The peopleremover – removing dynamic objects from 3D point cloud data by traversing a voxel occupancy grid

Johannes Schauer[1], and Andreas Nüchter[1]

Abstract—**Even though it would be desirable for most post-processing purposes to obtain a point cloud without moving objects in it, it is often impractical or downright impossible to free a scene from all non-static clutter. Outdoor environments contain pedestrians, bicycles, and motor vehicles which cannot easily be stopped from entering the sensor range and indoor environments like factory production lines cannot be evacuated due to production losses during the time of the scan. In this paper we present a solution to this problem that we call the "peopleremover". Given a registered set of 3D point clouds we build a regular voxel occupancy grid and then traverse it along the lines of sight between the sensor and the measured points to find differences in volumetric occupancy between the scans. Our approach works for scan slices from mobile mapping as well as for the more general scenario of terrestrial scan data. The result is a clean point cloud free of dynamic objects.**

Index Terms—**Range Sensing, Mapping**

I. INTRODUCTION

RECORDING 3D point cloud data is often carried out in environments that are not fully static and where moving objects like humans, cars or bicycles will inadvertently enter the field of view of the sensor. Possible scenarios are: (1) indoor office scans for intrusion detection and workspace planning, (2) 3D maps of factories to enable industry 4.0 applications, (3) scans of mining sites to monitor progress and watch for hazards, (4) scans of an urban environment for city planning and documentation purposes, (5) maps of historical sites for archaeology and preservation purposes, (6) and scans for gaming and virtual reality applications.

Nevertheless, these applications are usually interested in a 3D point cloud that only consists of the static environment. But obtaining such a "clean" point cloud by clearing public places of all dynamic objects like pedestrians, bicycles, and parked as well as moving cars might be impossible. And due to production losses, stopping work in a factory or at an excavation site is also undesirable. A cheap way to remove dynamic objects from scans would be to take multiple scans at different times from the same vantage point (the position of the scanner) but it would be desirable to avoid this overhead. In this paper we present a solution to this problem. We call our algorithm the "peopleremover" and it is able produce a "cleaned" point cloud without points belonging to dynamic objects in it.

The input to our algorithm is registered 3D range data, typically acquired by a 3D laser range finder. While we only test our approach with LIDAR scans, it is in principle also compatible with scans obtained from RADAR or RGB-D systems or point clouds from stereo vision. Given such a set of registered point clouds, our algorithm is able to partition the points into those belonging to static as well as dynamic objects. Essentially, if a volume is seen as free where another scan measured points, then these points must be non-static and get marked for removal. We approximate occupied volume by a voxel grid and determine free voxels by traversing the lines of sight from the sensor to the measured points through the voxel grid. Our approach is able to detect change where two or more scans overlap in the volumes they explicitly observe as free or occupied. Apparent change created by occlusion is suppressed.

Our algorithm makes very few requirements on the underlying geometry of the scanned data, vantage points and the temporal separation between individual scans. The vantage points together with the geometry of the scene must be chosen such that the volumes of interest are not occluded from the sensor. Instead, the volumes that one wants to remove moving objects from must have been observed at least by two different scans. Furthermore, the temporal difference between these two scans must be large enough such that any object that one considers "dynamic" in the observed volume was moved to a different location. But if a given voxel volume was observed more than twice, then it is sufficient that the voxel was seen as "free" by only a single scan.

Our method performs best in environments with clear surface normals but in their absence, false positives are easily removed by a fast clustering algorithm. To avoid artifacts due to the voxel discretization we also show an algorithm that reliably removes them without reducing the quality of the remaining point cloud. An example of our algorithm is shown in Figure 1 where pedestrians in the foreground and cars in the background are classified as non-static and are removed.

II. RELATED WORK

Our solution falls into the realm of change detection [1] but only few publications deal with classifying points as

Manuscript received: 09, 10, 2017; Revised 12, 07, 2017; Accepted 01, 17, 2018.

This paper was recommended for publication by Cyrill Stachniss upon evaluation of the Associate Editor and Reviewers' comments.

[1]Johannes Schauer and Andreas Nüchter are with Informatics VII: Robotics and Telematics, Julius-Maximilians-University, Am Hubland, Würzburg 97074, Germany {johannes.schauer, andreas.nuechter}@uni-wuerzburg.de

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

either dynamic or static. Even fewer compute the free volume between a measured point and the sensor itself. Most solutions for change detection (like the one by Vieira et al which uses spatial density patterns [2]) compare incoming geometries or point clouds in a way that results in “change” merely due to occlusion or incomplete sensor coverage. But for our purpose of “cleaning” scans, it is undesirable to remove these parts from the dataset. Doing so would mean to remove potentially useful data from the input. Instead, we designed our algorithm to be conservative. It only removes volumes which it is able to confidently determine to be dynamic. Volumes which it cannot make a decision upon, for example because they were only measured by a single scan, are left untouched. Meeting this requirement is only possible by computing unoccupied volumes and detecting change explicitly. The changes we are interested in can only be detected if a given point falls into the volume that another measurement observed as free.

The work most similar to ours is by Underwood et al. [3]. It is able to detect changes between two scans by “ray tracing” points in a spherical coordinate system. But since their algorithm is limited to comparing no more than two scans at a time it is not directly applicable to our use case without either additional heuristics or quadratic runtime with respect to the number of scans. Given N input scans and without additional processing to find scan pairs with a “meaningful” overlap in their observed volume, the only way to find all changed points is to compare all possible pairs of scans. With N scans this results in a worst-case scenario of $\frac{N(N-1)}{2}$ comparisons and thus quadratic runtime. Our approach is of linear complexity relative to the input size because all comparisons are made against a global occupancy grid and not directly against point data from other scans. The algorithm by Underwood et al. requires two parameters: the angle and the range threshold. In contrast to that, our algorithm only requires a single parameter: the voxel size. Thus, obtaining the optimal parameter for a given training dataset is simpler for our method as our parameter-space is only one-dimensional and not two-dimensional. Nevertheless, the authors publicly provide their code and their datasets which we thus use to benchmark our own method against theirs.

