reasonable results [28], [29]. Additionally, adapting such systems to target geometry-only predictions is not straightforward due to the complexity of the network architectures. Instead, we aim at producing a smooth surface representation of the map. Additionally, our geometric scan completion does not require any labels, allowing our system to be trained from pure real-world data.

Furthermore, semantic scene completion systems typically require the input data to be fitted into a fixed-size volume, to cope with memory limitations, typically $256\times256\times32$ voxels [28], [29]. One common drawback of such design choice is that all the information on the negative x axis is discarded, being of importance for mapping applications but of not much relevance for semantic scene completion. Additionally, such methods use only a voxel size of 0.2 m, while we can produce a higher resolution model by picking 0.1 m, instead. In contrast to those works, we do not assume the size or the volume to be processed, allowing us to entirely complete a full outdoor-like TSDF volume. It might be tempting at a first sight to see our approach as a subset of the semantic scene completion task. However, this is not the case as our contribution is to produce denser semantic-free, geometric models using low-resolution scanners, also without making any assumption on the size of the input data.

In sum, we propose a geometry-driven system that integrates sparse LiDAR scans into volumetric maps by completing each scan using a 3D CNN. Our approach aims at taking the best of both worlds: classical mapping and deep learning.

III. Single Geometric Scan Completion

Our approach creates a dense reconstruction of single LiDAR scans recorded with a low vertical resolution, meaning few (e.g., 16) beams, to obtain results similar to those recorded with a larger number of beams (e.g., 64). We target processing every single scan as it gets recorded and we do not target a post-processing solution as Dai et al. [10] for RGB-D mapping or Vizzo et al. [35] for LiDAR mapping. Furthermore, this work does not focus on pose estimation, but only on the mapping part.

Fig. 2 shows an overview of our processing pipeline that combines classical mapping with a learning-based refinement

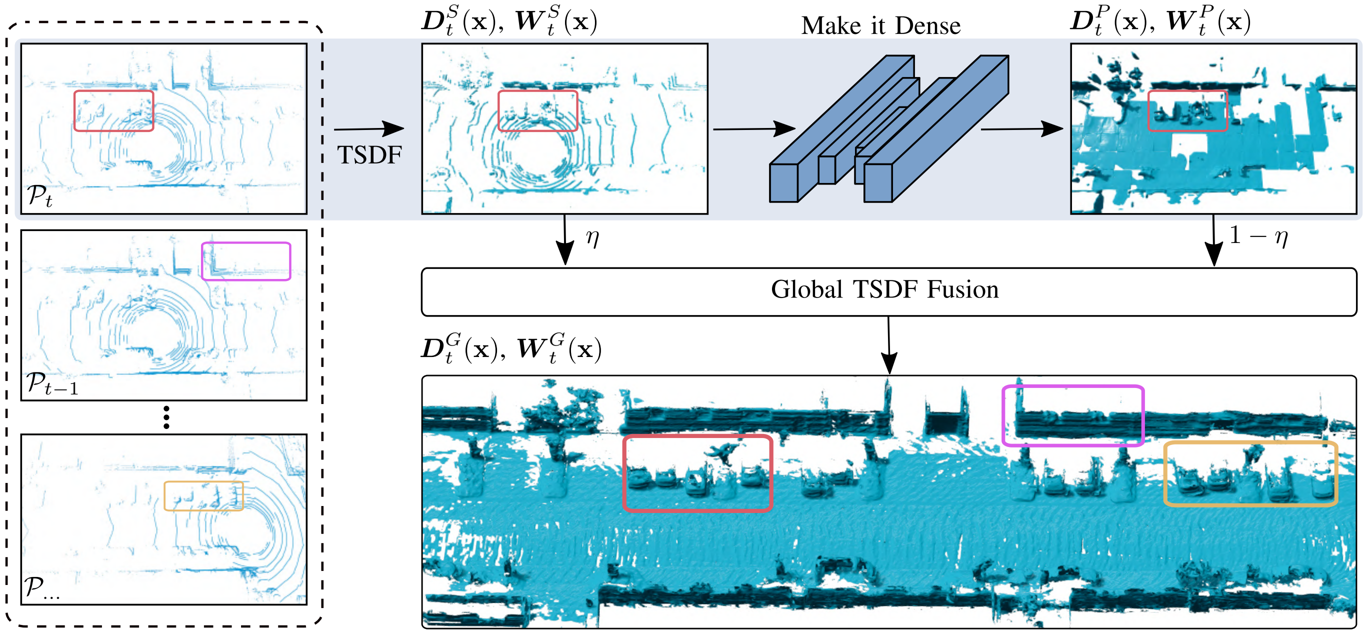
$\mathcal{P}_t$
$\mathcal{P}_{t-1}$
$\mathcal{P}_{\dots}$
TSDF
$\boldsymbol{D}_t^S(\mathbf{x}), \boldsymbol{W}_t^S(\mathbf{x})$
Make it Dense
$\boldsymbol{D}_t^P(\mathbf{x}), \boldsymbol{W}_t^P(\mathbf{x})$
η
$1-\eta$
Global TSDF Fusion
$\boldsymbol{D}_t^G(\mathbf{x}), \boldsymbol{W}_t^G(\mathbf{x})$

