NOctoSLAM: Fast Octree Surface Normal Mapping and Registration

Joscha Fossel[1] Karl Tuyls[1] Benjamin Schnieders[1] Daniel Claes[1] Daniel Hennes[2]

***Abstract*— In this paper, we introduce a SLAM front end called *NOctoSLAM*. The approach adopts an octree-based map representation that implicitly enables source and reference data association for point to plane ICP registration. Additionally, the data structure is used to group map points to approximate surface normals. The multi-resolution capability of octrees, achieved by aggregating information in parent nodes, enables us to compensate for spatially unbalanced sensor data typically provided by multi-line lidar sensors. The octree-based data association is only approximate, but our empirical evaluation shows that NOctoSLAM achieves the same pose estimation accuracy as a comparable, point cloud based approach. However, NOctoSLAM can perform twice as many registration iterations per time unit. In contrast to point cloud based surface normal maps, where the map update duration depends on the current map size, we achieve a constant map update duration including surface normal recalculation. Therefore, NOctoSLAM does not require elaborate and environment dependent data filters. The results of our experiments show a mean positional error of 0.029 m and 0.019 rad, with a low standard deviation of 0.005 m and 0.006 rad, outperforming the state-of-the-art by remaining *accurate* while running *online*.**

I. Introduction

Simultaneous localization and mapping (SLAM) refers to a fundamental problem in robotics in which a robot has to generate a map of an unknown environment without external tracking, while simultaneously localizing itself relative to the same recorded map (Figure 1). Many real-world scenarios, such as for instance search and rescue or planetary exploration, lack external tracking infrastructure and consequently require simultaneous localization and map generation.

Mobile robots are commonly equipped with 2D laser sensors motivated by high precision and affordable price. Recently, 3D laser sensors have been getting smaller and more affordable, a trend which might amplify with the introduction of solid state lasers in the near future. Currently, most 3D laser sensors provide multiple 2D scans at different inclinations. Unlike 2D lasers, 3D lasers allow for 6D pose estimation and 3D mapping without requiring additional sensors, e.g. inertial measurement units or altitude sensors. However, handling the large amount of data typically provided by such sensors can prove challenging. In order to register 3D scans online and in real-time, efficient scan matching algorithms are required.

The main contribution of our paper is the introduction of NOctoSLAM, a SLAM front-end which uses an octree-based map representation that supports surface normals. This enables quick nearest neighbour search for source to reference point association and fast surface normal approximation. Scan to map registration is performed using the Iterative Closest Point [1] algorithm with a point to plane error metric. Due to its computational performance, NOctoSLAM does not require explicit data filtering, and thus also voids the need for manual fine-tuning of filter parameters.

[1]Department of Computer Science, University of Liverpool, L69 3BX Liverpool, United Kingdom jfossel,bsc,dclaes,ktuyls@liv.ac.uk

[2]Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Robotics Innovation Center, 28359 Bremen, Germany daniel.hennes@dfki.de

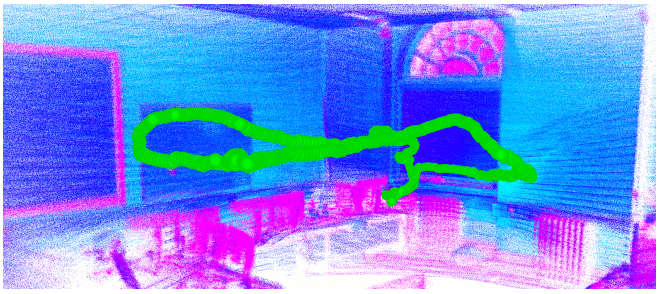

Fig. 1. This figure shows a NOctoSLAM generated 3D trajectory. The pose estimates are used to transform the sensor data, where color represents scan intensity. Recognizable details such as the whiteboard, radiators and the beams in the arched window indicate a good pose estimation accuracy.