Another approach close to ours is the method by Xiao et al. [4], [5]. Similar to our method and the method by Underwood et al. their algorithm also considers the volume by laser rays and fuses multiple rays into a larger volume using the Dempster-Shafer theory for intra-data evidence fusion and inter-data consistency assessment. Similar to our method, they rely on surface normals but unlike ours, the method detects changes at the point-level without voxelization. The authors do not provide any runtimes of their solution.

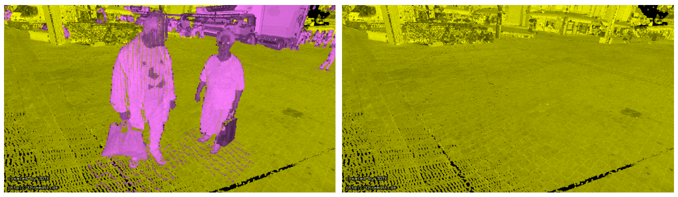
Fig. 1. After identifying non-static points (in magenta on the left) they are removed without artifacts (right).

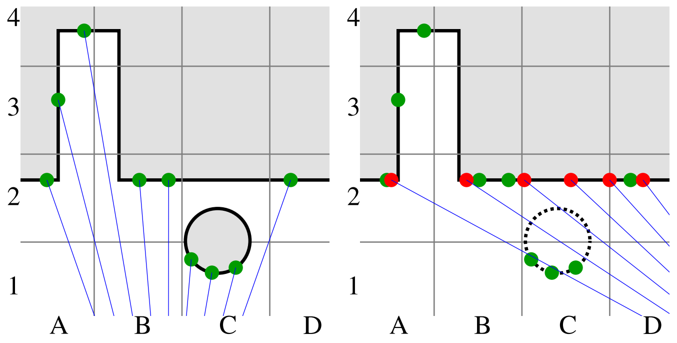

Fig. 2. The gray raster marks the 2D voxel boundaries. Blue lines mark the scanner lines of sight. Gray areas mark solid space while white areas mark free space. Round green and red dots represent the measured scan points of the first scan (left) in green and of the second scan (right) in red. The circular object in areas C1 and C2 is dynamic and only measured by the first scan (green points in C1 on the left). Dotted lines represent the former location of the circular object.

The authors of [6] and [7] also use a voxel data structure to distinguish between static and dynamic sections of the recorded point cloud but instead of recording free voxels, they count how often a voxel is occupied. Due to varying occlusion they have to make a number of assumptions about their environment and employ several heuristics that are not necessary with our algorithm. Furthermore, their approach requires a ground surface estimation — in contrast to our approach which does not require any such planar features to be present.

The creators of OctoMap [8] also use the same algorithm as we do by Amanatides and Woo [9] to cast rays. But instead of voxels they use an oct-tree data structure to find free volumes. They also employ a similar approach to avoid marking volumes as free in situations where rays meet a surface at a shallow angle by grouping multiple scan slices together. We improve on their work by generalizing their approach for scan slices to terrestrial scans.

III. General Design

The central component of our method is a global occupancy grid which we store as a voxel data structure. Each voxel holds a set of scan identifiers. A scan identifier is added to a given voxel if any point of that scan falls into the voxel. Thus, precise point coordinates are not stored in the grid. Instead, the data structure represents the union of all voxels that the input scans measured points in. For example in Figure 2 on the right, voxel B2 stores the information that it contains points from the green as well as from the red scan but neither their number nor the coordinates of these points is stored in the voxel grid. Thus, the global occupancy grid typically requires orders of magnitude less memory than the sum of the input data.

By traversing the occupancy grid from each sensor origin to the coordinates of each measured point, we find voxels that intersect with the line of sight of the sensor but contain a non-empty set of scan identifiers. These voxels are then classified as “see-through” or “dynamic”. At the end of the algorithm, this information serves as a binary classifier determining whether a given input point should be removed or not. A point is removed if it falls into a voxel that was marked as “see-through”.

Consider Figure 2. It displays the general idea of our algorithm in a two-dimensional scenario. The circular object in voxels C1 and C2 on the left is dynamic and not present anymore at the time of the second (red) scan on the right. Thus, only the first (green) scan measures parts of its surface. Since the second scan measures the red points in A2 and B2 with a line of sight that intersects C1, voxel C1 is classified as “see-through”. This in turn lets us deduce that the three green points measured in C1 belong to a non-stationary object.

The reason why we store a set of scan identifiers in each voxel instead of just storing a binary occupied/unoccupied property is to be able to abort voxel traversal early and avoid self-intersections. The point measured in voxel A4 in Figure 2 on the left has a line of sight intersecting with at least three voxels that must not be marked as free: A3, B3, and B2. To avoid wrongly marking these voxels as free, voxel traversal is aborted once a voxel is encountered containing the same scan identifier as the scan the current target point belongs to. This means that the ray toward voxel A4 aborts before marking voxel B2 as free. Another application for storing sets of scan identifiers in each voxel is our solution to achieve sub-voxel accuracy as explained in section VIII.