Fig. 2: Overview of our approach. We first generate a TSDF volume $\boldsymbol{D}_x^S(\mathbf{t})$ of a single scan $\mathcal{P}_t$ at time t. We then apply our geometric scan completion network in a strided fashion, such that missing values are added to the single-scan TSDF, giving a more complete TSDF volume $\boldsymbol{D}_x^P(\mathbf{t})$. The predicted TSDF values are then used to update the global TSDF representation $\boldsymbol{D}_t^G(\mathbf{x})$ using a weighting term η to avoid integrating the same observation twice into the map. The colored boxes highlight different areas of the input scans and its corresponding reconstruction in the final fusion results.

to generate high-fidelity representations from sparse LiDAR measurements. We first generate a TSDF-based volumetric representation of a single scan, which we denote as $\boldsymbol{D}_t^S(\mathbf{x})$, and (optionally) corresponding weights, denoted as $\boldsymbol{W}_t^S(\mathbf{x})$. Here, $\mathbf{x}$ refers to the voxel location in the grids. Then, we use $\boldsymbol{D}_t^S(\mathbf{x})$ to predict a new volume of TSDF values, denoted as $\boldsymbol{D}_t^P(\mathbf{x})$ using a fully convolutional network. The network is trained to fill in values in a self-supervised way as if the data would have been recorded with a high-resolution LiDAR.

As the grids are globally aligned, we integrate the scan-based predicted grid, $\boldsymbol{D}_t^P(\mathbf{x})$, into a global grid $\boldsymbol{D}_t^G(\mathbf{x})$. From the global signed distance fields, we can determine a surface representation using marching cubes [16]. In the following sections, we describe the individual processing steps in more detail.

A. *Scan Integration Using TSDF*

The LiDAR generates a point cloud $\mathcal{P}_t = \{\mathbf{p}_1, \dots, \mathbf{p}_N\}$ of N points $\mathbf{p}_i \in \mathbb{R}^3$ at time t. We assume that we have an estimate of the current pose of the sensor $\mathbf{T}_t \in \mathbb{R}^{4\times4}$ available (from a GPS/IMU combination). We denote by $\mathbf{R}_t \in \mathbb{R}^{3\times3}$ and $\mathbf{t}_t \in \mathbb{R}^3$ the rotational part and the translational part of the transformation $\mathbf{T}_t$, respectively.

To obtain the TSDF representation of the scan, we employ VDBFusion [36] with voxel size of v_{size}, truncation distance (also known as background value) of τ_{TD} and no space carving enabled [36]. We then extract the truncated signed distance field $\boldsymbol{D}_t^S(\mathbf{x})$ and the weight grid $\boldsymbol{W}_t^S(\mathbf{x})$ from the mapping system. We refer the curious reader to the original publication [36] for more details. As a result of this step, we end up with a volumetric scalar field $\boldsymbol{D}_t^S(\mathbf{x}) : \mathbb{R}^3 \to \mathbb{R}$ representing the TSDF.

B. *Geometric Scan Completion*

The TSDF grid $\boldsymbol{D}_t^S(\mathbf{x})$ encodes the surface representation of the single scan and is then passed through our CNN architecture that predicts a new TSDF grid $\boldsymbol{D}_t^P(\mathbf{x})$ with the same dimensions as $\boldsymbol{D}_t^S(\mathbf{x})$ to create the dense information. This is done for each incoming scan individually.

Network input. Instead of learning to regress a TSDF value directly from raw sensor data, we input a batch of TSDF volumes to the network and train it to improve its quality towards a more expensive sensor with more LiDAR beams. For this reason, the TSDF representation of the scan $\boldsymbol{D}_t^S(\mathbf{x})$, is split into non-overlapping volumes of $3.2\,\text{m}^3$. These volumes are batched together into a dense multidimensional tensor and then feed to our network. By storing the coordinates of the origin of each of these volumes, we can later reconstruct the original scene once the network has completed the TSDF values. Instead of using point-wise operations [34], [29], we can directly use 3D convolutions on these volumes. Since we operate on smaller sub-volumes from a single scan instead of a full scene, our network architecture is small and fast to train, making it rather compact and deployable on mobile robots. The fully convolutional design and the small number of parameters make it also hard for the network to overfit a particular dataset [23], [41]. Our network has about $1.4\,\text{M}$ parameters and already shows reasonable results after 1 hour of training.

Architecture. Our architecture is based on a 3D convolutional encoder-decoder architecture with skip connections between the output of each encoder stage and the input of the decoder stage with the same feature map dimensions. Our model is very similar to the architecture proposed by Dai et al. [10], with the difference that our architecture can process directly dense volumes, and therefore there is no need to convert back and forth between dense and sparse tensors. Analogously, our architecture is also similar to the Atlas [19] architecture, but in contrast, we do not fix the size of the volume but allow the scene to be arbitrarily large. In essence, our model reassembles a 3D-UNet [6], [30] architecture. More specifically, we first compute volumetric features using standard, dense 3D convolutions to increase the number of channels. For each subsequent encoder step, we increase the number of output channels by a factor of 2 using standard 3D convolutions and subsequently reduce the volumetric resolution with strided 3D convolutions. For upsampling, we use transposed convolutions to regain resolution in the output. Each convolution is followed by batch normalization and a ReLU activation.

