Poisson Surface Reconstruction for LiDAR Odometry and Mapping

Ignacio Vizzo   Xieyuanli Chen   Nived Chebrolu   Jens Behley   Cyrill Stachniss

***Abstract*— Accurately localizing in and mapping an environment are essential building blocks of most autonomous systems. In this paper, we present a novel approach for LiDAR odometry and mapping, focusing on improving the mapping quality and at the same time estimating the pose of the vehicle. Our approach performs frame-to-mesh ICP, but in contrast to other SLAM approaches, we represent the map as a triangle mesh computed via Poisson surface reconstruction. We perform the surface reconstruction in a sliding window fashion over a sequence of past scans. In this way, we obtain accurate local maps that are well suited for registration and can also be combined into a global map. This enables us to build a 3D map showing more geometric details than common mapping approaches relying on a truncated signed distance function or surfels. Our experimental evaluation shows quantitatively and qualitatively that our maps offer higher geometric accuracies than these other map representations. We also show that our maps are compact and can be used for LiDAR-based odometry estimation with a novel ray-casting-based data association.**

I. Introduction

Without a map of the environment as well as the knowledge about their pose, most autonomous systems cannot navigate effectively. Thus, localization, mapping, as well as simultaneous localization and mapping [4], [34] are important building blocks of autonomous systems.

We employ a rotating LiDAR sensor to map the environment and investigate the usage of an alternative scene representation for mapping and registration. Scene representation is essential since it is used to register incoming scans. To obtain accurate relative pose estimates and compelling mapping results, a scene representation must capture and represent the environment with a high level of detail.

Our paper aims at improving the geometrical accuracy of LiDAR-based mapping while simultaneously estimating the pose of the vehicle with low-drift over time. We achieve this by using a triangle mesh representation computed via the Poisson surface reconstruction technique [15]. This is in contrast to other state-of-the-art approaches which often use either surfels [1] or a truncated signed distance function [7], [23] as a representation, that often provide a rather low reconstruction quality, at least for large outdoor scenes. With our approach, we reconstruct meshes from robotic 3D LIDAR data for outdoor environments with a quality that was previously common only at the object level, for indoor scenes, by using terrestrial scanners, or by aggregating multiple passes over the same scene.

All authors are with the University of Bonn, Germany.

This work has partially been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy, EXC-2070 - 390732324 - PhenoRob and by the European Union's Horizon 2020 research and innovation programme under grant agreement No 101017008 (Harmony).

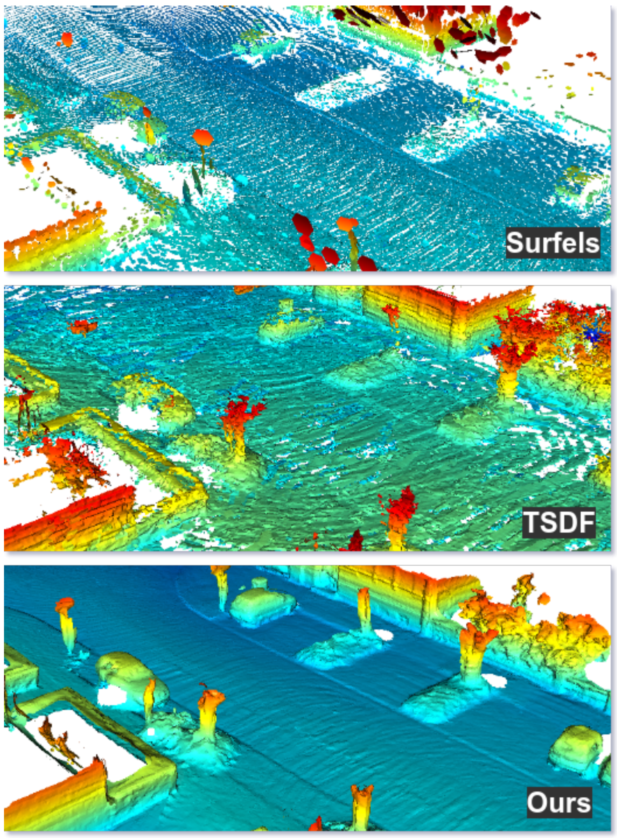

Fig. 1: Qualitative comparison between the different mapping techniques for sequence 00 of the KITTI odometry benchmark [11].

The main contribution of this paper is a novel LiDAR odometry and mapping system that builds upon a surface reconstruction method providing geometrical accurate maps. We aggregate individual scans into a local point cloud and use these to reconstruct a triangular mesh of the scene. Our experimental evaluation shows that such a triangle mesh is well-suited for the registration of 3D LiDAR scans as it is comparably compact, preserves rather detailed structures, and allows for accurate frame-to-mesh registration. This yields a new 3D LiDAR-based mapping approach that provides geometrically accurate maps and can be used for pose estimation, as shown in Fig. 1. We show that the proposed map representation (i) is a geometrically accurate representation of the environment, (ii) has better memory efficiency as compared to other map representations, and (iii) allows for accurate registration of the incoming scans with the model using a novel frame-to-mesh registration algorithm. We support these claims through experimental evaluation on synthetic and real-world data.