Figure 2 also shows how the algorithm does not remove points from voxels that were only visible in a single scan. For example the green points in voxels A3 and A4 belong to a part of the structure that is only seen by the left scan. Still, they are not removed because these voxels are never marked as “see-through” by another scan. The same holds for the red points in voxel C2. Their voxel is only seen by the second scan because the circular moving object in C1 and C2 temporarily occludes the this area during the first scan. Still, the points remain classified as static because their containing voxel is never marked as “see-through”.

IV. FAST VOXEL TRAVERSAL

To enumerate all “see-through” voxels from the laser origin up to the measured point, we used the algorithm proposed by Amanatides and Woo [9]. We improve the algorithm by making it adhere to a stricter definition of what it means for a ray to intersect with a voxel, by eliminating accumulation of floating point errors and by adding support for rays starting exactly at a voxel boundary. None of the existing open-source implementations (Octomap [8], MRPT [10], PCL [11], yt [12]) supports any of these properties and will result in false positives as well as false negatives in certain situations.

Even though our additions increase the number of instructions per loop cycle we were unable to measure a difference in runtime of the algorithm compared to the unaltered original implementation.

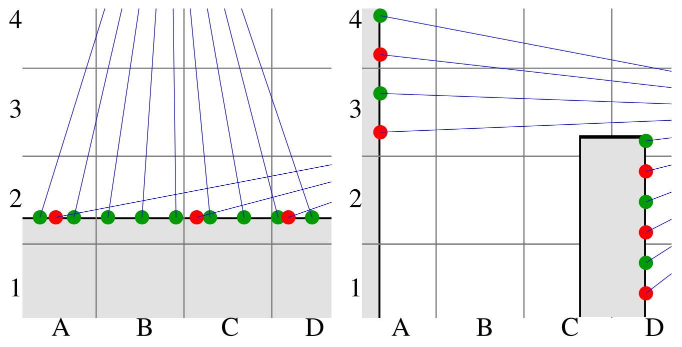

Fig. 3. Two examples for false positive when applying a naive approach to find dynamic voxels. See Figure 2 for a legend. Left: Due to the surface being scanned at a shallow angle, the scan measuring the red points will wrongly mark voxel B2 as free when traversing the line of sight up to the red point in A2. Right: Due to the tip of the structure in D3 only measured by the green scan, it will be wrongly marked as free when traversing the line of sight up to the red point in A3.

V. PANORAMA SCANS FROM TERRESTRIAL MAPPING

In [13] we applied our approach to the mobile mapping scenario to clean scans of a factory production line. The algorithm relied on the knowledge that subsequent scan slices are also spatially close to each other. In this paper we present a more general solution which drops this requirement. It is thus able to process terrestrial scan data which typically comes without any particular temporal or spatial ordering relationship.

Since our method identifies dynamic voxels, we call true positives all voxel that have correctly been classified as dynamic. Thus, false positives are voxels classified as dynamic even though they contain static points. Attempting to directly apply our method for mobile mapping to terrestrial scans will result in false positives due to the alignment of the voxel grid relative to the point cloud as well as due to heterogeneous sampling of surfaces by a scanner (depending on distance and incident angle). Two examples for these situations are shown in Figure 3. The problem is caused by the laser beam traversing voxels that were not sampled by the same scan and thus they are wrongly marks as “see through”.

The problem would be solved if we knew up to which range it is safe to traverse the voxel grid toward a specific point without generating these false positives. To compute these safe maximum search distances, we create the concept of points “shadowing” other points. As with real shadows, the size of the shadows we compute is depending on the distance of the shadowing point from the sensor origin. The volume that we choose to cast the shadow has the radius of one voxel diagonal. This is to account for all possible orientations of the voxel grid relative to the shadowing point. We do not clip the search distances exactly at the shadowing point but instead compute the surface normal at the shadowing point. The search distances toward the shadowed points are then clipped by the plane one voxel diagonal away from the surface that the shadowing point is part of. This is to cater for situations as shown in the left graphic of Figure 3.

Even with appropriate datastructures it would be computationally expensive to compute angular neighbors and surface normals for all points in a scan. Using the following approach, a full scene can be processed by only carrying out these computations for a small fraction of points. We start with sorting all points by distance from the scanner in ascending order. Then, starting with the closest point to the scanner, we find the points falling into its shadow and clip their distances. The algorithm then continues with the next closest not-yet-processed point. By not processing points which already fell into the shadow of another (closer) point, only few computations are required even for large scans because usually few points in the "foreground" shadow many points in the "background". Real world numbers are listed in Table I. The column "normals" displays the percentage of points for which shadowed points and the normal vector had to be computed.

Figure 4 illustrates our method. The closest point to the scanner is p and thus it is processed first. The algorithm computes all points shadowed by a sphere with the radius of one voxel diagonal and one voxel diagonal in front of p. These points shadowed by p are marked as magenta in the figure. Then, the shadowed points are used to compute the normal vector at p. The normal is in turn used to create a plane situated one voxel diagonal above p. That plane is then used to clip all the search distances to the points that p shadows (distances marked in orange). Since all the magenta points have been processed already, the algorithm continues with the next closest point which is one of the blue points.

VI. SPHERE QUADTREE

To efficiently compute points shadowed by another point under a given angle we make use of triangle quadtrees on a sphere surface, also known as hierarchical triangular meshes [14] or sphere quadtrees [15]. That data structure has so far mostly been used for Geographic Information Systems to model features on top of the earth surface [16] or in astronomy to map objects in the sky [17]. For our purposes we use it as a search tree to find all points in a certain angular neighborhood in a terrestrial panorama scan with an average lookup complexity of $O(\log n)$. We create one sphere quadtree per scan with its center at the origin of the scanner-local coordinate system.

