2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

# Large-Scale Volumetric Scene Reconstruction using LiDAR

Tilman Kühner[1] and Julius Kümmerle[1]

***Abstract*— Large-scale 3D scene reconstruction is an important task in autonomous driving and other robotics applications as having an accurate representation of the environment is necessary to safely interact with it. Reconstructions are used for numerous tasks ranging from localization and mapping to planning. In robotics, volumetric depth fusion is the method of choice for indoor applications since the emergence of commodity RGB-D cameras due to its robustness and high reconstruction quality. In this work we present an approach for volumetric depth fusion using LiDAR sensors as they are common on most autonomous cars. We present a framework for large-scale mapping of urban areas considering loop closures. Our method creates a meshed representation of an urban area from recordings over a distance of 3.7 km with a high level of detail on consumer graphics hardware in several minutes. The whole process is fully automated and does not need any user interference. We quantitatively evaluate our results from a real world application. Also, we investigate the effects of the sensor model that we assume on reconstruction quality by using synthetic data.**

## I. Introduction

In autonomous driving, 3D reconstruction is used for various tasks. A 3D representation of the environment can be used for localization. Most approaches use point clouds to represent the world and match current measurements against it ([1]–[3]). Other approaches like [4] use image alignment within a textured mesh to estimate the sensor pose.

Label transfer is another application that becomes more and more important as bigger datasets are created with more precise labels such as [5] and [6]. Costs for these datasets are enormous as images are manually labeled. The idea of label transfer is to infer labels for pixels within images by linking them to corresponding pixels in other images using a model of the environment or to directly label in 3D and project labels into images. This has been done using CRFs in [7]–[9] speeding up labeling time significantly.

Simulation is an important tool in developing and testing new algorithms and will also be needed for certification of autonomous cars. Numerous simulation environments such as presented in [10] exist. However, simulated scenarios are made up of hand modeled objects such as buildings or vegetation that often lack fine details and the complexity of the reality. Simulating a specific route requires to accurately replicate the scene. 3D reconstructions allow to automate this process and make scenarios more realistic at the same time. Lastly, reconstructions can reliably provide information such as free space and occlusions to planning and behavior generation.

[1] The authors are with FZI Research Center for Information Technology, Research Department "Mobile Perception Systems", Karlsruhe, Germany. {kuehner,kuemmerle}@fzi.de

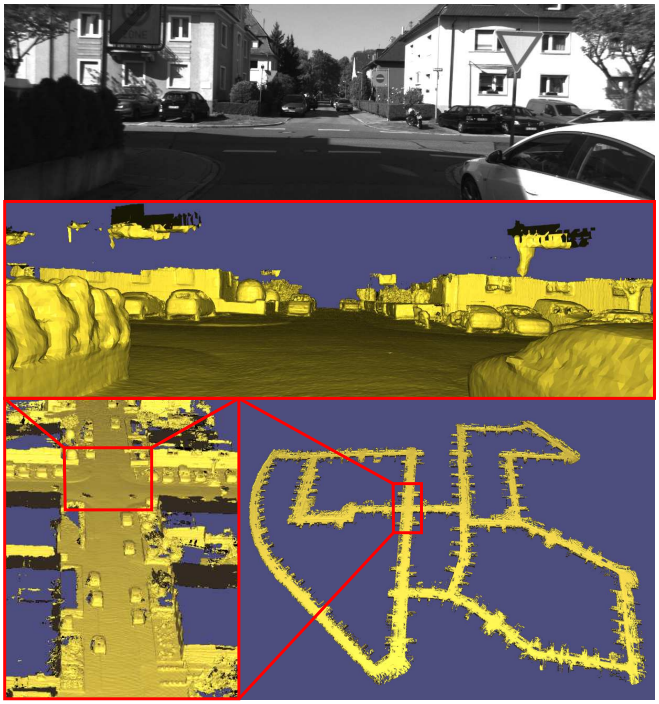


Fig. 1: Reconstruction of KITTI odometry sequence 00 with a closer look at an intersection that was passed multiple times during recording.

Multiple approaches exist that are used to represent the reconstructed scene. The simplest is to accumulate point clouds as it requires no fusion of individual sensor measurements and it is a by product of every LiDAR-based SLAM. SLAM is closely related to 3D reconstruction as it too builds up a model of the world. Examples are [1], [3] and [2]. The downside of accumulated point clouds is that they store a lot of redundant data which makes further processing inefficient. The point density varies vastly with sensor resolution and distant areas are sampled sparser than close areas. Also the reconstruction contains no topological information.

Another approach is to use surfels which are small surface patches. They are used in the work of [11] and [12]. Their advantage is that they, just as point clouds, do not need any spatial data structure while making it possible to fuse new measurements with previous ones. Despite their ability to represent surfaces their achieved reconstruction lacks the density of true volumetric approaches.

A third option is to represent the scene by discretizing space with a voxel grid. Occupancy grids as in [13] fuse probabilities about free and occupied space in each cell. While being

helpful for tasks like planning the reconstructions are often too coarse for other applications.