II. Related Work

Simultaneous localization and mapping (SLAM) has been investigated for decades [4], [34] and here we concentrate on 3D LiDAR-based approaches.

Laser-based SLAM systems either rely on sparse features [39], [22] or perform registration with a dense map representation [1], [8]. Deschaud [8] propose to use implicit moving least squares (IMLS) surfaces for the map representation. Behley and Stachniss [1] perform mapping with a surfel-based map representation. In contrast to these approaches, we register LiDAR scans to a dense map representation using triangular meshes instead of surfels or IMLS surfaces.

A common technique to obtain triangular meshes from point clouds is 3D surface reconstruction [2]. Traditional approaches determine an implicit function modelling the underlying surface, for example, using tangent planes [12], radial basis functions [5], truncated signed distance functions (TSDF) [7], or polynomial representations [17]. Poisson surface reconstruction [14], [15] provides geometrically accurate reconstruction based on this principle.

In contrast to feature-based SLAM systems, dense approaches aim to use all the input data and aggregate them into a dense map representation. A popular approach is using the aforementioned TSDFs popularized by Newcombe et al. [23] for RGB-D mapping. Most TSDF approaches need to know a priori the volume of the environment as they rely on a fixed voxel grid, but Whelan et al. [37] propose to use a rolling grid to alleviate this limitation and store the intermediate results in a triangular mesh. Other approaches use an octree [35] or allocate blocks on demand [24], [26]. Contrary to these methods, we do not make any assumption of the size of the environment to map or make use of any optimized data structures.

In robotics, the triangular mesh representation for reconstruction was employed by Marton et al. [19]. In recent years, such representations were also explored for visual-inertial systems [30], LiDAR-based approaches [6], [31] and purely vision-based systems [27]. In contrast to our work, these approaches typically compute a sparse reconstruction of the traversed environment, while we aim to reconstruct a continuous triangle surface capturing geometric details.

Also, 3D LiDAR-based approaches use variants of TSDF working towards larger volumes and building globally consistent maps [21], [25], [28], [29]. Kühner et al. [18] exploit a TSDF for large outdoor environments, but they need multiple passes over the same scene to obtain compelling reconstruction results. We propose a method that needs only a single pass to obtain a high level of detail.

III. Our Approach

We perform the following three steps for each scan: First, we compute per-point normals, second, we register the scan to the local map, and third, we fuse the registered scan into a global map. We furthermore propose a novel frame-to-mesh registration strategy that exploits that the map is represented by a triangle mesh.

Our approach distinguishes between a local map and the global map. The local map is used to perform the odometry estimation and is build from the last N localized scans. The global map is the aggregated mesh of the entire environment.

A. Normal Computation and Point Cloud Registration

For computing the normals, we project the point cloud into a range image using a spherical projection and estimate the normal vectors using the cross product from neighboring pixels [1]. While this is sometimes less accurate than estimating normals via principal component analysis on the covariance of a point neighborhood, it is far more efficient as it does not need to determine a point neighborhood.

For point cloud registration, we iteratively perform data association of the point cloud with the triangular mesh and determine pose increments to minimize an error metric.

For the data association between point clouds, the closest point association found via neighbor search or projectively [32] is a common choice. We can also use this strategy with our mesh representation by searching neighbors over the triangle vertices, but this is suboptimal as we will show in the experimental evaluation.

We propose to use ray-casting to determine ray-triangle intersections. For each intersection, we extract the point and associated normal of the intersected triangle. To this end, we first apply the last estimated pose at time $t-1$, i.e., $\mathbf{T}_{t-1} \in \mathbb{R}^{4\times4}$, to the current scan as an initial alignment. We then create a set of n rays $\mathcal{R} = \{\mathbf{r}_i\}$. Each ray $\mathbf{r}_i$ is defined by $\mathbf{o}_i + \tau\mathbf{d}_i$ with the origin $\mathbf{o}_i = \mathbf{t}_t^k$ at the currently estimated sensor position and directions $\mathbf{d}_i = \mathbf{T}_t^k\mathbf{p}_i$, passing through all points $\mathbf{p}_i$ of the current scan. Here, $\mathbf{T}_t^k$ is the estimated pose at iteration k and $\mathbf{t}_t^k \in \mathbb{R}^4$ is the translational part of $\mathbf{T}_t^k$.

The intersection of each ray $\mathbf{r}_i \in \mathcal{R}$ with the mesh results in the correspondence for point $\mathbf{p}_i$, denoted as $\mathbf{q}_i$, and the normal of the intersected triangle is the corresponding normal $\mathbf{n}_i$. To compute the relative transformation between the scan and the mesh, we can now use different error metrics $E(\cdot,\cdot)$, such as point-to-point, point-to-plane, or plane-to-plane error [3], [32], [33].