Overall, we have a symmetrical encoder-decoder structure with $S = 3$ stages such that we achieve the same spatial resolution in the output as with the input voxel grid. Let $\mathcal{D}_i$ be the i^th^ decoder stage of a transposed convolution followed by a convolutional layer with an output dimension $M \times M \times M$ of the corresponding scalar field. Consequently, the output of the $\mathcal{D}_{i-1}$ stage is $M/2 \times M/2 \times M/2$ and therefore $\mathcal{D}_1$ denotes the first decoder stage after the last encoder stage which corresponds to a strided convolution followed by a convolutional layer.

We not only predict a scalar field after the final decoder stage, i.e., $\mathcal{D}_S$, but generate also intermediate outputs of the intermediate decoder stages $\mathcal{D}_1, \ldots, \mathcal{D}_{S-1}$. To transform the output of each decoder stage at every resolution level, we use convolutions with kernel size 1 to reduce the number of channels to 1 (the scalar field) and use $\tanh$ as an activation function to predict both positive and negative values.

Multi-Resolution Loss. We optimize our network end-to-end with a masked multi-resolution ℓ_1 loss between the predicted TSDF values (at all decoder stages, i.e., $\mathcal{D}_1, \ldots, \mathcal{D}_S$) and the target TSDF values. In line with prior work [9], [10], [19], [33], we log-transform the predicted and target values before applying the ℓ_1 loss. This enable us to obtain better predictions close to the surface [9], [10]. In contrast to recent works in scene completion [10], [37], we do not add any classification layer to predict invalid or occlusions.

At each decoder stage $\mathcal{D}_i$, we mask out regions where the predicted TSDF values are equal to the background value τ_{TD}. The following decoder stages will then skip the loss computation on those voxels and focus more on the fine-grained TSDF predictions at surface boundaries. Furthermore, we also mask out all the planes in the target volumes that are equal to the background value [19], which avoids artifacts on the predicted scalar field.

We denote by $\mathcal{R}_i$ all locations that are relevant for computation of the loss at decoder stage $\mathcal{D}_i$, since they are not already predicted by stage $i-1$ or masked out. Therefore, we use the following loss for the i-th resolution:

$$\mathcal{L}(\hat{D}_i, D_i) = \sum_{\mathbf{x} \in \mathcal{R}_i} ||\phi(\hat{D}_i(\mathbf{x})) - \phi(D_i(\mathbf{x}))||_1, \tag{1}$$

where $\hat{D}_i \in \mathbb{R}^{M \times M \times M}$ and $D_i \in \mathbb{R}^{M \times M \times M}$ correspond to the target TSDF volume and the predicted TSDF volume at the decoder stage i, respectively, and $\phi(x) : \mathbb{R} \to \mathbb{R}$ is the log transform, defined as

$$\phi(x) = \operatorname{sgn}(x) \cdot \log(|x| + 1) \tag{2}$$

Overall, we therefore minimize the following loss:

$$\mathcal{L} = \sum_{i \in \{1, \ldots, S\}} \mathcal{L}(\hat{D}_i, D_i). \tag{3}$$

Self-Supervised Training. We train our geometric scan completion network self-supervised by using VDBFusion [36] on the scans of the KITTI odometry dataset [11] using the provided poses. To achieve self-supervision, we take the sequential scans with known poses and build the following TSDF representations. First, we generate the desired input for our network by using a subset of beams of the Velodyne HDL-64E sensor. To this end, we keep every 4^{th} row from a range-image representation of the original scan. While this input scan does not represent the exact sensor characteristics of a low-cost LiDAR sensor, e.g., a Velodyne VLP-16, it is close to it in terms of density.

Next, we generate the multi-resolution targets, i.e., $\hat{D}_1, \ldots, \hat{D}_S$, by applying the TSDF pipeline on the original scan, without dropping beams. To increase the density of the targets, we aggregate n_{past}=25 scans in the past and n_{future}=25 scans in the future for the current frame. Furthermore, to avoid dynamic objects corrupting the supervision data, we make use of SemanticKITTI labels [2] and remove dynamic objects from those scans that are being aggregated.

To train the network efficiently, we split the generated target TSDF representations into non-overlapping volumes of 32×32×32, 16×16×16, and 8×8×8 voxels, respectively. Due to the different voxel sizes for the three representations, each extracted volume spans a total of $3.2\,\text{m}^3$ in the volumetric space. At test time, we use non-overlapping contiguous volumes of 32×32×32 voxels, since this empirically provided the best results. Note that the completion artifacts visible in Fig. 1 are due to this choice but do not affect the final mapping results, as shown in Fig. 2, Fig. 4, and Fig. 5.

The varying speed of the vehicle can change the characteristics of the target representation, which serves as an additional data augmentation. Furthermore, the targets are not complete and include sensor noise and artifacts in the reconstruction. Nevertheless, our network can learn how to complete a sparse input scan in a coarse-to-fine fashion with a high level of detail and completeness.