An approach for volumetric depth fusion was introduced by [14]. A voxel grid holds an implicit truncated signed distance function (TSDF) to the nearest surface. New sensor measurements can be fused by a recursive update scheme for voxels that project into the sensor. The method gained great popularity with the work of Newcombe et al. [15], powerful graphics cards and the emergence of cheap commodity depth sensors such as the Kinect. The method achieves dense reconstructions with a high level of detail while having the ability to filter out sensor noise. Further, the method is capable of reconstructing surfaces with sub-voxel accuracy by interpolating the signed distance function in between voxels.

A fixed size voxel grid only allows to reconstruct a relatively confined space such as a desktop due to memory limitations. Numerous approaches tried to overcome this limitation. The authors of [16] shift their voxel grid along with the sensor. Areas that leave the grid are transferred from GPU to the host computer. [17] employs multiple volumes which they call patch volumes that lie freely oriented in space. Other approaches like [18]–[20] use sparse representations such as octrees or similar hierarchical data structures. They consider the fact that most voxels in a dense grid lie outside the truncation distance and therefore do not hold any information. [21] introduces a hash function that maps the spatial coordinates of a voxel to its location in memory which is particularly GPU friendly as it avoids traversing deep tree structures.

Mapping larger areas requires to create consistent reconstructions in areas that are revisited or when loops are closed. [22] uses visual features to detect loops. When sensor poses are updated the corresponding measurements are de-integrated by fusing them with negative weight and then reintegrated with the new pose. In [17] the authors simply realign the individual patch volumes when a loop is closed. In [23] the same principle is employed and extended by transforming the patches nonrigidly.

Only few work exists that uses volumetric depth fusion in outdoor environments. In [24] and [7] depth from stereo cameras is fused. However, a single stereo setup only reconstructs half the scene as it is only seen from one side. Further, depth from stereo is of much less quality than from LiDAR.

In this work, we utilize volumetric depth fusion with LiDAR data by using a cylindrical projection model. We describe our mapping pipeline that handles loop closures and creates a consistent reconstruction of areas that are revisited multiple times during recording. Further, we analyze all relevant effects that have to be considered when using a rotating LiDAR sensor such as non single view point and rolling shutter. Lastly, we show results on publicly available data.

## II. Methodology

Fig. 2 shows the data flow in our framework and how data is shared between CPU and GPU. In the following sections we will explain the individual parts in the order of execution after a new point cloud $\mathcal{P}_i = \{\mathbf{p}_1 \dots \mathbf{p}_p\}_i$ is recorded.

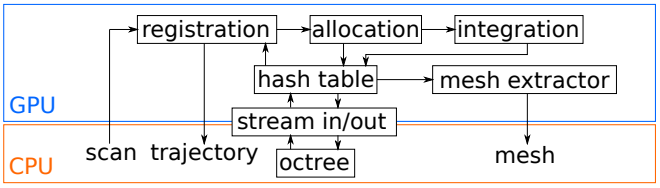



Fig. 2: Data flow in our framework. The GPU is used for fast computation while data has to be transferred to the host due to memory limitations of the GPU and for long time storage.

### A. Pose Estimation

First, for the new point in time $i$ the sensor pose

$$\mathbf{T}_i = \begin{bmatrix} \mathbf{R}_i & \mathbf{t}_i \\ \mathbf{0}^\top & 1 \end{bmatrix} \in \mathbb{SE}_3 \tag{1}$$

has to be determined relative to the world coordinate system that we define to be the first sensor pose $\mathbf{T}_0$ of a sequence. For pose estimation the effect from rolling shutter has to be compensated by deskewing the point cloud using the last pose increment as an estimate for the current sensor movement. A point to plane ICP as described in [25] is used thus solving the linear least squares problem

$$\boldsymbol{\xi}^\star = \arg\min_{\boldsymbol{\xi}} \sum_{(\mathbf{s},\mathbf{d},\mathbf{n})\in\mathcal{A}_i} ((\tilde{\mathbf{R}}_i\mathbf{s} + \mathbf{t}_i - \mathbf{d})^\top \mathbf{n})^2 \tag{2}$$

iteratively for the parameter vector

$$\boldsymbol{\xi} = \begin{bmatrix} \alpha, & \beta, & \gamma, & t_x, & t_y, & t_z \end{bmatrix}^\top \tag{3}$$

with the linearized orientation and translation

$$\tilde{\mathbf{R}} = \begin{bmatrix} 1 & -\gamma & \beta \\ \gamma & 1 & -\alpha \\ -\beta & \alpha & 1 \end{bmatrix}, \mathbf{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \tag{4}$$

and a set of point associations $\mathcal{A}_i = \{(\mathbf{s},\mathbf{d},\mathbf{n})_1 \dots (\mathbf{s},\mathbf{d},\mathbf{n})_a\}_i$ in which $\mathbf{s}$ denotes a source point from the current scan $\mathcal{P}_i$ and $\mathbf{d}$ denotes the corresponding destination point with normal $\mathbf{n}$. $\mathbf{d}$ and $\mathbf{n}$ are derived from the current reconstruction. Eq. 2 has the solution