The data association step can also result in wrong correspondences, where a given point from a surface is associated with an intersected point in the mesh from another surface. This typically happens when the ray does not hit any close by surface and hits a far away triangle. Therefore as outlier rejection, we remove correspondences $(\mathbf{p}_i, \mathbf{q}_i) \in \mathcal{C}$ from the set of correspondences $\mathcal{C}$ that satisfy $||\mathbf{p_i} - \mathbf{q_i}|| > \delta_d$. In our implementation, we use $\delta_d = 1\,\mathrm{m}$.

To obtain the estimated pose of the vehicle at time t, we then optimize the following objective:

$$\mathbf{T}^* = \underset{\mathbf{T}}{\operatorname{argmin}} \sum_{(\mathbf{p}_i,\mathbf{q}_i)\in\mathcal{C}} E(\mathbf{T}\mathbf{T}_t^{k-1}\mathbf{p}_i, \mathbf{q}_i), \tag{1}$$

considering the error metric $E(\mathbf{p},\mathbf{q})$ between points $\mathbf{p},\mathbf{q}$ and an increment $\mathbf{T}_t^k = \mathbf{T}^*\mathbf{T}_t^{k-1}$ at iteration k. In our experiments, we found out that the best performing metric is the point-to-plane metric [32]. To reduce the influence of outliers that were not filtered by the outlier rejection, we use a Huber weighting kernel [13].

The main advantage of the proposed data association is that it does not need to compute nearest neighbors, instead, the association step exploits the map representation through ray-casting the mesh, which as shown in the experiments can be executed faster, especially when dealing with high-resolution meshes. Nonetheless, this approach does not properly handle large rotational motions and requires a good initial estimate to converge, even though this also holds for nearest neighbor approaches.

B. Meshing Algorithm

A common technique to performing 3D surface reconstruction with point sets is to build an implicit function that aims to recover the underlying surface of the input data [12]. Such implicit function f is usually defined as a scalar field in $\mathbb{R}^3$, i.e., $f : \mathbb{R}^3 \mapsto \mathbb{R}$, where typically the zero set of f represents the surface we aim at modeling. A popular technique in robotics and SLAM is to approximate f by the signed distance function [7], i.e., the projected distance, from the sensor to the surface.

Our work, in contrast, explores the use of Poisson surface reconstruction [14], [15] to build consistent, smooth, high-quality maps for mobile robots, in particular, for autonomous vehicles. We refer to the original publications [14], [15] for details on the reconstruction algorithm. Our goal is to investigate the use of triangular meshes in SLAM besides the particular choice of the algorithm being used for the reconstruction[1].

Next, we explain out mesh post-processing. The aforementioned Poisson reconstruction is designed to recover closed surfaces of a single object in 3D as illustrated in Fig. 2. Our 3D world, especially outside environments, is not composed of closed surfaces. Therefore, we need to refine the reconstructed surface and perform a post-processing step, which involves removing vertices with low density. The density $\delta(\mathbf{v})$ of a vertex $\mathbf{v}$ on the mesh measures how many points from the input point cloud support the vertex $\mathbf{v}$. Intuitively, a low value means that the vertex is only supported by a low number of points, and therefore, is not densely measured in the original LiDAR scan or not measured at all (as the Poisson surface reconstruction algorithm will also extrapolate points where there is no data). After reconstructing the mesh, we compute the distribution of the per-vertex densities, as illustrated by the histogram in Fig. 2, right to the legend. The vertices of interest have a high density, i.e. those that are closer in space to point cloud data and are colored in yellow to red in the figure. We trim away the low-density vertices independently of the size of the triangles of the mesh. We make that decision only based on the density $\delta(\mathbf{v})$ of a vertex. We consider the cumulative histogram of densities starting with the highest density values and trim those vertices that belong to the last 10%. This means we trim away 10% of the vertices with the smallest density values.

[1]We only consider horizontally placed LiDAR sensors, therefore, we do not examine LiDARs shooting towards the sky, like profiler scanners.

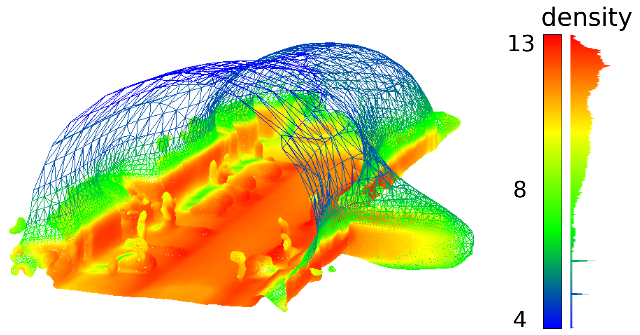

Fig. 2: Cross-section view of the vertex densities for an urban scene reconstruction, where the colors represent the density of the vertices, as shown in the colorbar. We trim away 10% of the vertices based on the density threshold $\delta(\mathbf{v})$.

This post-processing produces a tighter reconstruction of the input data, showing little artifacts, which allow us to incrementally build the global mesh as described in Sec. III-C. Note that without this step, it is not possible to build a global map of the environment, due to the artifacts shown in blue in Fig. 2.