C. Global Map Update

Once we have the two observations, one based on sensor data and the other one based on the geometric scan completion network, we proceed to integrate this information into our global volumetric grid $\boldsymbol{D}^G_t(\mathbf{x})$.

It is important to note that we need to provide a way to combine the two data sources without integrating the same observation twice. Additionally, the real TSDF measurements are sparse and incomplete but do not contain reconstruction artifacts. In contrast, the predicted TSDF volumes are denser and more complete but they hold reconstruction artifacts due to prediction errors in the network. To fuse these two sources,

	Approach / Sequence	00	01	02	03	04	05	06	08	09	10	Avg.
16-beams	Voxblox [22]	0.74	0.75	0.81	0.76	0.74	0.71	0.74	0.74	0.68	0.71	0.75
	TSDF [36]	7.01	7.20	7.83	6.93	7.03	7.02	7.25	6.84	6.75	7.05	7.20
	TSDF [36] + Upsample	14.08	14.10	15.83	13.97	14.12	14.21	14.65	13.66	13.76	14.55	14.10
	TSDF [36] + Morphological	13.39	13.68	14.32	13.11	13.33	13.76	13.81	13.14	13.72	13.67	13.68
	PSR [13]	15.38	15.19	16.09	15.20	15.27	15.37	15.60	14.98	14.94	15.67	15.19
	Our approach	**24.57**	**24.95**	**25.71**	**24.11**	**24.57**	**24.52**	**25.36**	**24.01**	**24.33**	**24.73**	**24.95**
64-beams	Voxblox [22]	13.35	13.88	14.66	13.30	13.52	13.54	14.07	12.84	12.89	13.72	13.88
	TSDF [36]	23.27	24.17	25.34	23.07	23.37	23.69	24.30	22.54	23.06	23.93	24.17
	TSDF [36] + Morphological	14.16	14.46	14.76	14.18	13.97	14.44	14.72	13.99	15.25	14.67	14.46
	PSR [13]	25.22	25.57	26.12	24.87	25.19	25.24	26.06	24.48	25.01	25.59	25.57
	Our approach	**33.44**	**33.70**	**34.31**	**33.09**	**33.29**	**33.53**	**34.64**	**32.79**	**33.78**	**33.58**	**33.70**

TABLE I: Intersection-over-union results of the single geometric scan completion experiment. First row exhibits the IoU for a 16-beam sensor, while second row shows the results for a 64-beam one. Sequences are taken for the KITTI odometry benchmark [11].

we introduce a weighting term $\eta \in [0, 1]$. The term η specifies how much more we want to trust an actual measurement over a predicted TSDF value.

In TSDF mapping using RGB-D cameras, one may use the weight matrix $\boldsymbol{W}_t^S(\mathbf{x})$ to control the update of the TSDF grid [20], [22], [4] and, for example, down-weigh certain measurements within a single image. This, however, is much less relevant for LiDAR scans where measurements have similar accuracy. Thus, we generally set $\boldsymbol{W}_t^S(\mathbf{x}) = 1$ for all cells next to the truncation distance τ_{TD} and 0 otherwise. We do the same for the weight for the predicted scan $\boldsymbol{W}_t^P(\mathbf{x})$. In case more knowledge about uncertainties are available, one can introduce different weights here.

We use the factor η to weight the actual observations and the predicted ones in a consistent manner, making sure information is not incorporated multiple times. We can effectively fuse both data sources without duplicating the observation at every timestamp by:

$$\Delta\mathbf{D}(\mathbf{x}) = \eta \boldsymbol{W}_t^S(\mathbf{x})\boldsymbol{D}_t^S(\mathbf{x}) + (1-\eta)\boldsymbol{W}_t^P(\mathbf{x})\boldsymbol{D}_t^P(\mathbf{x}) \quad (4)$$

$$\Delta\mathbf{W}(\mathbf{x}) = \eta \boldsymbol{W}_t^S(\mathbf{x}) + (1-\eta)\boldsymbol{W}_t^P(\mathbf{x}). \quad (5)$$

Intuitively choosing a η factor close to 1 will completely discard the prediction of the network while keeping the real TSDF values, producing fewer artifacts on the output but having a sparse and incomplete model of the scene. In contrast, picking η close to 0 will reject entirely the real TSDF measurements and keep only the prediction of the network, producing a dense map representation but with more artifacts on the reconstruction. In our work, we find empirically that a good compromise is to set η=0.7. Once we have the aggregated SDF values $\Delta\mathbf{D}(\mathbf{x})$ for the two sources and their respective weights $\Delta\mathbf{W}(\mathbf{x})$, we fuse both analogously to Curless and Levoy [7] for all voxels at location $\mathbf{x}$ as follows:

$$\boldsymbol{D}_t^G(\mathbf{x}) = \frac{\boldsymbol{W}_{t-1}^G(\mathbf{x}) \cdot \boldsymbol{D}_{t-1}^G(\mathbf{x}) + \Delta\mathbf{D}(\mathbf{x})}{\boldsymbol{W}_{t-1}^G(\mathbf{x}) + \Delta\mathbf{W}(\mathbf{x})} \quad (6)$$

$$\boldsymbol{W}_t^G(\mathbf{x}) = \boldsymbol{W}_{t-1}^G(\mathbf{x}) + \Delta\mathbf{W}(\mathbf{x}). \quad (7)$$

IV. Experimental Evaluation

The main focus of this work is to complete sparse LiDAR scans to aid in mapping the environment while navigating through it. The experiments are designed to showcase the capabilities of our geometric scan completion approach. We will see that we can estimate comparably dense TSDF representations from a single 16-beam LiDAR scan using purely self-supervised training for our network. We obtain improved maps fusing real observations and observations produced by a CNN. Furthermore, we also show that our approach improves the accuracy of existing SLAM systems [3] when low-resolution scanners are employed.