$$\boldsymbol{\xi} = (\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top\mathbf{b} \tag{5}$$

with

$$\mathbf{A} = \begin{bmatrix} (\mathbf{s}_1 \times \mathbf{n}_1)^\top & \mathbf{n}_1^\top \\ \vdots & \vdots \\ (\mathbf{s}_a \times \mathbf{n}_a)^\top & \mathbf{n}_a^\top \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} (\mathbf{d}_1 - \mathbf{s}_1)^\top\mathbf{n}_1 \\ \vdots \\ (\mathbf{d}_a - \mathbf{s}_a)^\top\mathbf{n}_a \end{bmatrix}. \tag{6}$$

Registration is carried out in two steps. In a first step correspondences are found by projecting all source points into the sensor model using the model described in chapter II-B and the corresponding target points are determined by raytracing the world model as done in [15]. In a second step the pose is further optimized by directly minimizing the TSDF values at

all source points. For each source point its destination point is found using

$$\mathbf{d} = \mathbf{s} - F(\mathbf{s}) \frac{\nabla F(\mathbf{s})}{\|\nabla F(\mathbf{s})\|^2}, \tag{7}$$

as we define the truncated signed distances $F(\mathbf{x})$ stored in the voxels to be positive outside objects. The gradient is computed using central differences. In theory, $F(\mathbf{x})$ holds the distance to the nearest surface. Practically however the voxel grid does not hold a proper TSDF as parts of the grid have only been observed under low angles of incidence thus $F(\mathbf{x})$ tends to be too large. This leads to an overshoot of the true solution during ICP. To solve this issue, $F(\mathbf{x})$ in Eq. 7 is divided by the magnitude of the gradient to compensate for the effect.

The reason we carry out registration in two steps is that point to TSDF ICP has only a small area of convergence as the source points have to lie within the truncation distance to the surface. Therefore, an initial solution has to be found first using the projective correspondence search.

### B. Integration and Sensor Model

In order to fuse the range measurements of a LiDAR into the voxel grid an efficient way to assign a voxel to a measurement is needed. We use a cylindrical projection model with the cylindrical image plane wrapped around the rotation axis of the LiDAR as shown in Fig. 3. For most common LiDAR sensors the rays have no common point of intersection and vertical angle increments are not the same for all rays. We approximate a single viewpoint by placing the center of projection on the point on the rotation axis with the smallest sum of squared distances to all rays. We denote the offset of the center of projection $\mathbf{c}$ from the origin of the sensor's coordinate system by $c_z$. The projection function which maps a point $\mathbf{x} = [x, y, z]^\top$ in sensor coordinates to a pair of pixel coordinates $\mathbf{u} = [u, v]^\top$ is defined as

$$\pi(\mathbf{x}) = \left[ c_u - n_u\left(1 - \frac{\varphi}{2\pi}\right),\ c_v - \frac{(z - c_z) f_r}{\rho} \right]^\top \tag{8}$$

$$\rho = \sqrt{x^2 + y^2} \tag{9}$$

$$\varphi = \operatorname{arctan2}(y, x), \quad \varphi \in [0, 2\pi), \tag{10}$$

with $\rho$ being the projection of the point onto the xy-plane, the azimuth $\varphi$ and the number of columns $n_u$. Fig. 3 visualizes how a point is projected onto the image plane.

Since for most laser scanners the angular resolution differs significantly in horizontal and vertical direction this effect has to be taken into account. For the cylindrical projection model we choose the number of columns $n_u$ to be the same as the horizontal resolution of the LiDAR and $c_u$ in such a way that all points from $\mathcal{P}_i$ project into the center of a pixel. The number of rows $n_v$ is chosen in a way that makes the pixels approximately square. This leaves most pixels empty when projecting a scan onto the cylinder and would lead to an incomplete reconstruction. Therefore, each pixel that a scan point projects to assigns its point measurement to empty pixels within the same column up to a certain range which is approximately half the gap in between two projected scan lines in the image plane. The process is shown in Fig. 4. After this step we have associations between points $\mathbf{p}_j$ from $\mathcal{P}_i$ and pixels $\mathbf{u}_j$ they project to.

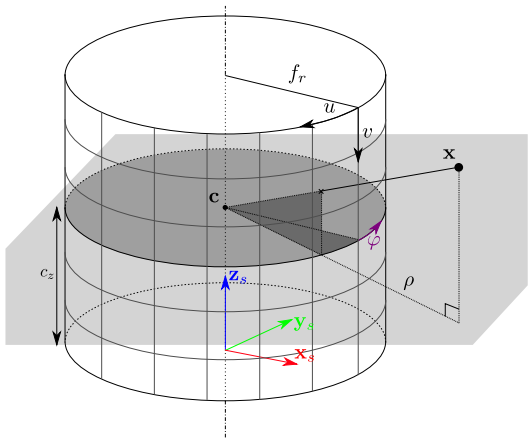



Fig. 3: Cylindrical projection model used for a rotating LiDAR.