Our evaluation illustrates that NOctoSLAM is outperforming the state-of-the-art (i.e. *libpointmatcher* [2]) computationally, while matching its accuracy. The computational performance of SLAM front ends is important as it has an impact on the latency between acquiring sensor data and updating the pose estimate. Furthermore, better computational performance can increase the update rate, and also determines how much of the sensor data can be processed instead of filtered. Adjusting filter parameters of libpointmatcher in order to enable near real-time processing, unavoidably reduces accuracy. Thus, in an attempt to achieve the same runtime as NOctoSLAM, applying source and reference point filters to speed up libpointmatcher resulted in a positional median error of ~0.09 m, compared to ~0.03 m for NOctoSLAM.

The remainder of this paper is organized as follows. Section II presents related work and provides a concise background. Section III introduces NOctoSLAM and specifies the functionality in detail. Section IV describes the experiments performed in order to compare NOctoSLAM to other approaches. Results are discussed and a conclusion is presented in Section V.

II. Related Work & Background

SLAM algorithms can differ in the way they register points and represent maps. Point registration is typically performed by an iterative closest point (ICP) algorithm.

The libpointmatcher [2] library supports different ICP variations intended for direct comparison of different registration and error measurement approaches. For fast nearest neighbour search, it makes use of libnabo [3], a kd-tree implementation. It furthermore features various finely tunable pre- and post-processing point filtering techniques, reducing the amount of data processed in order to lower computation costs.

An alternative way to address the limited computational capabilities of mobile robots is presented with LOAM [4]. It effectively separates the self localization from the mapping operation by performing them at different rates with different levels of accuracy. Pose estimation is performed at high-frequency, but with low fidelity, while mapping is done less often but investing a higher amount of computational resources in order to find a more precise result. In order to achieve good pose estimates despite low fidelity, LOAM uses intricate feature detection algorithms to preserve only meaningful data points.

An intuitive way to map laser sensor data is converting the measured distances into spatial coordinates, then storing the coordinates in a point cloud, an unordered set of all recorded points. In order to reduce the computational costs of finding nearest neighbours, the number of stored points can be reduced using filters, or by using a tree-based data structure, such as a kd-tree or an octree. An octree representation, recursively partitioning a cubic space into eight smaller cubes, is a memory and time efficient way to represent 3D environments in an organized manner. Figure 2 shows an example of storing laser sensor data in an octree. OctoMap [5] is an octree-based mapping implementation, representing occupied, free and unmapped areas distinctly in a memory efficient way.

In structured environments, such as buildings, one can make reasonable assumptions of the terrain layout in between neighbouring measurements. Point to plane ICP [6] presents an extension to the traditional iterative closest point algorithm [1], using an error metric from each point to the tangent plane of the corresponding closest points, thus relaxing the requirement for exact point matches. This error metric has been shown to converge faster in general [7], especially for few or distant points. As most 3D laser scanners produce sparse data in the vertical axis, point to plane ICP is a reasonable choice. Another way of dealing with such non-uniform density point clouds is presented in [8], where the authors propose approximate surface reconstruction to enhance registration performance.

A detailed overview of point cloud registration algorithms is given in [9].

III. NOctoSLAM

We propose an octree-based map representation that enables fast point to plane scan registration [6]. Iterative Closest Point (ICP) in the point to plane variant requires both source/input to reference/map point association and a surface normal estimate for the reference points. The proposed map structure inherently provides surface normal approximations

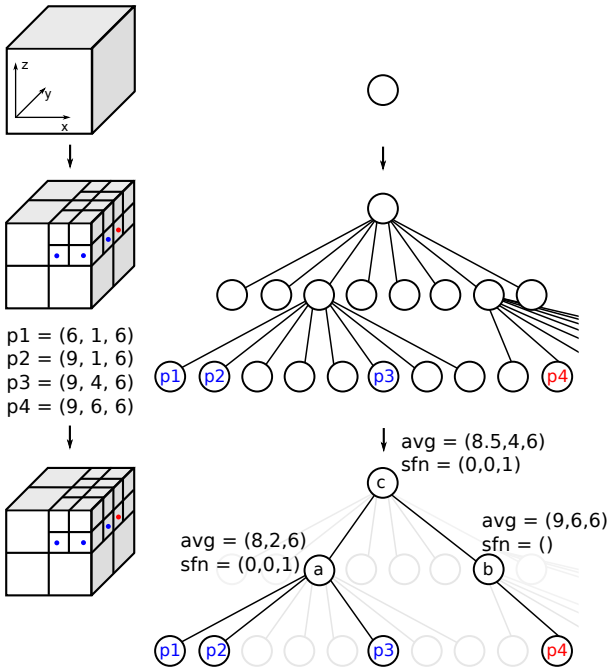

Fig. 2. This figure shows a simplified map update. First, the initially empty octree is extended to accommodate new data points. After inserting the data points into the tree, the changes are propagated through the tree from bottom to top, and surface normals (sfn) are calculated if feasible.