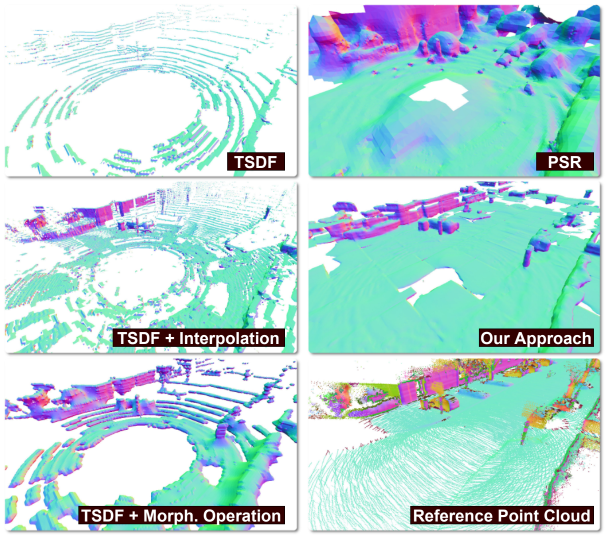

Fig. 3: Illustration of the results obtained through different geometric scan completion approaches. All representations were built with just one 16-beam LiDAR point cloud except for the GT point cloud used for evaluation.

A. Experimental Setup

We optimized our network with the Adam optimizer [14] and with an adaptive learning rate of 0.001. Our network takes only 1 hour to converge to reasonable results, but we keep training as long as 8 hours for fine-tuning some details. We use the KITTI odometry dataset [11] for evaluation and use labels provided by the SemanticKITTI dataset [1] to filter dynamic objects for the evaluation of the mapping results. We train our network on sequence 07 and report quantitative results on sequences 00 to 06 and 08 to 10. We use the GPS/IMU poses provided by the KITTI dataset. For all our experiments, we

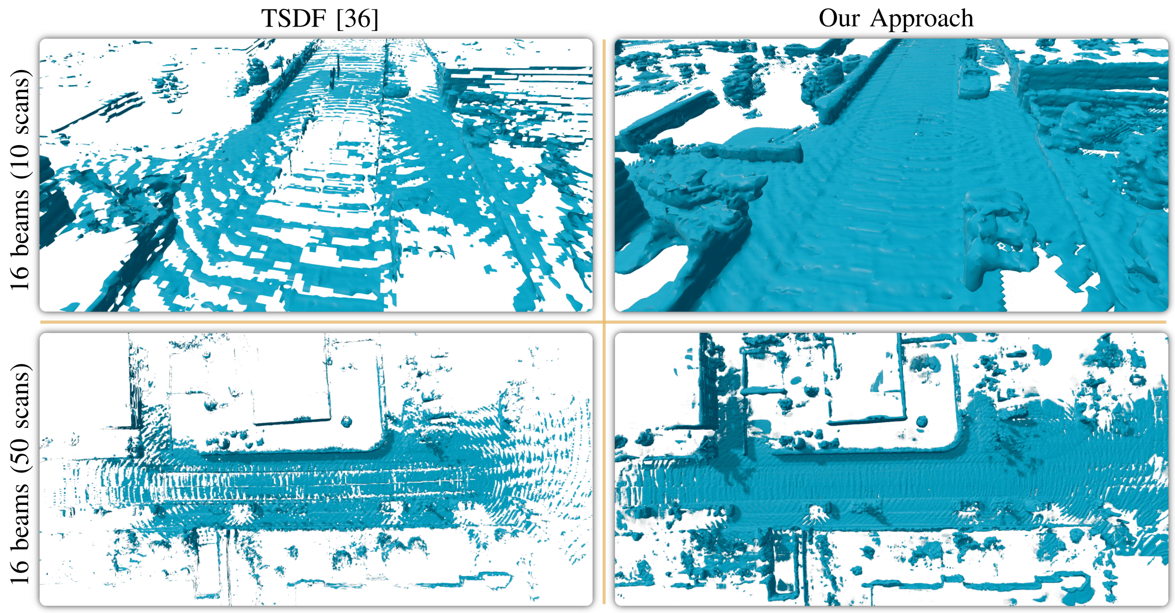

Fig. 4: Results of TSDF-based mapping (left) and our approach (right) after aggregating and subsequently fusing 10 (top row) and 50 (bottom row) scans from a 16-beam LiDAR. As can be seen, our approach yields a more complete model than standard TSDF mapping. The LiDAR frames are taken from sequence 03 of the KITTI odometry benchmark [11].

used a voxel size of $v_{\text{size}}{=}0.10\,\text{m}$ at the finest resolution. The truncated distance, also known as background value, is set to $\tau_{TD}{=}3$ voxels.