Figure 5 on the left shows a visualization of a finished sphere quadtree for a terrestrial scan. Each triangle represents one node of the search tree.

VII. CLUSTERING FOR NOISE REMOVAL

If no good normal vector can be computed, our algorithm generates false positives. These are usually limited to single voxels at corners of static objects. On the other hand, most true positives belong to much larger structures and thus span many more connected voxels. To reduce the number of false positives we cluster all voxels marked as free. Each cluster consists of a group of connected voxels. We identify all clusters containing less than a certain threshold number of voxels. Voxels belonging to these small clusters are then marked as static as they are likely false positives.

VIII. SUB-VOXEL ACCURACY

In this section we present an algorithm that addresses a specific kind of false negatives our algorithm produces. In the common case where a dynamic object is seen directly adjacent to a static object, false negatives are introduced because the voxel grid is only traversed up to the maximum traversal range computed from the point shadows. For example, for a person standing on the ground, the person might be removed but their feet remain.

To avoid these false negatives we present an algorithm that is able to achieve sub-voxel accuracy: instead of marking a full voxel as dynamic and removing all points from it, we remove all points from the voxel that belong to the scan identifier producing the false negatives. All static points in the voxel will be kept and thus avoid the creation of any holes in the remaining point cloud.

We use Figure 6 to illustrate our approach. The leftmost graphic shows the original point cloud containing both the green and the red scan. The green scan only measures the static horizontal surface while the red scan also measures parts of the static surface as well as a dynamic vertical structure. After traversing the voxel grid to find voxels seen as free, voxels B2, and C2 got classified as "see-through" and points in them were removed (center graphic). What remains are false negatives in voxel B1 and C1. Classifying these voxels as dynamic is wrong because that would remove all points from them and create holes in the scan. We solve the problem by iterating over all voxels marked as free (voxels B2 and C2), identify the scans that got removed from them (the red scan) and then remove only points belonging to these scans from adjacent voxels. This will result in a situation as is shown in the right-most graphic: All red points are removed from voxels B1 and C1 while the green points remain.

While our approach removes all false negatives, it does so at the cost of introducing some false positives. In the right-most graphic of Figure 6, some of the red points in voxels B1 and C1 should've remained because they are part of the horizontal surface. Thus, our approach to achieve sub-voxel accuracy implements a trade-off. We remove remaining false negatives at the cost of more false positives. We accept this trade-off because qualitatively speaking, for the purpose of "cleaning" a scan, the result shown in the right-most graphic is superior to the result in the center graphic. Even though we now classified too many points as dynamic, after removing them from the scene, there are still enough static (green) points left in the respective voxels to not create any "holes".

Figure 1 from the introduction shows this algorithm applied to real data. Some points on the ground are marked as dynamic (magenta points in the left graphic) but removing them doesn't negatively impact the remaining point cloud as is shown in the right graphic.

IX. RESULTS

We publish the source code for the "peopleremover" as part of *3DTK – The 3D Toolkit*[1]. For future verification and

[1]http://threedtk.de

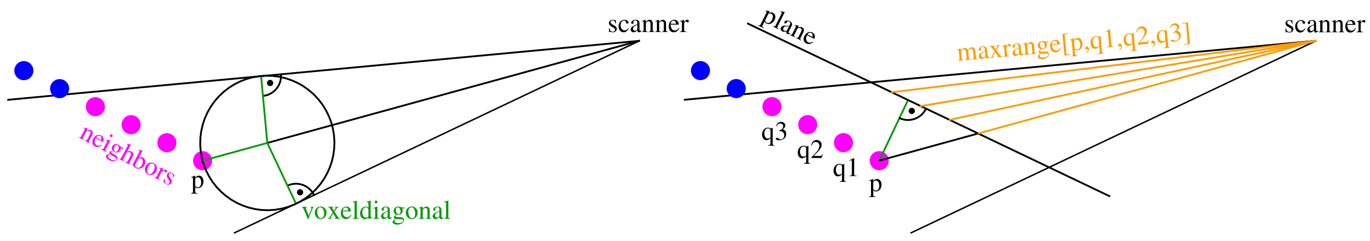

Fig. 4. A scanner on the right measured the points in blue and magenta on the left. The closest point to the scanner is p and points in the shadow of p are marked in magenta. Remaining points are marked in blue. Search distances are marked orange. Distances equal to one voxel diagonal are highlighted in green.

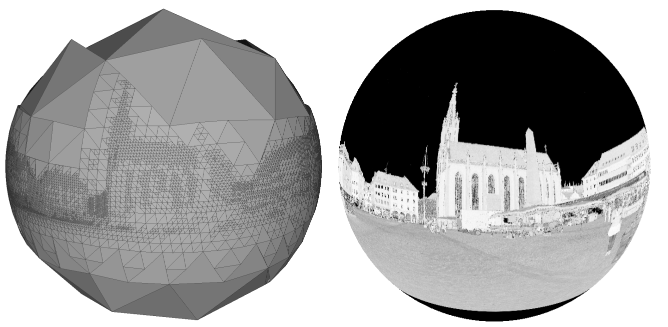

Fig. 5. Left: Visualization the nodes of a sphere octree from a terrestrial scan. Right: reflectance values of that scan as a texture on a sphere in the same orientation.

comparison, we also publish the scripts and datasets used to generate the measurements found in this section[2].