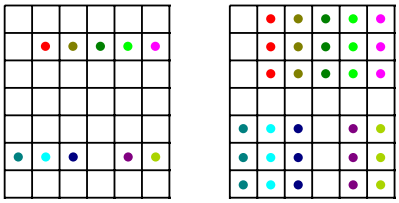


Fig. 4: 3D points are projected onto the cylinder (colored points) on the left. Pixels with projections fill empty pixels above and below with their point (right image).

We use voxel hashing as proposed in [21] as it is very efficient in memory consumption while being well suited for parallel implementations on GPUs. Before a scan is integrated into the voxel grid voxels within the truncation distance of the current scan have to be allocated in the hash table. [21] does so by checking all voxels along the rays of the sensor measurements. As a LiDAR scan is quite sparse, voxels at greater distances can be missed and stay unallocated which leads to holes in the reconstruction. Therefore, for each pixel $\mathbf{u}_j$ we allocate all voxels that are within its viewing frustum and within the truncation distance to $\mathbf{p}_j$ by shooting multiple rays through each pixel.

After allocation, all allocated voxels in the hash table are projected using the sensor model from Eq. 8. Their TSDF values $F(\mathbf{x})$ and weights $W(\mathbf{x})$ are updated with the recursive update rule from [14]

$$F(\mathbf{x})_i = \frac{W(\mathbf{x})_{i-1} F(\mathbf{x})_{i-1} + w(\mathbf{x}) f(\mathbf{x})}{W(\mathbf{x})_{i-1} + w(\mathbf{x})} \tag{11}$$

$$W(\mathbf{x})_i = W(\mathbf{x})_{i-1} + w(\mathbf{x}). \tag{12}$$

Voxels that project to $\mathbf{u}_j$ are updated with sensor measurement $\mathbf{p}_j$. $f(\mathbf{x})$ is the truncated signed distance from $\mathbf{p}_j$ to the voxel center point. We use the same weighting scheme as proposed in [26] to take a sensor measurement's footprint into account. The weight decreases the further the voxel

is away from the sensor measurement and weights fall off faster behind the sensor measurement reflecting a higher uncertainty.

During the recording of a full scan the sensor head rotates 360 ° and moves continuously with the vehicle. To consider this effect we add the option to continuously integrate measurements. The full scan is split up into contiguous fractions that are integrated separately while the sensor pose is adjusted assuming a linear motion. Ideally each column is integrated separately. However, this comes at greater computational cost as for each integration of a fraction all voxels have to be projected.

### C. Streaming and Mesh Extraction

Since most graphics cards have relatively few memory compared to its host device we stream voxels which leave the sensor's maximum viewing range to the host and voxels which enter the range are streamed to the GPU. On the host we store voxels in larger cubic blocks as leafs of an octree that dynamically grows with the space that is explored. One further advantage is that it can easily be serialized for long time storage and deserialized at a later time to integrate multiple drives.

After the last frame has been integrated into the voxel grid we stream all remaining voxels from the hash table to the octree. Then, we stream back large blocks from the octree to the hash table and extract the mesh on GPU. The surface is implicitly encoded in $F(\mathbf{x})$ as its zero iso-surface. We use marching cubes ([27]) for this step. The individual meshes are stitched together for the final result as shown in Fig. 1.

### D. Mapping

In order to consider loop closures in our mapping framework we run multiple stages to get the final reconstruction. First, we run all previous steps in odometry mode which means that we only stream out voxels and do not store them in the octree as this would create conflicts when a loop closure occurs after accumulating drift. We denote this trajectory with $\mathcal{T}_{\text{odom}}$. We then run the OpenFABMAP place recognition algorithm [28], which is an open source implementation of [29], to determine pose associations $\mathcal{L} = \{(\mathbf{T}_s, \mathbf{T}_d)_1 \dots (\mathbf{T}_s, \mathbf{T}_d)_l\}$. $\mathcal{T}_{\text{odom}}$ is then optimized such that associated poses coincide while changing the delta poses as little as possible by minimizing a least squares optimization problem for all poses in the trajectory $\mathcal{T} = \{\mathbf{T}_0 \dots \mathbf{T}_{n-1}\}$.

$$\mathcal{T}_{\text{loop}} = \arg\min_{\mathcal{T}} \sum_{i=0}^{n-2} \|\Delta\mathbf{t}_i - \Delta\mathbf{t}_{i\,\text{odom}}\|^2 + \alpha \measuredangle^2(\Delta\mathbf{R}_{i\,\text{odom}}^{\top} \Delta\mathbf{R}_i) + \beta \sum_{(\mathbf{T}_s, \mathbf{T}_d) \in \mathcal{L}} \|\mathbf{t}_s - \mathbf{t}_d\|^2 + \alpha \measuredangle^2(\mathbf{R}_d{}^{\top} \mathbf{R}_s) \tag{13}$$

with

$$\Delta\mathbf{R_i} = \mathbf{R_i}^{\top} \mathbf{R_{i+1}}, \quad \Delta\mathbf{t}_i = \mathbf{t}_{i+1} - \mathbf{t}_i. \tag{14}$$

$\alpha$ is a weighting factor to weight the influence of translational and rotational differences and $\beta$ weights the importance of closing loops against preserving the delta poses from odometry.