and allows for direct association of source point cloud to reference map. Thus, we avoid having to maintain additional data structures that perform nearest neighbour search for said tasks. In comparison, point to plane ICP [6] and generalized ICP [10] based algorithms, e.g. [2], [11], use kd-trees for data association and surface normal estimation, whilst storing the map in an unorganized point cloud format. While there exist highly optimized approaches to tackle the k-nn problem, such as libnabo [3], kd-tree based nearest neighbour search in large point cloud based maps is not feasible online (see Section IV). To achieve registration and map updates that are faster than the sensor update rate, typically requires reducing the number of points by filtering out large amounts of the provided scan endpoints. The nearest neighbour approximation method proposed in this paper is very fast and allows us to map and register 300,000 points per second, whilst being sufficiently accurate to generate valid surface normals. Hence, NOctoSLAM does not require explicit filtering of sensor data as other point to plane ICP based methods do. Tuning such filters can be challenging, environment dependent, and also costly in terms of runtime.

In the following subsection we will explain how (i) updating the map, i.e. scan end point insertion and surface normal approximation, and (ii) registration, i.e. data association and pose updates, are performed in NOctoSLAM.

A. Mapping

To represent the environment, we extend the octomap approach introduced in [5]. In the octomap approach, every octree node represents a voxel, where its resolution depends

on the node's level in the tree. Commonly, every octree node stores the probability of representing occupied space, while the position it represents in 3D space depends on the node's position in the tree. In NOctoSLAM we additionally store two 3D vectors per node, i.e., a surface normal and a position that can deviate from the center of the voxel. The former is necessary for point to plane scan registration. The latter allows for a more precise map representation at coarse map resolutions. NOctoSLAM uses both data stored in leaf and non-leaf nodes, thus such a position anchor is required.

Updating the map consists of two steps, insertion and propagation, depicted in Figure 2. First, the tree is extended if necessary, and the scan endpoint positions are inserted into the corresponding nodes (*p1*, *p2*, *p3*, *p4* in Figure 2). If a leaf node n_i already exists, the stored position is updated with the scan endpoint position by using the weighted average according to sensor model M and node occupancy probability n_i^p:

$$\mathbf{n}_i = \frac{\mathbf{n}_i * n_i^p + M * \mathbf{d}_k}{M + n_i^p}, \tag{1}$$

where vector $\mathbf{n}_i$ is the position previously stored in node n_i and vector $\mathbf{d}_k$ is the endpoint position of scan d_k.

After having inserted all scan endpoints in this fashion, the updates are propagated through the tree: all non-leaf nodes traversed during the first step are updated from bottom to top, based on the information stored in their descendants. In particular, parent nodes store the average position of their descendants, and use these positions to approximate a surface normal if feasible (node *a* in Figure 2).

We approximate the surface normal by estimating the normal of a plane tangent to the surface, which can be formulated as a least-square problem. As shown in [12] the solution can be reduced to a principal component analysis of the covariance matrix $\mathscr{C}$ generated from the direct descendants $n_{i-1,k}$ of a node $n_{i,j}$:

$$\mathscr{C}(n_{i,j}) = \frac{1}{K}\sum_{k=1}^{K}(\mathbf{n}_{i-1,k} - \mathbf{p}) \cdot (\mathbf{n}_{i-1,k} - \mathbf{p})^T, \tag{2}$$

$$\mathbf{p} = \frac{1}{K}\sum_{k=1}^{K}\mathbf{n}_{i-1,k}, \tag{3}$$

$$\mathscr{C} \cdot \mathsf{v}_m = \lambda_m \cdot \mathsf{v}_m, \; m \in \{0,1,2\}, \tag{4}$$