To quantitatively assess our method, we compare it to the approach by Underwood et al. [3]. Table I is an overview of the datasets we used. All points in the first four datasets were manually labeled with information whether they are dynamic, static or invalid. The points in the latter two datasets come without labelling and thus we cannot compute F_1-scores for them. The first three datasets were recorded by Underwood et al.[3]. while the remaining three were recorded by us using a Riegl VZ-400 laser scanner[4].

Since the method by Underwood et al. only works for scan pairs, we executed their method on all possible combination of pairs in each dataset and computed the final F_1-score from the sum of all the false positives and negatives from each comparison. To achieve best scores, we tested their method on multiple discrete T_a and T_r values and kept the best. We did the same for our method and tried different voxel sizes and display the best result. We ran both methods without clustering at the end to be able to compare the raw quality of both approaches. We didn't make use of our approach to achieve sub-voxel accuracy for these quantitative tests because, the number of newly introduced false positives results in worse F_1-scores.

We achieve similar F_1-scores on the synthetic "sim" dataset. False negatives are introduced in our method due to the alignment of the floor with the voxel grid, preventing a perfect score. Our method is outperformed in the "lab" dataset. The dataset is challenging because of its very noisy nature (see Figure 8 on the left) and because the dynamic objects are very small. Our algorithm correctly identifies the moving boxes in the "lab" dataset and does not introduce false negatives. But it generates comparatively large number of false positives on corners and edges of the environment. Since only 0.19% of all points in the dataset are labeled as dynamic, it only requires few voxels marked as false positives to produce a bad F_1-score. Our method slightly outperforms the approach by Underwood et al. in the "carpark" dataset. The best F_1-score we achieved for the carpark dataset with the Underwood method differs from the value they present in their paper because we used their full dataset including the last scan as well as all scan lines. Both methods result in equal scores on the "lecturehall" dataset.

Our algorithm produces false positives in situations where scans are either not correctly registered or due to sensor noise. An example is a flat surface where not all points lie on the surface. The points "in front" of the surface in scanner direction will then be marked as "see through" even though they belong to a static object. Another source of false positives arises when surface normals are wrongly computed and thus point shadows are not determined correctly. This in turn will lead to false positives as they were shown in Figure 3. Due to the very noisy nature of the "lab" dataset there were many sources of both of these issues, leading to a high number of false positives. Another source of false positives are mirrors and transparent objects. Lastly – if enabled – some false positives are introduced by our approach to subvoxel accuracy.

False negatives are created either in situations where a volume was only seen by a single laser scan or in volumes that were "shadowed" by closer points. We observed the latter problem in a dataset where we placed the scanner directly on the ground instead of on a tripod to take a scan. This resulted in points from the ground directly adjacent to the scanner to shadow most of the lower part of the scan and thus make it impossible for our algorithm to classify any points close to the ground as dynamic. Additionally, false negatives are introduced if the chosen voxel size is so small, that rays are able to penetrate objects without intersecting a voxel with points in it. Since the point density typically decreases with their distance from the sensor, this effect also occurs at very far distances. Applying a clustering filter can also introduce false negatives if the dynamic object is smaller than the chosen

[2]https://robotik.informatik.uni-wuerzburg.de/telematics/download/RA-L_2018/

[3]http://www.acfr.usyd.edu.au/papers/icra13-underwood-changedetection.shtml

[4]http://kos.informatik.uni-osnabrueck.de/3Dscans/

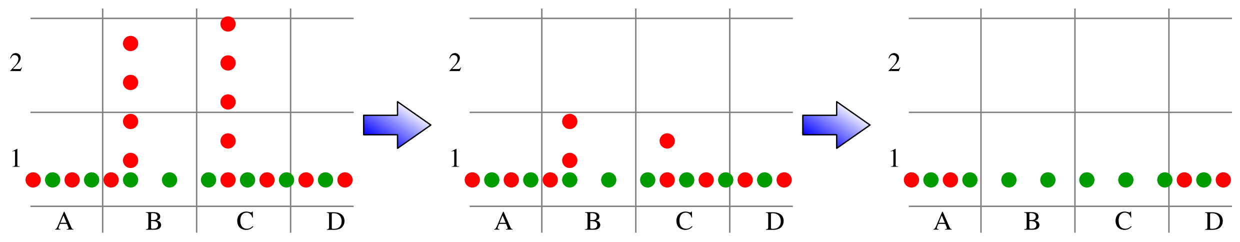

Fig. 6. Left: original two scans. Center: Voxels B2 and C2 found to be see through and cleared, artifacts remain in B1 and C1. Right: Red points from voxels adjacent to freed voxels B2 and C2 also get removed, leaving a smooth surface

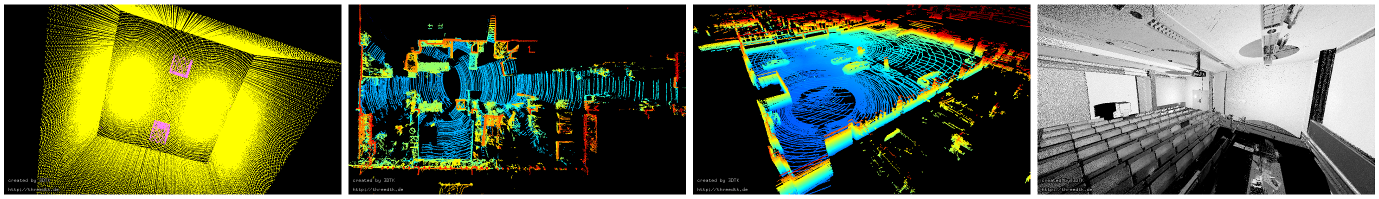

Fig. 7. Datasets from left to right: "sim" (yellow static points, magenta dynamic points), "lab" and "carpark" (both colored by height) by Underwood et al [3], and "lecturehall" (laser reflectivity as grayscale values).