For quantitative evaluation, we use the standard scene completion metric proposed by Song et al. [31], which measures the intersection-over-union (IoU) between a ground truth voxel grid of occupied voxels and the predicted occupancies. Note that we account for areas that are never observed and are occluded and ignore these voxels.

B. *Completeness Achieved by Geometric Scan Completion*

The first experiment evaluates the performance of our approach for geometric scan completion and shows that we can predict a local scene in an urban environment using a single 16-beam LiDAR scan. As baselines for comparison, we use three other geometric approaches. First, a combination of morphological operations [12] such as opening and closing plus smoothing. Second, we use Poisson surface reconstruction (PSR) [13]. Third, we apply tri-linear interpolation on the range-image representation of the 16-beam LiDAR scanner to up-sample the point cloud to a 64-beam scan. This experiment is key to evaluate the performance of our approach as each scan is processed by the neural network and immediately incorporated into the mapping pipeline.

To use scene completion metrics [31] we need to obtain ground truth voxel grids that contain only occupancy information. To this end, we follow the approach of Behley et al. [2] and for each input scan at timestamp t, we aggregate $2L+1$ full-resolution scans (L before, L after, and the current scan) into a reference point cloud. We use then labels from SemanticKITTI [2] and filter out dynamic objects in this reference point cloud. Lastly, we crop the reference point cloud with the bounding box of the input scan under evaluation. We then determine the occluded voxels by raycasting the point cloud from the known poses, such that never observed voxels, i.e., voxels that cannot be reached by the raycasting, are marked as occluded. We then proceed to obtain occupancy voxel grids for the methods under consideration. To ensure that the IoU is not affected by the density of the point sampling [40], we first obtain a triangle mesh representation for each method. We sample a dense point cloud on these meshes and use it to obtain an occupancy voxel grid as described above. A qualitative illustration of this process is depicted in Fig. 3, and all methods visible in this figure will be compared to the reference point cloud. For the evaluation, we use every 500th scans from each of the test sequences and average errors across each sequence.

The results of this experiment are shown in Tab. I. The first row of Tab. I provides the quantitative results for all methods under consideration when using a single 16-beam LiDAR scan. This is the most relevant experiment for our work, as it shows how well we can complete a single 16-beam LiDAR observation. Nevertheless, we also run all the methods using a single 64-beam scan. Our approach outperforms all baselines in all the testing sequences for both sensor resolutions. It shows that we can complete sparse as well as dense scans.

To illustrate what such numbers mean, Fig. 3 shows a single scan and the completion results. The traditional TSDF-based method [36] can reconstruct the surfaces only where observations are available. Even when trying to augment the results of a traditional TSDF pipeline with morphological operations and upsampling, the results are not on-par with our approach. PSR [13] shown overall the best performance among the baselines, but as seen on Fig. 3 the method tends to over-smooth the reconstruction results. Our approach outperforms all baselines and provides accurate reconstructions close to the observations plus smooth and complete results for areas without measured points.

Method	Error	00	03	04	05	06	07	08	09	10	Avg.
SuMa (16 beams)	trans.	3.39	8.83	8.25	3.27	4.37	2.03	4.60	6.18	6.60	5.28
	rot.	1.62	4.53	1.32	1.53	2.10	1.66	2.10	2.52	2.32	2.19
Ours (16 beams)	trans.	**1.45**	**4.99**	**1.05**	**0.66**	**0.87**	**1.06**	**1.52**	**2.52**	**2.87**	**1.89**
	rot.	**0.63**	**1.60**	**0.85**	**0.37**	**0.44**	**0.83**	**0.73**	**0.73**	**1.08**	**0.81**

TABLE II: Pose estimates results on the KITTI dataset [11]. The relative translational error (trans.) is in % and the relative rotational error (rot.) in degrees per 100 m. All errors are averaged over trajectories of 100 to 800 m length. Bold numbers indicate the best approach for the given sequence. Sequences 01 and 02 are not reported due failure in both methods.

C. Improving Existing SLAM Systems using Low-Resolution LiDARs

This experiment showcases how our approach can be employed on existing 3D LiDAR SLAM pipelines, e.g. SuMa [3]. To conduct this experiment, we compare the pose estimates from the SLAM system using the standard metrics from the KITTI Odometry benchmark [11] with and without our densification pipeline. As a baseline, we run SuMa with the 16-beam LiDAR using the default parameters but changing the vertical resolution to 16 beams. For our approach, we first complete each scan individually, using our geometric scan completion network, and then project the completed scan into the vertex and normals maps needed for SuMa [3]. Note that given the high resolution of the completed scan, we have the freedom to choose any vertical resolution for the projection. For our experiments, we choose to use a vertical resolution of 64.

As shown in Tab. II the original SLAM system performs quite poorly when employing low-resolution LiDAR scanners. We overcome the spareness of the data by employing our geometric scan completion network and, as a result, we can improve the pose estimation on all sequences, except for sequences 01 and 02 where both methods fail, our results show to be almost 3 times better than the baseline.