where K is the number of descendants of node $n_{i,j}$, λ_m the m-th eigenvalue of the covariance matrix, and v_m the m-th eigenvector, which can be computed analytically. If exactly two of three eigenvalues are similar, the corresponding eigenvectors determine the plane through nodes $n_{i-1,k}$, and hence the surface normal for node n_i,k. If no good surface normal can be estimated, the surface normal vector centroid of the descendants is used instead, if available (node *c* in Figure 2). If neither is available, as for node *b*, no surface normal is set. Figure 2 also shows the downside of this approach compared to using traditional nearest neighbour search: the right branch containing *b* and *p4* has no surface normal, and even though *p3* is close to *p4*. This relation is ignored because they are in different branches of the octree.

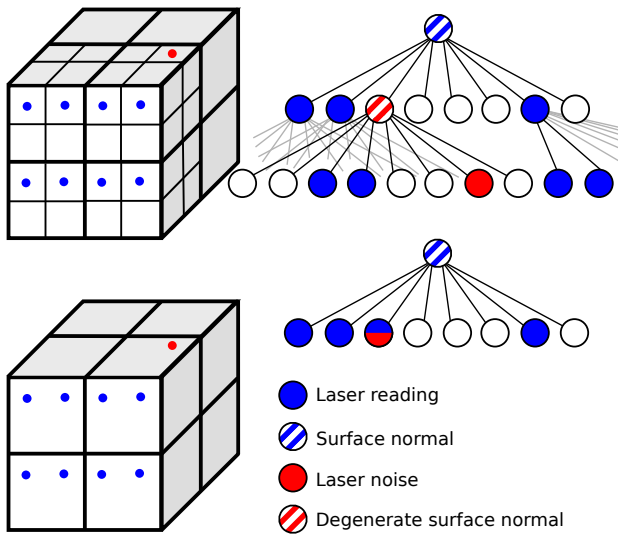

Fig. 3. This figure gives an example for how noisy sensor data can lead to bad surface normal approximations in NOctoSLAM, as shown in the top half of the figure. Here, the noisy data point occupies its own node and the noise propagates to the parent node. The issue is tackled by using dynamic map resolutions. The coarser the map, the more points are being averaged reducing the impact of sensor noise, as shown on the bottom half of the figure, where multiple points are averaged in the leaf nodes.

Nevertheless, experiments (Section IV) show that in practice being able to process more points compensates for not being able to approximate surface normals for some points. Note that the data for such points is not discarded, and it is likely that approximating surface normals in such cases will become feasible at a future iteration due to high map update rates.

Unlike RGBD cameras, multi-line lidars provide only sparse data along the vertical axis. This can lead to degenerate surface normal approximations, as illustrated in Figure 3. In the upper part we can see how noisy input (red dot) would lead to a degenerate surface normal estimate. The red striped node would have a surface normal perpendicular to the actual one. To tackle this, we use dynamic map resolutions by inserting leaf nodes at variable tree depths, as shown in the lower part of Figure 3. By decreasing the resolution we implicitly increase the number of input points that are averaged to estimate a surface normal. Thus, the surface normal ends up being only slightly skewed, instead of being completely off. In particular, we choose a leaf node resolution that forces nodes on vertical scan lines to be adjacent, depending on distance to sensor origin, vertical resolution of the sensor and map occupancy.

B. Pose Updates

To update the pose estimate we employ a point to plane iterative closest point algorithm, originally introduced in [6]. The basic ICP algorithm consists of two steps. First, correspondence between the source and reference data is computed (i.e. scan endpoints are associated with points in the map). Second, a transformation that minimizes the distance between corresponding points from input and reference set

is computed.