TABLE I
COMPARISON OF F_1-SCORES AND RUNTIMES ACHIEVED BY OUR METHOD COMPARED WITH THE METHOD BY UNDERWOOD ET AL.

dataset			underwood					3dtk			
name	#points	#scans	#cmp	T_a	$T_r(m)$	runtime (s)	F_1-score	voxel size (m)	normals (%)	runtime (s)	F_1-score
sim	387838	8	28	1.4	0.1	25	**0.98**	0.6	8.03	**6**	**0.98**
lab	5815910	12	66	1.2	0.2	405	**0.71**	0.175	0.02	**29**	0.42
carpark	1965017	4	6	1.0	0.35	34	0.78	0.125	0.23	**23**	**0.83**
lecturehall	44574647	2	1	0.8	0.3	837	**0.96**	0.1	0.003	**687**	**0.96**
campus	2227455077	146	3456	0.8	0.3	12.8 days	n.a.	0.1	0.16	**13.1 hours**	n.a.
würzburg	86585411	6	15	0.8	0.3	7961	n.a.	0.1	0.21	**4967**	n.a.

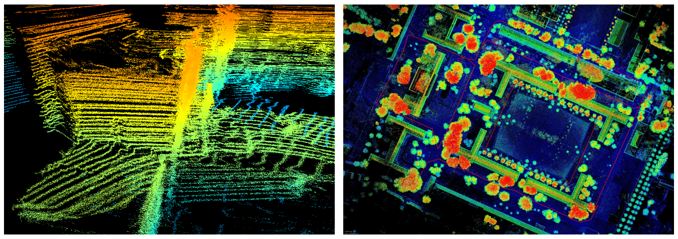

Fig. 8. Left: The "lab" dataset by Underwood et al. Right: The campus dataset from the top

minimum cluster size.

We also observe how the optimal input parameters to the algorithms T_a, T_r and the voxel size are different for each dataset despite the lab and the carpark dataset being recorded with the same sensor. More research is needed to determine if the input parameters may be predicted upfront without requiring manual labelling of a training dataset.

The column "normals" displays the percentage of points for which surface normal computation as part of finding the shadowed points was required. As detailed in section V, the computations have to be carried out for only a very small fraction of all input points.

The runtime measurements shown in Table I were obtained by timing the full execution pipeline. To speed up the approach by Underwood et al. we converted the original ASCII point cloud data files into their binary format. As the method by Underwood et al. is only able to compare pairs of scans, the runtime results for the "sim", "lab" and "carpark" datasets are not very meaningful. Our method easily outperforms theirs in terms of runtime because we apply their method on all possible combination of scan pairs, leading to $\frac{N(N-1)}{2}$ comparisons for N scans. The number of comparisons that were carried out is noted in the "#cmp" column. For a fairer comparison we recorded the "lecturehall" dataset. It only consists of two scans and thus allows to directly compare one run of the Underwood et al. method with one run of our approach. As listed in Table I, both approaches require a similar amount of time.

To also give evidence for our claim that the method by Underwood et al. performs slower for the purpose of "scan cleaning" on datasets with many scans, we used the "campus" dataset. Figure 8 on the right shows the dataset from above. That dataset consists of 146 scans with 15 Million points per scan on average for a total of 2.2 Billion points for the whole dataset. Comparing all possible scan pairs of this dataset would lead to 10585 comparisons. But since it doesn't make sense to compare scans that do not overlap in their observed volume we used a heuristic to discard all scan pairs that do not share a sufficiently large observed volume. Our heuristic uses the voxel datastructure that was already generated by the "peopleremover" to find those scans pairs. This heuristic under-approximates because ideally we are not only interested in the scans that measure points in a shared volume but also in the scan pairs where the free volume observed by one scan

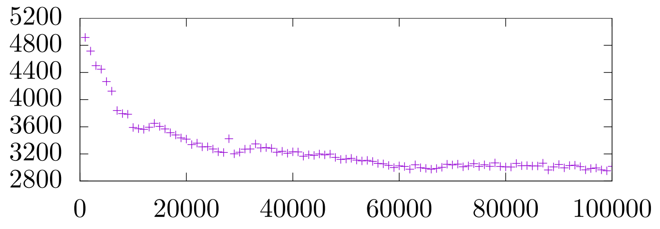

Fig. 9. The x-axis shows the number of scan slices passed to the algorithm. The y-axis shows the number of slices that the algorithm is able to process per second.

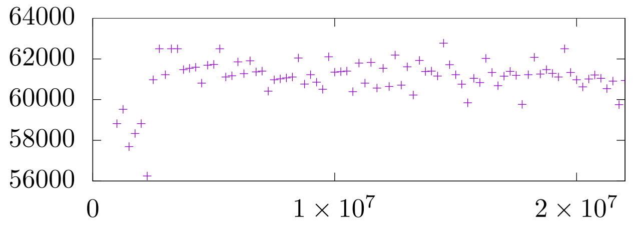

Fig. 10. The x-axis shows the number of points passed to the algorithm. The y-axis shows the number of points that the algorithm is able to process per second.

intersects with the measured points by the other. But even with this conservative heuristic, there exist 8372 scan pairs (79% of all possible scan pairs) in this dataset that share at least one 10 cm voxel with each other. This is explained by the large open spaces in the dataset. To further reduce the number of scan pairs that we choose for comparison with the algorithm by Underwood et al. we also discard all pairs that share less than 1000 voxel with each other. This leaves 3456 scan pairs to compare. Since the “campus” dataset does not contain any labels of dynamic objects, we re-used the parameters that worked best for the “lecturehall” dataset. The results shown for the “campus” dataset in Table I indicate, that the algorithm by Underwood et al. performs an order of magnitude slower in this task compared to our solution. The number of compared scan pairs could be further reduced but for the purpose of “scan cleaning”, the fewer comparisons are made, the more false negatives will be introduced in situations where a volume is seen as occupied by most scans and only seen as free by a few.