D. Qualitative Evaluation for Mapping on Scan Sequences

16 beams. The next experiment illustrates how the presented self-supervised approach can aid a traditional volumetric reconstruction pipeline when considering whole *scan sequences* and not only individual scans. To do so, we illustrate the resulting local maps obtained when combing 10 and 50 consecutive 16-beam scans (with the scanner running at the maximum frame rate of 10 Hz). We compare our results against the most frequently used TSDF-based approach [36] that works on large-scale LiDAR data. As can be seen in Fig. 4, the traditional TSDF pipeline creates consistent maps, but the results are by far less complete when compared to our results. For example, the scene's street, sidewalks, cars, and other major shapes are properly reconstructed.

64 beams. The following experiment evaluates qualitatively how our system can improve the mapping results in cases where the frame rate of the scanner is relatively low or when driving at high speeds. We generate synthetic data from an urban sequence by dropping frames but keeping the full resolution of the scanner. We choose sequence 00 from the KITTI dataset [11] and run a traditional TSDF mapping pipeline [36] as baseline. As shown in Fig. 5, when the number of frames is relatively low, traditional integration methods fail to reconstruct a dense map of the environment. Our approach to geometric scan completion can help to cope with this type of situation improving the results of the mapping pipeline as well as its completeness.

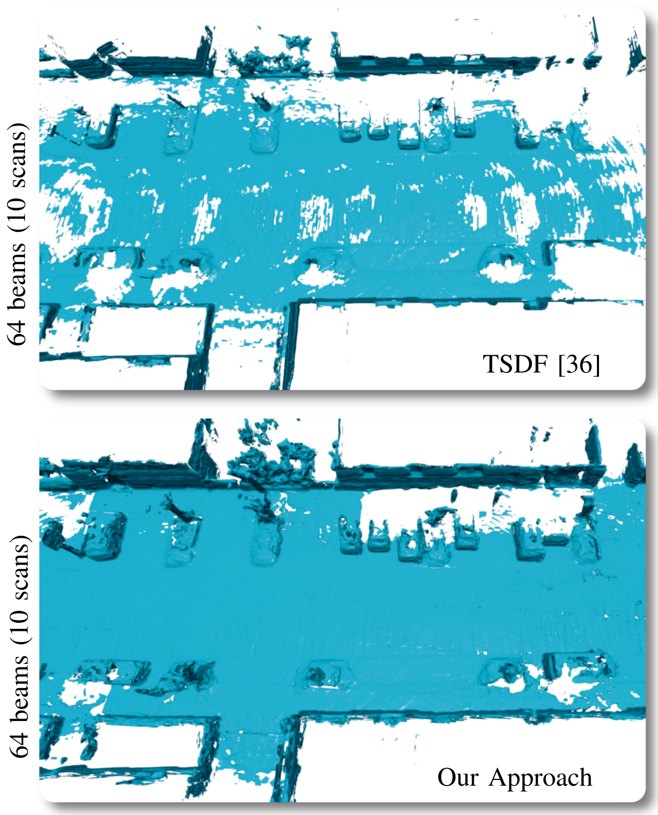

Fig. 5: Qualitative results of a high-speed/low frame rate LiDAR sequence. The LiDAR used for this experiments uses 64 beams, nevertheless classical integration methods [36] (top row) are not able to properly map the environment. Our geometric scan completion approach (bottom row) is able to aid the reconstruction system enabling a more complete representation of the scene.

V. Conclusion

In this paper, we proposed a novel mapping approach that builds dense 3D models from sparse LiDAR data. We investigate sparse 3D geometric scan completion through a learning-based approach. We combine traditional TSDF-based volumetric mapping with 3D convolutional neural networks to aid reconstruction on a frame-to-frame basis. From a single sparse scan, we can generate comparably dense TSDF surface models. Our completion network is trained in a fully self-supervised fashion. Our experiments suggest that our maps built with 16 LiDAR beams are on-par or even better compared to traditionally-built TSDF maps generated using 64-LiDAR beams.

The current limitation of our method is the network architecture, which does not exploit the sparse nature of the data, instead, it employs dense convolutions on the input volumes, which are generally not fully occupied. A natural extension is to replace the current architecture with one that exploits sparse convolutions to reduce memory consumption but also runtime operation at the same time.

REFERENCES

[1] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, J. Gall, and C. Stachniss. Towards 3d lidar-based semantic scene understanding of 3d point cloud sequences: The semantickitti dataset. *Intl. Journal of Robotics Research (IJRR)*, 40(8-9):959–967, 2021.

[2] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2019.

[3] J. Behley and C. Stachniss. Efficient Surfel-Based SLAM using 3D Laser Range Data in Urban Environments. In *Proc. of Robotics: Science and Systems (RSS)*, 2018.

[4] E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers. Real-Time Camera Tracking and 3D Reconstruction Using Signed Distance Functions. In *Proc. of Robotics: Science and Systems (RSS)*, volume 2, page 2, 2013.

[5] A. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[6] Ö. Çiçek, A. Abdulkadir, S.S. Lienkamp, T. Brox, and O. Ronneberger. 3d u-net: learning dense volumetric segmentation from sparse annotation. In *Medical Image Computing and Computer-Assisted Intervention*, 2016.