To associate a scan endpoint d_i with a position and surface normal stored in the map, the octree is traversed as far as possible from root node towards the leaf node corresponding to the coordinates of d_i. If a leaf node n_i, with a parent node that has its surface normal set is reached, d_i is associated with the position stored in n_i and the surface normal ($\mathbf{s}_i$) stored in the parent node. Otherwise, n_i, the last node traversed for which the surface normal is set, is associated with d_i. Instead of calculating the actual euclidean distance for maximum correspondence distance rejection, the tree level in which n_i is found is used instead. For each such association found $\mathbf{d}_i$, $\mathbf{n}_i$ and $\mathbf{s}_i$ are used to determine the point to plane error according to the following metric:

$$E = \sum_i \left(\left(\mathbf{P} \cdot \begin{bmatrix} \mathbf{d}_i & 1 \end{bmatrix}^T - \begin{bmatrix} \mathbf{n}_i & 1 \end{bmatrix}^T \right) \cdot \begin{bmatrix} \mathbf{s}_i & 0 \end{bmatrix}^T \right)^2, \quad (5)$$

where $\mathbf{P}$ is the current pose matrix.

$$\mathbf{P} = \begin{bmatrix} 1 & -\gamma & \beta & t_x \\ \gamma & 1 & -\alpha & t_y \\ -\beta & \alpha & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (6)$$

with α, β, γ being roll, pitch, yaw angle and t_x, t_y, t_z being the translations along the respective axes. Minimizing Equation 5 is essentially a least-squares optimization problem. We use the libpointmatcher error minimizer [2] to determine $\mathbf{P}$ for the pose update estimates.

IV. EMPIRICAL EVALUATION

In this section we perform experiments to evaluate the pose estimation quality as well as the runtime performance of NOctoSLAM.

The experiments are conducted on an Intel® Core™ i7-4770 CPU with 16GB of RAM, and the sensor used is a VLP-16[1] multi-line lidar. It can provide about 15000 measurements at 20 Hz, with a maximum range of 100 m. The measurements are distributed among 16 horizontal lines over a vertical FOV of 30 deg and over a horizontal FOV of 360 deg. To estimate the accuracy of pose estimates we compare against ground truth poses provided by an Optitrack[2] motion capture system.

We furthermore compare the NOctoSLAM results against ETH Zurich's ICP Mapping tool[3], which uses libpointmatcher [2] for registration and libnabo [3] for nearest neighbour search. The point to plane ICP algorithm, excluding data association, is essentially the same in NOctoSLAM and the ETHZ ICP Mapping tool.

The NOctoSLAM (referred to as *NOcto* in the figures) performance is evaluated against the ETH Mapping tool with two different sets of parameters. For NOctoSLAM, a minimum map resolution of 0.01 m is used, and the map is updated after every registration. In the following, algorithm *ETH* refers to using no input filters and a $0.1 \times 0.1 \times 0.1$ m^3 voxel grid map filter, i.e. the same map resolution as NOctoSLAM. Thus, registration is performed with approximately 15000 input points at 20 Hz. In order to increase runtime performance, *ETH** uses various input filters that reduce the number of input points to approximately 2500. Among others, a max density filter configured to 300 points per m^3 is used. Additionally, the map is also limited to a point density of 50 points per m^3. The algorithm denoted with *ETH** updates the map not after every registration, but only if the map overlaps less than 95 %.

[1] http://velodynelidar.com
[2] http://optitrack.com
[3] http://wiki.ros.org/ethzasl_icp_mapping

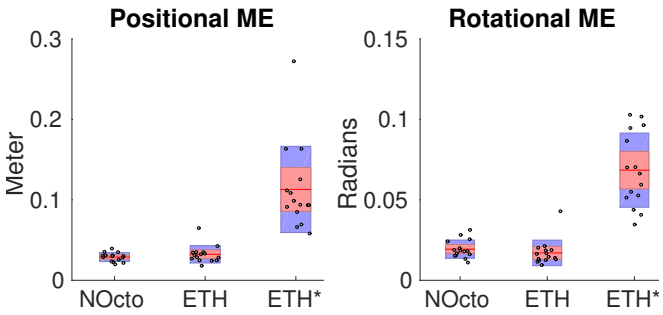

Fig. 4. On the left the mean error (ME) of the position, and on the right the ME of the rotation, are shown. *ETH** performs significantly worse than the other two algorithms, which perform similarly well.