Our approach allows trading solution quality for runtime. For example, if the “lecturehall” dataset were processed with a voxel size of 17.5 cm instead of 10 cm as shown in Table I, then the F_1-score would only slightly decrease from 0.96 to 0.95 but computation time would be cut by 18% down to 567 seconds.

The voxel traversal algorithm is inherently well suited for multithreading. Even though the performance of our approach scales linearly with the number of threads we executed all benchmarks in single-threaded operation for better comparability.

We also investigated the runtime dependency of our algorithm on the number of input scans and the number of input points. For the former we used a dataset which we obtained via mobile mapping from an automotive production line [13]. We recorded the run time of our algorithm for 100 different input sizes from just 1000 scan slices up to 100000 scan slices. The scan slices contain 877 points on average. Since the performance of the algorithm depends on the geometry of the point cloud, we took 130 samples from random locations in our test data for each of the 100 different input sizes and then used the median run time.

The results of our measurements are shown in Figure 9. Due to the asymptotic behaviour toward larger inputs we deduct that our algorithm scales linearly with the number of input scan slices.

We conducted a similar experiment to find the dependency of the algorithm runtime from the number of input points. We randomly sampled the first scan of the “lecturehall” dataset to obtain input point clouds ranging from 1 million up to 22 million points and then executed our method on each of the resulting point clouds. The results are shown in Figure 10 and again indicate a linear relationship.

For qualitative results, we recorded the “würzburg” dataset. We registered the dataset using slam6D from 3DTK. It contains six scans of the Würzburg marketplace with 86 million points in total. Table I displays runtime results for both algorithms on that dataset. A fly-through video displaying the result of our method is available together with this publication or online[5]. Some stills are shown in Figure 11.

X. Future Work

So far we use C++ standard library data structures to store the voxel grid in memory. But while this enabled us to quickly assess the viability of our approach, replacing our current voxel data structure with a more efficient one will not only boost our runtime but also significantly reduce memory consumption. Specifically we want to evaluate using OctoMap [8], our own Octree implementation [18] as well as sparse voxel DAGs [19]. Our current approach holds the complete point cloud data in memory but the only data structure that has to stay in memory at all times is the voxel occupancy grid. Especially by using memory saving data structures like sparse voxel DAGs we could load point cloud data only on demand and thus be able to process much larger data sets than we are able to process right now.

XI. Conclusions

We presented the “peopleremover”, an approach specifically tailored to removing dynamic portions of 3D point cloud data. Our solution is suitable for scan slices from mobile mapping as well as for terrestrial scan data. We show experimental evidence that our approach compares favourably in quality to an existing solution for scan pairs and is only outperformed on a very noisy dataset. In terms of runtime our method is superior as it compares arbitrarily many scans with linear complexity.

Acknowledgment

We thank James Underwood and Vsevolod Vlaskine for a very fruitful exchange of ideas on the topic of explicit change detection in point cloud data.

[5]https://robotik.informatik.uni-wuerzburg.de/telematics/download/RA-L_2018/flythrough.mp4