[7] B. Curless and M. Levoy. A Volumetric Method for Building Complex Models from Range Images. In *Proc. of the Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 303–312. ACM, 1996.

[8] A. Dai, A. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. ScanNet: Richly-Annotated 3D Reconstructions of Indoor Scenes. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[9] A. Dai, C.R. Qi, and M. Niessner. Shape Completion Using 3D-Encoder-Predictor CNNs and Shape Synthesis. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[10] A. Dai, C. Diller, and M. Nießner. Sg-nn: Sparse generative neural networks for self-supervised scene completion of rgb-d scans. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[11] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361, 2012.

[12] M.W. Jones, J.A. Baerentzen, and M. Sramek. 3D distance fields: A survey of techniques and applications. *IEEE Trans. on Visualization and Computer Graphics*, 12(4):581–599, 2006.

[13] M. Kazhdan and H. Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics*, 32(3):1–13, 2013.

[14] D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2015.

[15] T. Kühner and J. Kümmerle. Large-Scale Volumetric Scene Reconstruction using LiDAR. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2020.

[16] W. Lorensen and H. Cline. Marching Cubes: a High Resolution 3D Surface Construction Algorithm. In *Proc. of the Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 163–169, 1987.

[17] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[18] M. Mihajlovic, S. Weder, M. Pollefeys, and M.R. Oswald. Deepsurfels: Learning online appearance fusion. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[19] Z. Murez, T. van As, J. Bartolozzi, A. Sinha, V. Badrinarayanan, and A. Rabinovich. Atlas: End-to-end 3d scene reconstruction from posed images. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2020.

[20] R.A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A.J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-Time Dense Surface Mapping and Tracking. In *Proc. of the Intl. Symposium on Mixed and Augmented Reality (ISMAR)*, pages 127–136, 2011.

[21] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3D Reconstruction at Scale using Voxel Hashing. *Proc. of the SIGGRAPH Asia*, 32(6), 2013.

[22] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwar, and J. Nieto. Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 1366–1373, 2017.

[23] K. O'Shea and R. Nash. An introduction to convolutional neural networks. *arXiv preprint*, 2015.

[24] E. Palazzolo, J. Behley, P. Lottes, P. Giguere, and C. Stachniss. ReFusion: 3D Reconstruction in Dynamic Environments for RGB-D Cameras Exploiting Residuals. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2019.

[25] J.J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[26] S. Peng, M. Niemeyer, L. Mescheder, M. Pollefeys, and A. Geiger. Convolutional occupancy networks. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*. Springer, 2020.

[27] G. Riegler, A.O. Ulusoy, H. Bischof, and A. Geiger. Octnetfusion: Learning depth fusion from data. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[28] C. Rist, D. Emmerichs, M. Enzweiler, and D. Gavrila. Semantic scene completion using local deep implicit functions on lidar data. *IEEE Trans. on Pattern Analalysis and Machine Intelligence (TPAMI)*, pages 1–1, 2021.

[29] L. Roldão, R. de Charette, and A. Verroust-Blondet. Lmscnet: Lightweight multiscale 3d semantic completion. In *Proc. of the Intl. Conf. on 3D Vision (3DV)*, pages 111–119, 2020.

[30] O. Ronneberger, P.Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015.

[31] S. Song, F. Yu, A. Zeng, A. Chang, M. Savva, and T. Funkhouser. Semantic Scene Completion from a Single Depth Image. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[32] F. Steinbrücker, J. Sturm, and D. Cremers. Volumetric 3D Mapping in Real-Time on a CPU. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2014.

[33] J. Sun, Y. Xie, L. Chen, X. Zhou, and H. Bao. NeuralRecon: Real-Time Coherent 3D Reconstruction From Monocular Video. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[34] H. Thomas, C. Qi, J. Deschaud, B. Marcotegui, F. Goulette, and L. Guibas. KPConv: Flexible and Deformable Convolution for Point Clouds. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2019.

[35] I. Vizzo, X. Chen, N. Chebrolu, J. Behley, and C. Stachniss. Poisson Surface Reconstruction for LiDAR Odometry and Mapping. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2021.

[36] I. Vizzo, T. Guadagnino, J. Behley, and C. Stachniss. Vdbfusion: Flexible and efficient tsdf integration of range sensor data. *Sensors*, 22(3), 2022.

[37] L. Wang, H. Ye, Q. Wang, Y. Gao, C. Xu, and F. Gao. Learning-Based 3D Occupancy Prediction for Autonomous Navigation in Occluded Environments. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2021.

[38] S. Weder, J.L. Schonberger, M. Pollefeys, and M.R. Oswald. Neuralfusion: Online depth fusion in latent space. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[39] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J.J. Leonard, and J. McDonald. Real-time large scale dense RGB-D SLAM with volumetric fusion. *Intl. Journal of Robotics Research (IJRR)*, 34(4-5):598–626, 2014.

[40] L. Wiesmann, A. Milioto, X. Chen, C. Stachniss, and J. Behley. Deep Compression for Dense Point Cloud Maps. *IEEE Robotics and Automation Letters (RA-L)*, 6:2060–2067, 2021.

[41] A. Zhang, Z.C. Lipton, M. Li, and A.J. Smola. Dive into deep learning. *arXiv preprint*, 2021.