Note that, as an interesting side-effect, this density-based filtering also tends to eliminate most of the moving objects in the scene, since 3D points on the surface of moving objects often only support a small number of triangles as the position of the surface changes in every scan. This leads to low-density triangles on moving objects and thus no surface will be reconstructed at these locations.

C. Local and Global Map

In our approach, we distinguish between a local and a global map. The local map is built from the last N aggregated scans. The global map is only used for visualization and for reporting the final output but is not used inside our approach, which will change in case we add loop closing. A new mesh is reconstructed from the local map each time a new LiDAR frame has been registered to the local map. This produces a rolling grid-like mesh that moves with the estimated pose of the vehicle and stores enough information for the registration of new incoming scans. During the initial N scans, we disable the mesh reconstruction module and rely on standard point-to-plane ICP for estimating the pose of the vehicle. After M scans have been registered, the last generated local mesh is integrated into the global mesh map. This means that the global mesh will be updated only after M scans have arrived and registered (in contrast to the local mesh that is updated every time a new scan arrives). To do so, we add all the triangles in the local mesh to the global one and then we remove the duplicated triangles that can occur due to overlaps in the local map region. In our implementation we use $N = M = 30$.

IV. Experimental Evaluation

We implemented our approach on top of the Open3D library [40] and use Intel's Embree library [36] for effi-

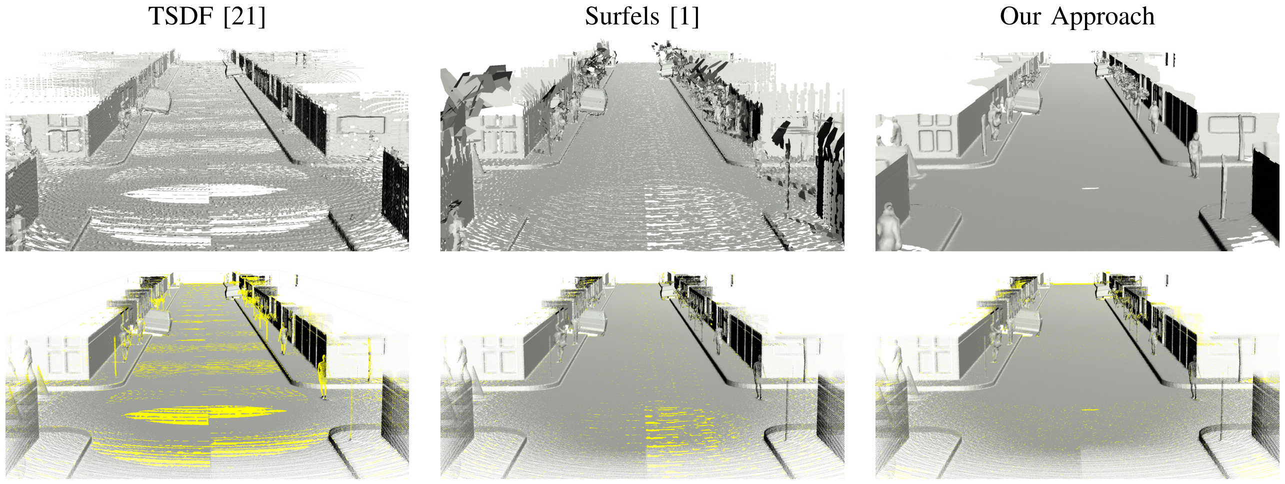

Fig. 3: Qualitative examples showing the map accuracy. The first row exhibits the three map representations, TSDF, surfels and our approach, respectively. The second row depicts the dense GT point clouds used to compute the metrics in Tab. I. Along with the GT clouds, we highlight in yellow the points in the GT model whose distances to the closest point in the built models shown in the first row are greater than $\delta_d = 3\,\text{cm}$. Intuitively, the greater number of yellow points, the more mistakes or gaps a model contains.

cient ray-to-triangle intersection queries. Our algorithm runs entirely on CPU and was tested on an Intel Xeon W-2145 with 8 cores @3.70 GHz with 32 GB RAM. The source code of our approach is available at: `https://github.com/PRBonn/puma`.

For all our experiments we used the default SuMa settings and a voxel size of 0.10 m for TSDF. For our approach, we set the depth of the octree used in the Poisson surface reconstruction [14] to $\Delta_{\text{tree}} = 10$.

A. Datasets

We use simulated and real-world data for our evaluation. For assessing the accuracy of our map, we need ground truth data and thus we generate synthetic sensor data for an urban environment[2]. To obtain the ground truth (GT) model of the scene, we sample points from the CAD using a virtual LiDAR sensor model with the same field of view and properties as the one used to obtain the scans, however using 320 beams instead of 64. This way we obtain a dense ground truth point cloud but only from the parts of the scene that can be observed with the sensor which is key to a fair comparison of the built 3D model. The final GT point cloud contains 62.5 million points and is shown in the second row of Fig. 3.

For real-world experiments, we use the odometry benchmark from the KITTI Dataset [11].