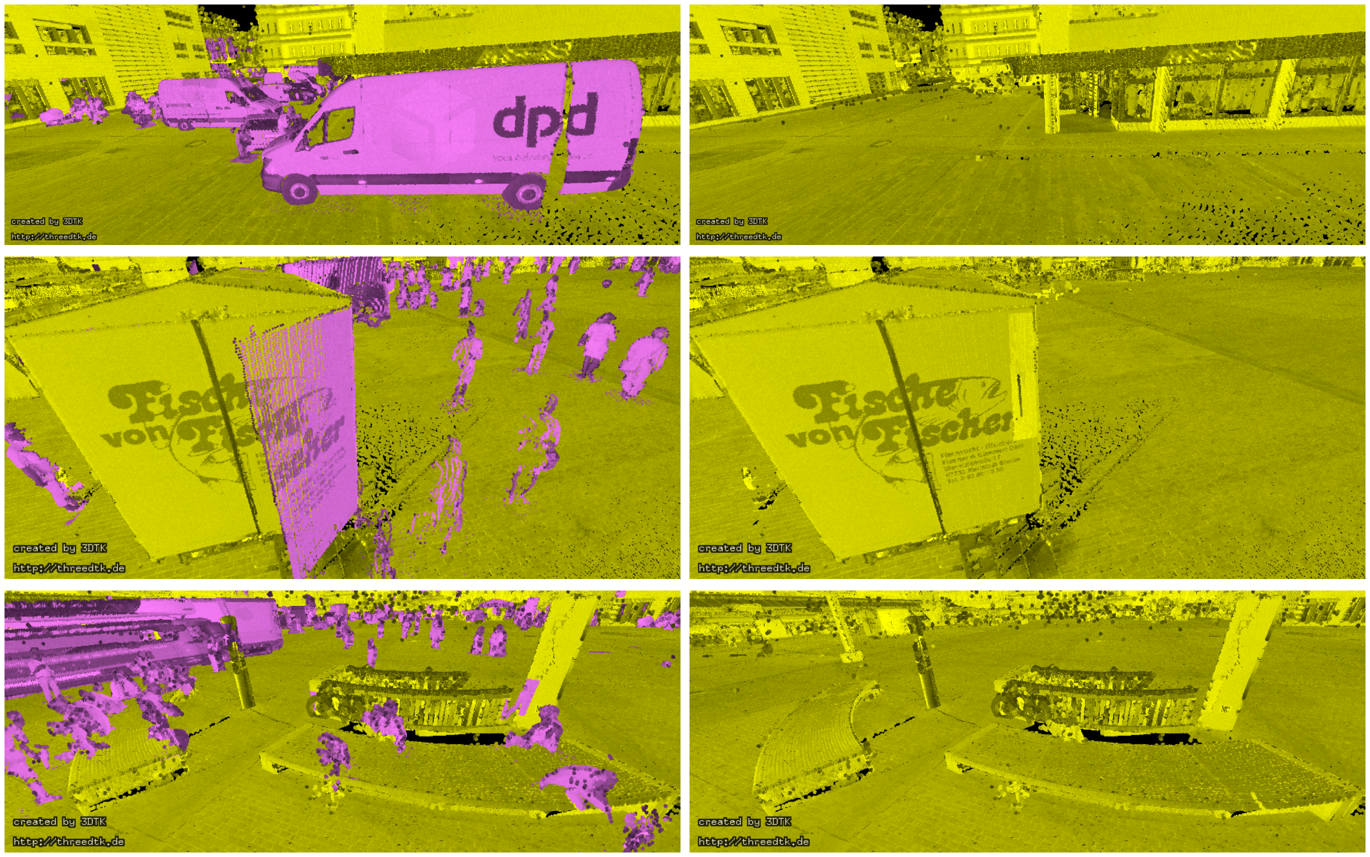

Fig. 11. Each row shows the same camera position. Left column: Static points in yellow and dynamic points in magenta. Right column: scene with only static points.

References

[1] R. Qin, J. Tian, and P. Reinartz, "3d change detection–approaches and applications," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 122, pp. 41–56, 2016.

[2] A. W. Vieira, P. L. Drews, and M. F. Campos, "Spatial density patterns for efficient change detection in 3d environment for autonomous surveillance robots," *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 3, pp. 766–774, 2014.

[3] J. P. Underwood, D. Gillsjö, T. Bailey, and V. Vlaskine, "Explicit 3d change detection using ray-tracing in spherical coordinates," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 4735–4741.

[4] W. Xiao, B. Vallet, and N. Paparoditis, "Change detection in 3d point clouds acquired by a mobile mapping system," *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 1, no. 2, pp. 331–336, 2013.

[5] W. Xiao, B. Vallet, M. Brédif, and N. Paparoditis, "Street environment change detection from mobile laser scanning point clouds," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 107, pp. 38–49, 2015.

[6] A. Asvadi, P. Peixoto, and U. Nunes, "Two-stage static/dynamic environment modeling using voxel representation," in *Robot 2015: Second Iberian Robotics Conference*. Springer, 2016, pp. 465–476.

[7] A. Asvadi, C. Premebida, P. Peixoto, and U. Nunes, "3d lidar-based static and moving obstacle detection in driving environments: An approach based on voxels and multi-region ground planes," *Robotics and Autonomous Systems*, vol. 83, pp. 299–311, 2016.

[8] 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.

[9] J. Amanatides, A. Woo *et al.*, "A fast voxel traversal algorithm for ray tracing," in *Eurographics*, vol. 87, no. 3, 1987, pp. 3–10.

[10] J. Blanco-Claraco, "Mobile robot programming toolkit (mrpt)," 2014. [Online]. Available: https://www.mrpt.org

[11] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–4.

[12] M. J. Turk, B. D. Smith, J. S. Oishi, S. Skory, S. W. Skillman, T. Abel, and M. L. Norman, "yt: A Multi-code Analysis Toolkit for Astrophysical Simulation Data," *The Astrophysical Journal Supplement Series*, vol. 192, p. 9, Jan. 2011.

[13] J. Schauer and A. Nüchter, "Digitizing automotive production lines without interrupting assembly operations through an automatic voxel-based removal of moving objects," in *Control & Automation (ICCA), 2017 13th IEEE International Conference on*. IEEE, 2017, pp. 701–706.

[14] A. S. Szalay, J. Gray, G. Fekete, P. Z. Kunszt, P. Kukol, and A. Thakar, "Indexing the sphere with the hierarchical triangular mesh," *arXiv preprint cs/0701164*, 2007.

[15] G. Fekete, "Rendering and managing spherical data with sphere quadtrees," in *Proceedings of the 1st Conference on Visualization'90*. IEEE Computer Society Press, 1990, pp. 176–186.

[16] M. F. Goodchild and Y. Shiren, "A hierarchical spatial data structure for global geographic information systems," *CVGIP: Graphical Models and Image Processing*, vol. 54, no. 1, pp. 31–44, 1992.

[17] T. Budavári, A. S. Szalay, and G. Fekete, "Searchable sky coverage of astronomical observations: Footprints and exposures," *Publications of the Astronomical Society of the Pacific*, vol. 122, no. 897, p. 1375, 2010.

[18] J. Elseberg, D. Borrmann, and A. Nüchter, "One billion points in the cloud–an octree for efficient processing of 3d laser scans," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 76, pp. 76–88, 2013.

[19] V. Kämpe, E. Sintorn, and U. Assarsson, "High resolution sparse voxel dags," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, p. 101, 2013.