TABLE I
POSE METRICS FOR 15 INDOOR EPISODES

	Position in m			Rotation in radians		
	Median	Mean	σ	Median	Mean	σ
NOcto	**0.030**	**0.029**	**0.005**	0.018	0.019	**0.006**
ETH	0.032	0.032	0.011	**0.015**	**0.017**	0.008
ETH*	0.094	0.113	0.054	0.066	0.068	0.023

A. Pose Estimation Accuracy

We evaluate the accuracy of pose estimates for 15 recorded episodes, where each episode is between 1 and 2 minutes long. In each episode, the sensor is moved manually through a $8 \times 8 \times 3.5$ m^3 room. The sensor is equipped with motion capture markers, which are externally tracked. A rigid transformation resulting from aligning the ground truth and estimated trajectories via ICP is used to calibrate the sensor's optical axis to the markers. For each episode we calculate the mean error (ME) of the position and rotation compared to the ground truth data.

Figure 4 shows the results for 15 episodes, where the center line marks the mean, the inner box represents the 95 % confidence interval, and the outer box illustrates the standard deviation. The figure shows that there is no significant difference of either positional or rotational accuracy between *NOcto* and *ETH*. Strongly reducing the number of source and reference points leads to a significantly worse performance of *ETH**. In fact, with a position mean error of 0.113 m (see Table I), *ETH** approximately quadruples that of *NOcto*. Additionally, the standard deviation of *ETH** is also one order of magnitude higher as the one of *NOcto*. A similar trend can be observed for the three rotation axes.

To visualize the impact of a 0.094 m positional error

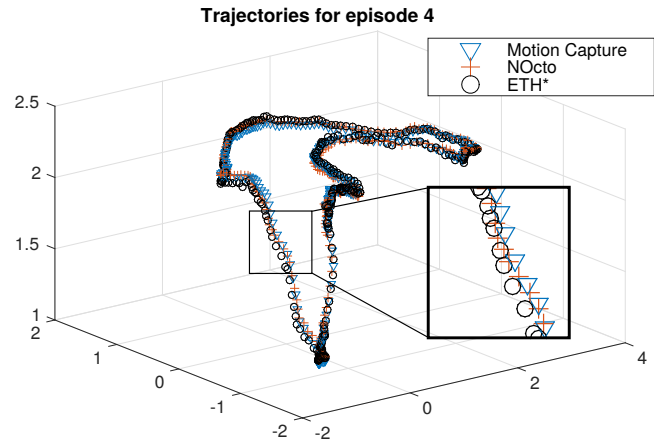

Fig. 5. The pose estimates for the median performance episode of *ETH** are shown. The performance difference between *NOcto* and *ETH** is indicated by the superior alignment of *NOcto* to the motion capture trajectory.

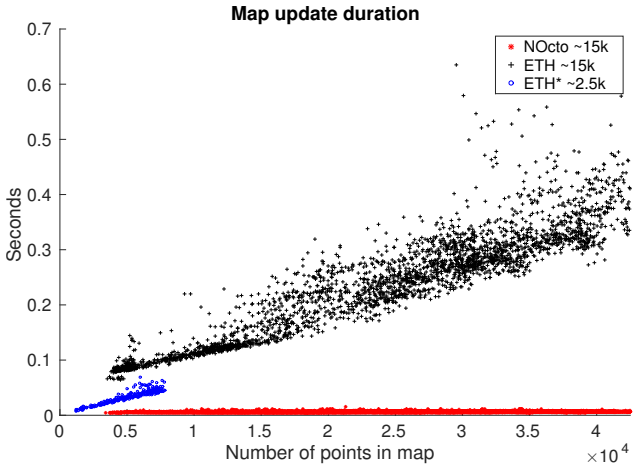

Fig. 6. This figure shows the map update durations in relation to number of points in the map. The points describing the lower constant line belong to *NOcto*, and the points describing the shorter linear line below 0.08 s duration belong to *ETH**. A linear dependency between map size and required time for both *ETH* and *ETH** is visible, while *NOcto* performs in constant time.

versus a 0.029 m error, we plot the trajectories for the median performance episode of *ETH** in Figure 5. In order to increase clarity only every fourth trajectory point is plotted, and we omit the *ETH* trajectory as it is very similar to the *NOcto* trajectory. It can be seen that the *ETH** trajectory aligns significantly worse with the motion capture trajectory than the *NOcto* trajectory does.

B. Runtime Performance

When performing localization and mapping, the main factors in terms of time consumption are updating the surface normals and associating source to reference data. In the following analysis the same data as for the experiments in the previous subsection are used.