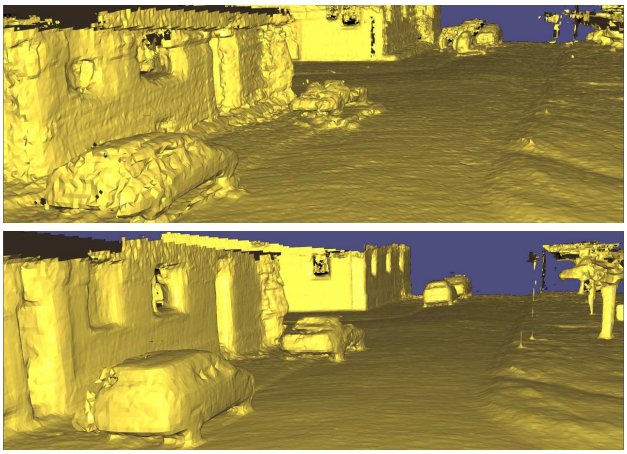


Fig. 5: Area that was passed three times during recording. Reconstruction from misaligned chunk poses (top) and after three iterations of pose refinement (bottom).

Using $\mathcal{T}_{\text{odom}}$, we create meshes by integrating measurements from a certain number of consecutive LiDAR scans. All meshes overlap by multiple scans. We convert the mesh vertices to a point cloud, which we call a chunk. Normals can be computed directly on the mesh. The chunks are initialized with $\mathcal{T}_{\text{loop}}$. Its purpose is to let the chunks overlap in areas where loop closures occur so point to point associations can be found between the chunks. Then, they are globally aligned using the LUM algorithm [30]. We use point clouds instead of the extracted meshes in this step since point to surface distances are much more expensive to compute than point to plane distances using point clouds. All poses within a chunk are rigidly transformed along with the chunk. The new trajectory is $\mathcal{T}_{\text{LUM}}$.

Poses will jump from the end of one chunk to the beginning of the next chunk due to the rigid transformation. Therefore, we iteratively refine $\mathcal{T}_{\text{LUM}}$. This is done in two alternating steps. First, all scans of the sequence are integrated into a clean voxel grid holding $\mathcal{T}_{\text{LUM}}$ fixed. Then, all sensor poses are realigned within the voxel grid using point to TSDF ICP as described in chapter II-B. A more consistent trajectory is obtained which is then used again to integrate all scans into a clean voxel grid and the process repeats. Performing this refinement step multiple times smooths out the discontinuities in the trajectory and improves reconstruction quality in areas that were visited multiple times as shown in Fig. 5. Since the first pose is also realigned during the process all poses have to be multiplied with the inverse of the first pose.

## III. Evaluation

We split this evaluation in two parts. The first part is about the quality of the reconstruction and how the sensor model influences its quality. The second part evaluates a real world mapping scenario on publicly available data.

### A. Reconstruction

In this experiment we evaluate the reconstruction quality of an object. We simulate a LiDAR moving around a car at a distance of 10 m with a velocity of 10 m/s at a height of 1.9 m above ground. The sensor records at a rate of 10 Hz. 63 scans are recorded capturing the model from all sides. We record two datasets. First we create synthetic LiDAR scans with zero noise and assume that the sensor obeys an ideal cylindrical projection model which means that all rays pass exactly through the center of each pixel and all rays originate from the projection center. Further, all scan points are recorded at one point in time for a full sensor head rotation thus eliminating effects from rolling shutter. We then simulate a real LiDAR sensor by adding Gaussian noise to the measured ranges with a standard deviation of $\sigma_r = 1.5\,\mathrm{cm}$. The orientation of all rays is taken from a Velodyne HDL-64 calibration file and the sensor moves continuously while scanning.