B. Mapping Accuracy

The first set of experiments is designed to show the geometrical accuracy of our map representation. We compare our mesh generation pipeline with two commonly used map representations: surfels [1] and TDSF [21]. To decouple errors in the resulting map caused by the representation itself and the pose estimation accuracy of the SLAM approaches,

[2]The data is available at `http://www.ipb.uni-bonn.de/data/mai-city-dataset`.

Method	Chamfer Distance $d_{\mathcal{CD}}(\mathcal{P},\mathcal{G})$	Precision $P(\delta_d)$	Recall $R(\delta_d)$	F-score $F(\delta_d)$
TSDF [21]	0.66	79.96	85.42	82.6
Surfels [1]	0.11	75.54	98.43	85.48
Ours	**0.05**	**93.28**	**98.69**	**95.91**

TABLE I: Distance evaluation metrics for mapping accuracy. In all experiments $\delta_d = 0.03\,\text{m}$

we use the same poses provided by the ground truth for all systems for a fair comparison.

For a quantitative evaluation of the accuracy of the mapping systems, we densely sample the map representations with a density of 1,000 points per cm^2. Sampling the evaluated map representation allows us to have a fair comparison between each approach and the GT model using standard point-cloud metrics [16], [9].

For our evaluation, we use the following metrics. Let $\mathcal{P}$ be the point cloud sampled from the map and let $\mathcal{G}$ be the GT point cloud. For a point $\mathbf{p} \in \mathcal{P}$, we define the distance to the GT model as:

$$d(\mathbf{p},\mathcal{G}) = \min_{\mathbf{g}\in\mathcal{G}} \|\mathbf{p}-\mathbf{g}\|, \tag{2}$$

where $\|\cdot\|$ is the L2 norm. Analogously, we define the distance for a point $\mathbf{g} \in \mathcal{G}$ to the reconstructed map as $d(\mathbf{g},\mathcal{P}) = \min_{\mathbf{p}\in\mathcal{P}}\|\mathbf{g}-\mathbf{p}\|$. For computing precision, recall, and f-score metrics, we follow exactly the work by Knapitsch et al. [16].

We also define the Chamfer distance [9] as:

$$d_{\mathcal{CD}}(\mathcal{P},\mathcal{G}) = \frac{1}{2|\mathcal{P}|}\sum_{\mathbf{p}\in\mathcal{P}} d(\mathbf{p},\mathcal{G})^2 + \frac{1}{2|\mathcal{G}|}\sum_{\mathbf{g}\in\mathcal{G}} d(\mathbf{g},\mathcal{P})^2. \tag{3}$$

The results for the mapping accuracy are shown in Tab. I. We are especially interested in the areas where the 3D models deviate from the ground truth. Therefore, we highlighted all points that have an error of $\delta_d \geq 3\,\text{cm}$ in *yellow* (second row of Fig. 3).

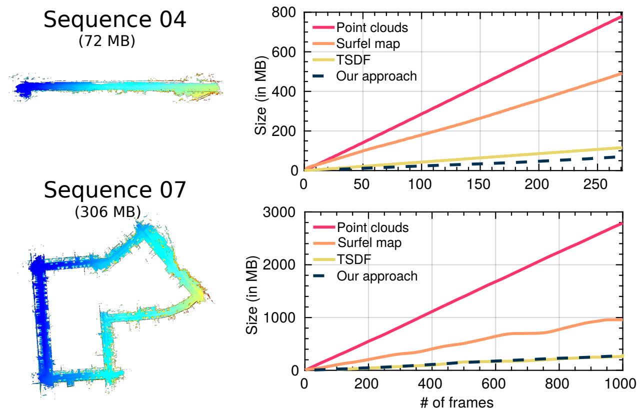

Fig. 4: Memory consumption for different map representations for two KITTI sequences. On the left, we show the triangular mesh maps built by our approach along with their final size. Sequence 04 is a country environment and sequence 07 was recorded in an urban scenario. We see that point clouds and surfels result in higher memory usage. TSDF performs similarly to our approach in terms of memory usage, but as seen in Fig. 3, our approach outperforms TSDFs in terms of geometrical accuracy.

We can see that the TSDF-based approach fails to reconstruct important distinctive objects of the scene such as trees, pedestrians, or poles. Is important to remark that this is a limitation of the method as the framework used [20] does not drop any input scans. The surfel-based approach often provides more accurate reconstructions, however, it also adds spurious artifacts to the reconstructed scene. Our approach outperforms both in terms of accuracy and correctness of the model since it does not miss important features and does not add artifacts at the same time.

C. Memory Efficiency

To use these high definition maps in practice, it is necessary to be able to represent this map efficiently. The second evaluation is designed to investigate the memory requirements of different map representations. We show that our map representation, based on triangular meshes, is a good choice in terms of a trade-off between geometrical accuracy and the corresponding memory footprint. We evaluate the amount of memory needed to represent a given map using our approach, TSDFs [21], surfels [1], and the LiDAR point clouds, only considering the raw geometric model, not any other additional information (such as normals, color information, etc.) that could be stored. The voxel size of $0.10\,\mathrm{m}$ for the TSDF-based [21] approach is chosen such that the spatial resolution of the extracted mesh results similar to the one built with our approach. SuMa [1] does not provide a mechanism to control its map size.