We investigate the map update duration in Figure 6. The figure shows that *ETH* and *ETH** map update durations are proportional to the number of points in the map. This stems from the unorganized point cloud representation used that requires reprocessing at least large parts of the map during

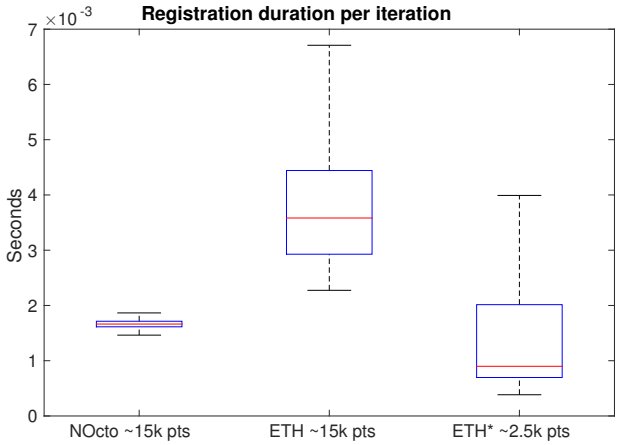

Fig. 7. The registration durations per iteration for the three algorithms are shown. *NOcto* significantly outperforms *ETH* when both process 15000 points per iteration. With the number of points reduced to 2500, *ETH** is faster than *NOcto*, but suffers from a significantly larger standard deviation.

each update. *ETH** shows that the update duration can be reduced by only storing sparse point clouds and using smaller update sizes. Additionally, updating the map in parallel to scan registration, and at a low frequency, makes using a point cloud format computationally feasible. On the other hand, the tree based map representation used in *NOcto* allows for constant time map updates, regardless of map size. Thus, it is not necessary to filter map data for performance reasons with this approach. The mean map update duration for *NOcto* is 0.0062 s with a standard deviation of $6 * 10^{-4}$ s, i.e. ~1/8th of the available time at 20 Hz update rate.

Figure 7 shows a box plot of the scan registration duration per iteration for the three algorithms. The figure was generated from $\sim 4 * 10^5$ iterations. The median of *NOcto* is 0.0018 s, halving that of *ETH* (0.0036 s), while processing approximately the same number of points. With a median of 0.0009 s, *ETH** outperforms the other two, albeit processing only ~2500 points instead of ~15000 points per iteration. However, reducing the number of input points also causes additional computational cost, where the severity depends on the filters used. In our experiments, input filtering for *ETH** took on average 0.0167 s per registration.

C. Visual Inspection

Figure 8 and Figure 9 show maps generated by transforming the sensor data with pose estimates from NOctoSLAM, where the color represents intensity. Figure 8 demonstrates that NOctoSLAM works indoors as well as outdoors. The capability of mapping multiple levels is shown in Figure 9. More material for visual inspection can be found online[4].

D. Summary

In the previous subsections we have shown that NOctoSLAM performs as well as the ETH ICP Mapping Tool in terms of pose estimation accuracy. Since the VLP-16 has a typical accuracy of ± 0.03 m, both algorithms perform well

[4] github.com/smartlab-liv/noctoslam

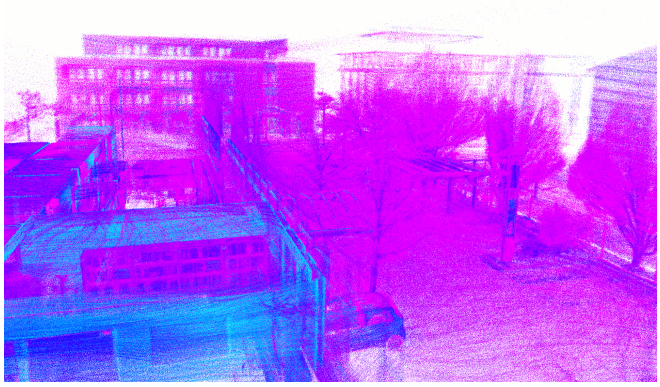

Fig. 8. Intensity sensor data transformed with NOctoSLAM pose estimates from a combined indoor and outdoor episode, showing the DFKI Bremen building.