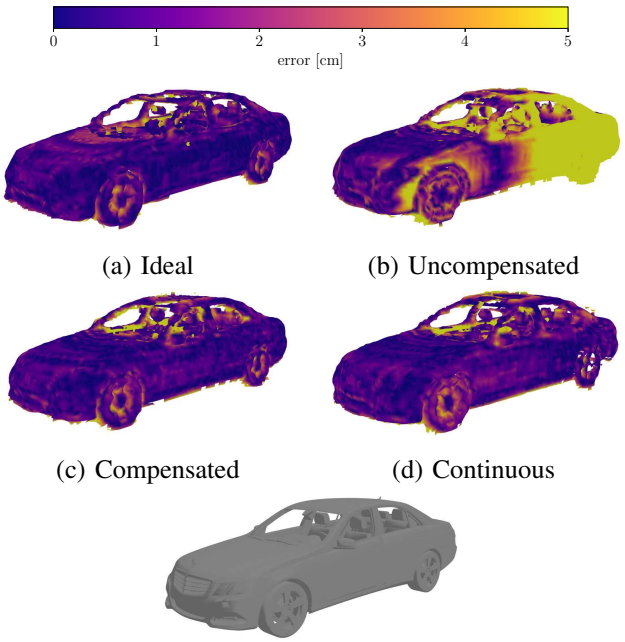
We use three different methods for reconstruction using sensor data from the realistic scenario and compare the results with the reconstruction from ideal data. In all cases we use the ground truth sensor poses. First, we neglect all effects from rolling shutter. Second, we compensate the point cloud for sensor movement by assuming a linear motion in between two poses. Lastly, we continuously integrate measurements into the voxel grid with the method described in chapter II-B. Each scan is split into ten fractions so the sensor moves only 10 cm in between each integration step. For all scenarios we use a voxel size of $l_v = 5\,\mathrm{cm}$.

For evaluation we use the metrics from [31] which are accuracy and completeness. Accuracy is defined as how close the reconstruction is to the ground truth mesh. It is computed by finding the distance from the ground truth mesh in which 90 % of the vertices of the reconstruction reside. The second metric is completeness which is a measure for how much of the ground truth is reconstructed. It is computed by defining an inlier distance $d$ that we set to $d = 5\,\mathrm{cm}$. Completeness is the fraction of vertices of the ground truth mesh that lies within $d$ from the reconstructed mesh.

The results are shown in Table I. Note that in all scenarios completeness is suffering from the fact that the sensor could not see the whole interior of the car neither was the bottom of the car visible during recording. As can be seen, not compensating for the sensors movement leads to drastically worse results than all other methods with less than half as good accuracy as well as approximately one third less completeness than the reconstruction that uses simple motion compensation. The extra effort of continuously integrating sensor measurements only partly pays off as it increases accuracy by 0.255 cm and completeness by 5.4 % compared to the computationally much cheaper integration of the compensated point cloud. Fig. 6 reflects these results as the two reconstructions almost look the same. Also visually as well as by comparing the error metrics the reconstructions on realistic data are close to the reconstruction from ideal data. This leads to the conclusion that errors from noise and the sensor model have only a marginal effect on the reconstruction and therefore our assumption to model a rotating LiDAR using a simple cylindrical projection model is valid. Actual sensor resolution and voxel size are the true limitations for reconstruction.

| Scenario | Accuracy [cm] | Completeness [%] |
|---|---|---|
| Ideal | 3.557 | 77.04 |
| Uncompensated | 11.184 | 47.85 |
| Compensated | 4.553 | 72.33 |
| Continuous | 4.298 | 77.73 |

TABLE I: Error metrics for reconstructed car model.




(a) Ideal (b) Uncompensated

(c) Compensated (d) Continuous

(e) Ground truth

Fig. 6: Distance of the reconstructed surface from the ground truth. Errors are cut off at 5 cm for better visualization.

By comparing the results to the ground truth it can be said that the method achieves a high level of detail with most parts on the outer surface having errors smaller than the range uncertainty $\sigma_r$. Fine structures such as rims, side mirrors and pillars are captured.

### B. Mapping

We use data from the KITTI odometry benchmark with known ground truth ([32]) to evaluate the localization result and show qualitative results of the reconstruction. For odometry we use $l_v = 10\,\mathrm{cm}$ and limit the sensor range to 50 m for performance reasons. We notice that a long sensor range improves odometry as far measurements constrain the pose better than close measurements while sensor noise is practically the same. For final integration after pose refinement we set $l_v = 5\,\mathrm{cm}$ and a sensor range of 20 m is used to limit the influence of pose errors on the reconstruction result.