For a mesh-based map representation, the minimum amount of memory needed to represent a triangular mesh is given by $N_v\, v_{\text{size}} + N_f\, f_{\text{size}}$, where N_v and N_f are the numbers of vertices and faces in the model respectively and v_{size} and f_{size} are the size of a vertex and a triangular face respectively.

In the case of a TSDF based map representation, Whelan et al. [37] shows that it is not directly feasible to store the TSDF volume as a map representation for large outdoor environments in general. In practice, we need to extract the triangular mesh from the TSDF volume to store these maps. As a result, while comparing the memory footprint of the TSDF approach, we use the size of the extracted mesh instead of the TSDF volume. For a surfel, we need to represent the center position of the surfel, the normal, and a radius. Therefore, the size of a surfel map is given by $N_S\,(c_{\text{size}} + n_{\text{size}} + r_{\text{size}})$, where N_S is the number of surfels, c_{size} is the size of the center position, n_{size} is the size of the normal vector, and r_{size} is the size of the radius. For the case of a point cloud map, the size of the map is given by $N_P\, p_{\text{size}}$, where N_P is the number of points and p_{size} is the size of a point.

Fig. 4 shows the memory consumption of the different approaches over time for two different sequences of the KITTI odometry benchmark. We see that our approach scales well with the number of input scans, while surfel-based maps or point cloud maps require much more memory, which makes it not feasible to run on mobile platforms. TSDF-based maps exhibit similar memory consumption, however with a reduced mapping accuracy, as shown in Sec. IV-B.

D. Odometry and Localization Accuracy

In this experiment, we illustrate that our approach is well-suited for estimating the pose of the vehicle by registering the LiDAR scans using our map representation. We evaluate our performance on the KITTI dataset [11] where we estimate the poses by registering the 3D LiDAR scans incrementally. We compare our approach to commonly used registration algorithms based on ICP, namely point-to-point ICP [3], point-to-plane ICP [32] and generalized-ICP [33]. For all these three approaches, we use the same optimization framework described in Sec. III-A, including also a Huber loss [13] function to reject outliers. As we do with our approach, we also initialize the ICP with the last increment estimate obtained at time $t-1$. Additionally, we consider a different scenario for these three methods, instead of performing frame-to-frame ICP, we perform frame-to-model ICP, where the *model* is the last N aggregated scans. This is represented as *point cloud map* in Tab. II. This way we can evaluate the benefits of running the reconstruction algorithm on this local map or not. For performing data-associations on the three ICP variants, we employ nearest neighbors search. Lastly, we also consider a different data-association scenario, the projective data association. For this, we compare our approach to *frame-to-frame* SuMA [1]. The TSDF [20] framework does not provide pose estimates and therefore is omitted in this experiment. It is important to remark that to compare all methods equally, we use the same normal computation for all approaches as described in Sec. III-A. This normal computation will generally decrease the performance of any normal-based metric for estimating the pose.

In terms of estimation performance, we can see in Tab. II that our approach provides solid pose estimate performance when comparing the poses provided by our system to the ground truth poses provided by KITTI.

Map	Method	DA	avg.	00	01	02	03	04	05	06	07	08	09	10
None	point-to-point ICP [3]	NN	4.86	3.65	15.40	4.60	5.74	2.44	3.38	2.94	4.92	3.75	3.38	3.24
			1.71	1.62	1.06	1.70	2.70	1.37	1.47	1.36	2.43	1.78	1.59	1.75
	point-to-plane ICP [32]	NN	7.60	9.12	10.00	6.19	6.04	4.51	7.69	8.24	5.53	8.70	9.37	8.18
			3.49	3.89	1.53	2.42	3.05	2.65	3.79	4.10	4.17	3.97	3.50	5.29
	GICP [33]	NN	14.35	7.35	73.10	14.40	9.37	13.60	5.23	2.23	6.40	7.27	10.90	8.00
			4.78	3.00	24.10	4.04	2.85	2.15	2.14	1.06	2.87	3.17	3.78	3.38
	SuMa [1]	Proj.	2.93	2.09	4.05	2.30	**1.43**	11.90	1.46	0.95	1.75	2.53	1.92	1.81
			0.92	0.93	1.22	0.79	**0.75**	1.06	0.79	0.64	1.17	0.96	0.78	0.97
Point cloud	point-to-point ICP [3]	NN	29.98	9.25	93.20	30.40	10.90	91.40	**0.92**	33.90	8.35	2.81	33.90	14.80
			2.61	1.58	8.75	5.04	2.21	0.99	**0.46**	6.38	0.56	0.68	6.59	1.65
	point-to-plane ICP [32]	NN	18.92	9.99	77.10	11.70	2.31	70.00	2.62	1.84	1.79	3.67	17.40	9.70
			4.01	4.29	17.60	2.70	1.01	5.21	1.17	0.84	1.16	1.47	5.47	3.15
	GICP [33]	NN	20.43	4.34	93.10	10.70	2.21	83.70	1.56	1.42	1.19	2.33	21.80	2.37
			2.76	0.94	17.20	2.25	1.19	1.81	0.73	0.68	0.78	0.95	2.91	0.90
Mesh	Ours ($\Delta_{\text{tree}} = 10$)	NN	2.15	3.14	4.32	1.91	1.34	2.09	1.56	1.41	1.88	1.97	1.80	2.21
			1.14	1.49	1.04	0.73	1.07	1.46	1.07	0.72	1.36	1.10	0.82	1.67
	Ours ($\Delta_{\text{tree}} = 10$)	**RC**	**1.55**	**1.46**	**3.38**	**1.86**	1.60	**1.63**	1.20	**0.88**	**0.72**	**1.44**	**1.51**	**1.38**
			0.74	**0.68**	**1.00**	**0.72**	1.10	**0.92**	0.61	**0.42**	**0.55**	**0.61**	**0.66**	**0.84**