Fig. 9. Intensity sensor data transformed with NOctoSLAM pose estimates mapping the four story staircase of the UoL Ashton building.

in this regard. However, map updates as well as scan registration are significantly faster in NOctoSLAM. Especially relevant is the constant map update time of NOctoSLAM independent of map size. In contrast, with point cloud based approaches map updates quickly become infeasible in near real-time for large maps and high update rates. While the use of source and reference point filters can increase the performance, it is a cumbersome task to find the correct filter parameters. Also, some parameters depend on the environment, thus requiring prior expert knowledge that may be expensive to get or may not be available. Furthermore, filters might become infeasible if the environment changes, e.g. when changing from indoor to outdoor. For instance, indoor maps typically require more points per m^3 for good pose estimates than outdoor maps. While we do not have quantitative data on the pose estimation accuracy on larger maps, Figures 8 and 9 indicate a good performance, as features such as the bookshelf, car and windows are discernible.

V. Conclusions & Future Work

We presented NOctoSLAM, a novel approach to surface normal based mapping in conjunction with point to plane ICP scan registration. We empirically demonstrated that the proposed algorithm is as accurate, but significantly faster than the state-of-the-art. In terms of precision we achieve a mean error well within the rated accuracy (± 0.03 m) of the sensor used: 0.029 m positional, and 0.019 rad rotational, with a low standard deviation of 0.005 m and 0.006 rad, respectively. Unlike point cloud based approaches, NOctoSLAM can maintain high map update rates, regardless of the map size. Because of the superior computational performance configuring data filters is not necessary.

An octree-based map representation limits the maximum volume the map can cover, depending on the minimum resolution and maximum tree depth. At a minimum resolution of 0.1 m and a tree depth of 16, we can only map a maximum volume of approximately $3000 \times 3000 \times 3000$ m^3. Therefore, as increasing the tree depth leads to higher computation time, in future work we plan to implement a map stitching approach to overcome this limitation. This will allow to evaluate NOctoSLAM with the large scale KITTI [13] benchmark data set.

References

[1] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Sensor Fusion IV: Control Paradigms and Data Structures*, vol. 1611. International Society for Optics and Photonics, 1992, pp. 586–606.

[2] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat, "Comparing icp variants on real-world data sets," *Autonomous Robots*, vol. 34, no. 3, pp. 133–148, 2013.

[3] J. Elseberg, S. Magnenat, R. Siegwart, and A. Nüchter, "Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration," *Journal of Software Engineering for Robotics (JOSER)*, vol. 3, no. 1, pp. 2–12, 2012.

[4] J. Zhang and S. Singh, "LOAM: Lidar Odometry and Mapping in Real-time," in *Robotics: Science and Systems Conference (RSS)*, 2014, pp. 109–111.

[5] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: an efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[6] Y. Chen and G. Medioni, "Object modelling by registration of multiple range images," *Image Vision Comput.*, vol. 10, no. 3, pp. 145–155, Apr. 1992.

[7] S. Rusinkiewicz and M. Levoy, "Efficient variants of the icp algorithm," in *Third International Conference on 3-D Digital Imaging and Modeling, 2001. Proceedings*. IEEE, 2001, pp. 145–152.

[8] D. Holz and S. Behnke, *Mapping with Micro Aerial Vehicles by Registration of Sparse 3D Laser Scans*. Cham: Springer International Publishing, 2016, pp. 1583–1599.

[9] F. Pomerleau, F. Colas, and R. Siegwart, "A review of point cloud registration algorithms for mobile robotics," *Foundations and Trends in Robotics (FnTROB)*, vol. 4, no. 1, pp. 1–104, 2015.

[10] A. Segal, D. Haehnel, and S. Thrun, "Generalized-icp," in *Proceedings of Robotics: Science and Systems*, Seattle, USA, June 2009.

[11] J. Servos and S. L. Waslander, "Multi channel generalized-icp," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 3644–3649.

[12] R. B. Rusu, "Semantic 3d object maps for everyday manipulation in human living environments," Ph.D. dissertation, Computer Science department, TU Muenchen, Germany, October 2009.

[13] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research*, 2013.