Fig. 8 shows the initial trajectory from odometry in blue. The

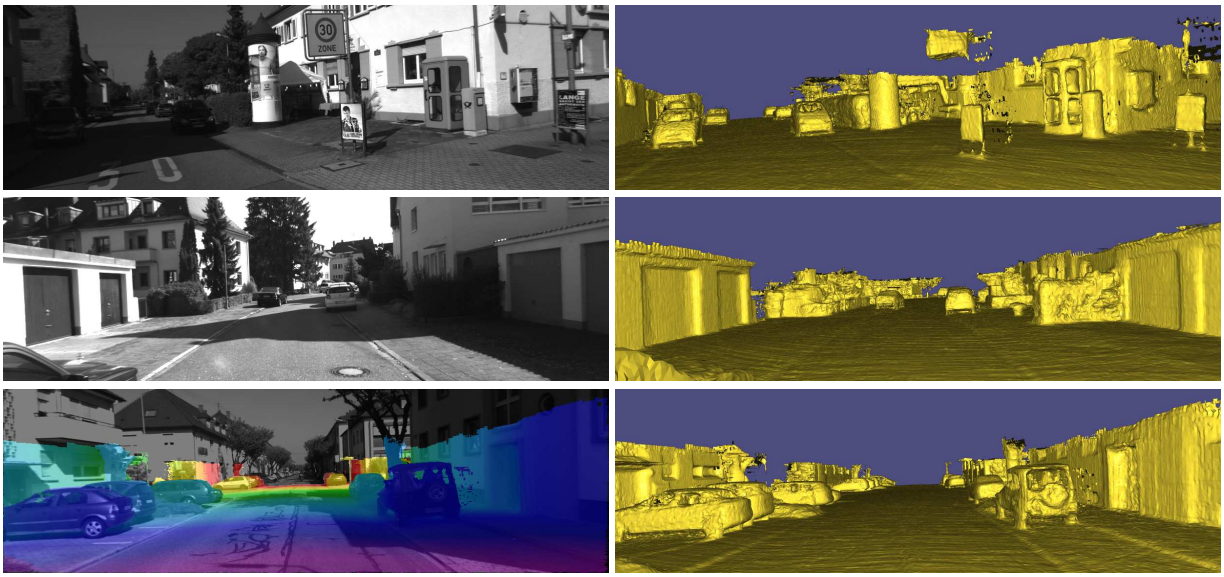


Fig. 7: Camera images and 3D reconstruction of the same scene side by side. Depth map from reconstruction is overlayed on the bottom left image.

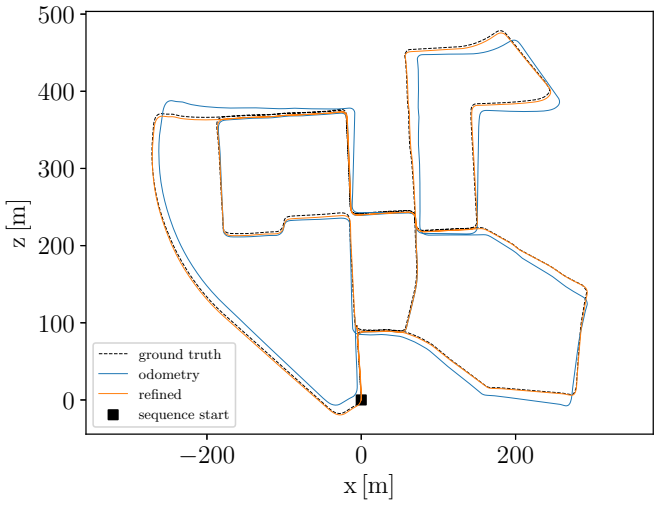



Fig. 8: Trajectories of KITTI odometry sequence 00 with a total length of 3.7 km.

final trajectory after closing loops and performing three iterations of pose refinement is shown in orange. The translation error is 2.79 % and the rotation error is 0.013 °/m. In Fig. 7 qualitative results are shown alongside the corresponding camera images. Fine structures such as shallow curbs that separate road from parking area are clearly visible in the mesh. In the last row of Fig. 7 the agreement of mesh and camera image is shown by overlaying the depth map from reconstruction with the image. Fine ripples on the ground in the direction of movement are caused by the relatively low vertical resolution of the LiDAR and the low angle of incidence of the rays hitting the ground. This causes voxels with vastly different ranges to be updated with the same range measurement. The effect also occurs on simulated data assuming an ideal sensor. Fig. 1 (top rows) as well as Fig. 7 (bottom rows) show scenes that were passed multiple times by the sensor thus inconsistent trajectories would show deteriorated reconstruction results. Similar results were achieved on other sequences in urban areas however odometry fails at some points on highways and country roads where there is not enough structure close to the vehicle.

## IV. Conclusions and Future Work

We proposed a method for large-scale scene reconstruction by fusing LiDAR measurements using a volumetric approach. We showed how to account for the rotating LiDAR's specific sensor characteristics and evaluated the effects of the sensor model as well as the whole framework on a real world mapping scenario. A high reconstruction quality could be achieved throughout the whole map and specifically in parts that were visited multiple times.

We also found limitations that come with the used method. Fine structures that are smaller than the voxel size are lost when seen from opposite sites during reconstruction. Therefore, thin poles and traffic signs are often missing in the reconstruction.

In the future, we plan to combine visual localization with volumetric fusion to further improve localization accuracy especially in areas where ICP fails. We further want to incorporate cameras in the reconstruction process not only for texture but also to capture structure that is below the resolution of most LiDAR sensors.

## REFERENCES


[1] J. Deschaud, "IMLS-SLAM: Scan-to-Model Matching Based on 3D Data," in *International Conference on Robotics and Automation (ICRA)*, 2018.

[2] M. Vlaminck, H. Luong, W. Goeman, and W. Philips, "3D Scene Reconstruction Using Omnidirectional Vision and LiDAR: A Hybrid Approach," in *Sensors*, 2016.

[3] F. Neuhaus, T. Koß, R. Kohnen, and D. Paulus, "MC2SLAM: Real-Time Inertial Lidar Odometry Using Two-Scan Motion Compensation," in *German Conference on Pattern Recognition (GCPR)*, 2018.