TABLE II: Odometry estimation results for the KITTI Odometry benchmark [10]. The rows highlighted in gray correspond to the translational error and the row below to the rotational error. All errors are averaged over trajectories of 100 to 800 m length. The translational error is in % and the relative rotational error in degrees per 100 m. DA for data association, RC for ray casting, NN for nearest neighbor. Bold numbers indicate the best approach for the given sequence.

E. Registration Algorithm

We briefly show that the novel registration scheme depicted in Sec. III-A is superior in terms of accuracy and speed. We compare our registration pipeline against a standard point-to-plane ICP [32], where the source in both cases is the incoming scan from the LiDAR sensor and the target is set to be the triangular mesh built for our approach. We compare two different registration algorithms, one that sample all vertices and normals of the mesh to obtain a point cloud, and the second, that uses our novel data association algorithm based on ray-casting (RC) as explained in Sec. III-A. While traditional ICP makes use of nearest neighbor searches, we perform ray-casting through the mesh to obtain the target correspondences and the normal information. The registration is evaluated by running our registration algorithm in the full KITTI training sequences, registering all LiDAR scans to the local mesh-map (Sec. III-C), using different map resolutions, i.e., by setting different resolutions of the Δ_{tree} on the Poisson surface reconstruction [15]. To investigate the accuracy of the registration method we compute the average translational and rotational errors overall training sequences of the dataset. We see that the proposed registration algorithm scales better when the size of the input mesh increases. The results of this experiment are shown in Tab. III.

Δ_{tree}	Vertices	Mesh vertex-sampling			Mesh ray-casting		
		t_{err}	r_{err}	runtime	t_{err}	r_{err}	runtime
8	30K	2.82	1.23	682	**1.73**	**0.90**	**395**
9	100K	2.05	1.08	645	**1.53**	**0.74**	**418**
10	300K	2.14	1.17	789	**1.56**	**0.74**	**535**

TABLE III: Ray-casting vs mesh-sampling registration evaluation in the full KITTI [11] training sequences. Relative errors are averaged over trajectories of 100 to 800 m length. Relative translational error (t_{err}) in % and relative rotational error (r_{err}) in degrees per 100 m. The runtime values are expressed in *milliseconds*.

F. Runtime

The pre-processing and the normal estimation take 45 ms on average per scan, and the scan-matching algorithm takes another extra 500 ms. However, the bottleneck is the meshing algorithm which takes on average 5 s when executed on a CPU. This makes our approach infeasible for online operation on autonomous vehicles. In this work, we have primarily focused on the use of triangular meshes in developing SLAM pipelines and show that both the reconstruction quality and the pose estimation accuracy are promising. In future work, we need to investigate techniques for optimizing the meshing algorithm to enable online performance, for instance, running the reconstruction algorithm on the GPU, additionally, GPU-enabled ray tracing engines, like the NVIDIA OptiX™, can be used to accelerate the ray-casting registration algorithm.

V. Conclusion

In this publication, we presented a novel approach for simultaneous odometry and mapping with 3D LiDARs. Our approach performs a novel frame-to-mesh registration but in contrast to other SLAM or odometry and mapping approaches, we represent the map as a triangle mesh estimated using Poisson surface reconstruction in a sliding window over a sequence of past scans. We show that we obtain high-quality local meshes that show more details than common alternative methods such as state-of-the-art TSDF or surfel representations. We also show that our map representation is well-suited for incremental scan registration for pose estimation. For future work, our system can be extended by incorporating loop closures [38] and a pose graph optimization framework for efficient dense map correction.

References

[1] 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.

[2] M. Berger, A.Tagliasacchi, L. Seversky, P. Alliez, G. Guennebaud, J. Levine, A. Sharf, and C. Silva. A survey of surface reconstruction from point clouds. In *Computer Graphics Forum*, volume 36, pages 301–329, 2017.

[3] P.J. Besl and N.D. McKay. A Method for Registration of 3D Shapes. *IEEE Trans. on Pattern Analalysis and Machine Intelligence (TPAMI)*, 14(2):239–256, 1992.

[4] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J.J. Leonard. Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age. *IEEE Trans. on Robotics (TRO)*, 32:1309–1332, 2016.

[5] J.C. Carr, R.K. Beatson, J.B. Cherrie, T.J. Mitchell, W.R. Fright, B.C. McCallum, and T.R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *Proc. of the Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 67–76, 2001.

[6] X. Chen, I. Vizzo, T. Läbe, J. Behley, and C. Stachniss. Range Image-based LiDAR Localization for Autonomous Vehicles. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2021.

[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] J. Deschaud. Imls-slam: scan-to-model matching based on 3d data. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[9] H. Fan, H. Su, and L.J. Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 605–613, 2017.

[10] M. Gehrig, E. Stumm, T. Hinzmann, and R. Siegwart. Visual Place Recognition with Probabilistic Voting. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2017.

[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] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Proc. of the Intl. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 71–78, 1992.

[13] P. J. Huber. *Robust Statistics*. Wiley, 1981.

[14] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006.

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

[16] A. Knapitsch, J. Park, Q. Zhou, and V. Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics*, 36(4):1–13, 2017.

[17] R. Kolluri. Provably good moving least squares. *ACM Transactions on Algorithms (TALG)*, 4(2):18, 2008.

[18] T. Kühner and J. Kümmerle. Large-scale volumetric scene reconstruction using lidar. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2020.

[19] Z.C. Marton, R.B. Rusu, and M. Beetz. On fast surface reconstruction methods for large and noisy point clouds. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 3218–3223. IEEE, 2009.

[20] A. Milan, T. Pham, V.K.B. G, D. Morrison, A.M.W. Tow, L. Liu, J. Erskine, R. Grinover, A. Gurman, T. Hunn, N. Kelly-Boxall, D.Q. Lee, M. McTaggart, G.M. Rallos, A. Razjigaev, T.J. Rowntree, T. Shen, R. Smith, S. Wade-McCue, Z. Zhuang, C. Lehnert, G. Lin, I. Reid, P. Corke, and J. Leitner. Semantic segmentation from limited training data. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[21] A. Millane, Z. Taylor, H. Oleynikova, J. Nieto, R. Siegwart, and C. Cadena. C-Blox: A Scalable and Consistent TSDF-based Dense Mapping Approach. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[22] F. Moosmann and C. Stiller. Velodyne SLAM. In *Proc. of the IEEE Vehicles Symposium (IV)*, pages 393–398, 2011.

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

[24] 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.

[25] 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.

[26] 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.

[27] E. Piazza, A. Romanoni, and M. Matteucci. Real-time cpu-based large-scale 3d mesh reconstruction. *arXiv preprint arXiv:1801.05230*, 2018.

[28] V. Reijgwart, A. Millane, H. Oleynikova, R. Siegwart, C. Cadena, and J. Nieto. Voxgraph: Globally consistent, volumetric mapping using signed distance function submaps. *IEEE Robotics and Automation Letters (RA-L)*, 5(1):227–234, 2019.

[29] L. Roldão, R. Charette, and A. Verroust-Blondet. 3d surface reconstruction from voxel-based lidar data. *arXiv preprint*, (1906.10515v1), 2019.

[30] A. Rosinol, T. Sattler, M. Pollefeys, and L. Carlone. Incremental visual-inertial 3d mesh generation with structural regularities. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 8220–8226. IEEE, 2019.

[31] F. Ruetz, E. Hernández, M. Pfeiffer, H. Oleynikova, M. Cox, T. Lowe, and P. Borges. Ovpc mesh: 3d free-space representation for local ground vehicle navigation. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 8648–8654. IEEE, 2019.

[32] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Proc. of Int. Conf. on 3-D Digital Imaging and Modeling*, pages 145–152, 2001.

[33] A. Segal, D. Haehnel, and S. Thrun. Generalized-ICP. In *Proc. of Robotics: Science and Systems (RSS)*, 2009.

[34] C. Stachniss, J. Leonard, and S. Thrun. *Springer Handbook of Robotics, 2nd edition*, chapter Chapt. 46: Simultaneous Localization and Mapping. Springer Verlag, 2016.

[35] 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.

[36] I. Wald, S. Woop, C. Benthin, G.S. Johnson, and M. Ernst. Embree: a kernel framework for efficient cpu ray tracing. *ACM Transactions on Graphics*, 33(4):1–8, 2014.

[37] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald. Kintinuous: Spatially Extended KinectFusion. In *Proc. RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2012.

[38] T. Whelan, M Kaess, J. Leonard, and J. McDonald. Deformation-based loop closure for large scale dense rgb-d slam. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2013.

[39] J. Zhang and S. Singh. LOAM: Lidar Odometry and Mapping in Real-time. In *Proc. of Robotics: Science and Systems (RSS)*, 2014.

[40] Q.Y. Zhou, J. Park, and V. Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.