[4] G. Pascoe, W. Maddern, A. D. Stewart, and P. Newman, "FARLAP: Fast Robust Localisation using Appearance Priors," in *International Conference on Robotics and Automation (ICRA)*, 2015.

[5] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The Cityscapes Dataset for Semantic Urban Scene Understanding," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[6] P. Wang, X. Huang, X. Cheng, D. Zhou, Q. Geng, and R. Yang, "The ApolloScape Open Dataset for Autonomous Driving and its Application," in *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2019.

[7] S. Sengupta, E. Greveson, A. Shahrokni, and P. H. Torr, "Urban 3D Semantic Modelling Using Stereo Vision," in *International Conference on Robotics and Automation (ICRA)*, 2013.

[8] V. Garro, A. Fusiello, and S. Savarese, "Label Transfer Exploiting Three-dimensional Structure for Semantic Segmentation," in *International Conference on Computer Vision/Computer Graphics Collaboration Techniques and Applications*, 2013.

[9] J. Xie, M. Kiefel, M.-T. Sun, and A. Geiger, "Semantic Instance Annotation of Street Scenes by 3d to 2d Label Transfer," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[10] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An Open Urban Driving Simulator," in *arXiv preprint arXiv:1711.03938*, 2017.

[11] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb, "Real-time 3D Reconstruction in Dynamic Scenes using Point-based Fusion," in *International Conference on 3D Vision (3DV)*, 2013.

[12] T. Whelan, S. Leutenegger, R. Salas-Moreno, B. Glocker, and A. Davison, "ElasticFusion: Dense SLAM without a Pose Graph," in *Robotics: Science and Systems (RSS)*, 2015.

[13] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: A Probabilistic, Flexible, and Compact 3D Map Representation for Robotic Systems," in *International Conference on Robotics and Automation (ICRA) workshop on best practice in 3D perception and modeling for mobile manipulation*, 2010.

[14] B. Curless and M. Levoy, "A Volumetric Method for Building Complex Models from Range Images," in *Special Interest Group on Graphics and Interactive Techniques (SIGGRAPH)*, 1996.

[15] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. W. Fitzgibbon, "KinectFusion: Real-Time Dense Surface Mapping and Tracking," in *International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011.

[16] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald, "Kintinuous: Spatially Extended KinectFusion," in *Robotics: Science and Systems (RSS) Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2012.

[17] P. Henry, D. Fox, A. Bhowmik, and R. Mongia, "Patch Volumes: Segmentation-based Consistent Mapping with RGB-D Cameras," in *International Conference on 3D Vision (3DV)*, 2013.

[18] M. Zeng, F. Zhao, J. Zheng, and X. Liu, "Octree-based Fusion for Realtime 3D Reconstruction," in *Graphical Models*, 2013.

[19] F. Steinbrucker, C. Kerl, and D. Cremers, "Large-Scale Multi-Resolution Surface Reconstruction from RGB-D Sequences," in *International Conference on Computer Vision (ICCV)*, 2013.

[20] J. Chen, D. Bautembach, and S. Izadi, "Scalable Real-time Volumetric Surface Reconstruction," in *ACM Transactions on Graphics (TOG)*, 2013.

[21] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-Time 3D Reconstruction at Scale using Voxel Hashing," in *ACM Transactions on Graphics (TOG)*, 2013.

[22] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt, "BundleFusion: Real-time Globally Consistent 3D Reconstruction using On-the-fly Surface Re-integration," in *ACM Transactions on Graphics (TOG)*, 2017.

[23] N. Fioraio, J. Taylor, A. Fitzgibbon, L. Di Stefano, and S. Izadi, "Large-Scale and Drift-Free Surface Reconstruction using Online Subvolume Registration," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[24] I. A. Bârsan, P. Liu, M. Pollefeys, and A. Geiger, "Robust Dense Mapping for Large-Scale Dynamic Environments," in *International Conference on Robotics and Automation (ICRA)*, 2018.

[25] K.-L. Low, "Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration," in *Technical Report TR04-004, Department of Computer Science, University of North Carolina at Chapel Hill*, 2004.

[26] S. Fuhrmann and M. Goesele, "Floating Scale Surface Reconstruction," in *ACM Transactions on Graphics (TOG)*, 2014.

[27] W. E. Lorensen and H. E. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," in *Special Interest Group on Graphics and Interactive Techniques (SIGGRAPH)*, 1987.

[28] A. Glover, W. Maddern, M. Warren, S. Reid, M. Milford, and G. Wyeth, "OpenFABMAP: An Open Source Toolbox for Appearance-Based Loop Closure Detection," in *International Conference on Robotics and Automation (ICRA)*, 2012.

[29] M. Cummins and P. Newman, "FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance," in *The International Journal of Robotics Research (IJRR)*, 2008.

[30] F. Lu and E. Milios, "Globally Consistent Range Scan Alignment for Environment Mapping," in *Autonomous Robots*, 1997.

[31] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, "A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.

[32] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.