Int. J. Appl. Math. Comput. Sci., 2016, Vol. 26, No. 1, 99–122
DOI: 10.1515/amcs-2016-0007

EFFICIENT GENERATION OF 3D SURFEL MAPS USING RGB–D SENSORS

ARTUR WILKOWSKI [a,*], TOMASZ KORNUTA [b], MACIEJ STEFAŃCZYK [b], WŁODZIMIERZ KASPRZAK [b]

[a] Industrial Research Institute for Automation and Measurements
Al. Jerozolimskie 202, 02-486 Warsaw, Poland
e-mail: A.Wilkowski@gik.pw.edu.pl

[b] Institute of Control and Computation Engineering
Warsaw University of Technology, ul. Nowowiejska 15/19, 00-665 Warsaw, Poland
e-mail: {T.Kornuta,M.Stefanczyk,W.Kasprzak}@elka.pw.edu.pl

The article focuses on the problem of building dense 3D occupancy maps using commercial RGB-D sensors and the SLAM approach. In particular, it addresses the problem of 3D map representations, which must be able both to store millions of points and to offer efficient update mechanisms. The proposed solution consists of two such key elements, visual odometry and surfel-based mapping, but it contains substantial improvements: storing the surfel maps in octree form and utilizing a frustum culling-based method to accelerate the map update step. The performed experiments verify the usefulness and efficiency of the developed system.

Keywords: RGB-D, V-SLAM, surfel map, frustum culling, octree.

1. Introduction

Industrial robots operate in structured, fully deterministic environments, thus they usually do not need an explicit representation of the environment. In contrast, the lack of up-to-date maps of the surroundings of service and field robots might cause serious effects, including the damaging of both robots and objects, or even injuring people. Besides, the diversity and complexity of tasks performed by such robots requires several types of maps and different abstraction levels of representation. For example, simple geometric (occupancy) maps enable robots to avoid obstacles and collisions, whereas semantic maps allow reasoning, planning and navigation between different places (Martínez Mozos *et al.*, 2007). A semantic map combines a geometric map of the environment with objects recognized in the image (Kasprzak, 2010; Kasprzak *et al.*, 2015).

Multiple sensor readings (images, point clouds) are required to generate a geometric map of the environment. Theoretically, the coordinate system transformation between a pair of readings can be obtained in advance, i.e., from the wheel odometry of mobile robots or from the direct kinematics of manipulators. However, inaccurate robot motion models (wheel drifts, friction, imprecise kinematic parameters, etc.) lead to faulty odometry readings, resulting in map inconsistencies. To overcome these effects, algorithms were developed to estimate the robot's trajectory directly from sensor readings. They are part of a solution named "simultaneous localization and mapping" (SLAM) (Thrun and Leonard, 2008). There are many flavours of SLAM systems, using diverse methods of estimation of the robot and environment state, as well as applying all kinds and combination of sensors, e.g., monocular vision in an probabilistic, EKF-based framework (Skrzypczyński, 2009). In particular, a specialized term "visual SLAM" (V-SLAM in short) was coined to distinguish the SLAM systems relying mainly on cameras, which relates to the fact that the estimation of subsequent poses of the moving camera is called "visual odometry" (VO) (Nistér *et al.*, 2004).

The recent advent of cheap RGB-D sensors accelerated the V-SLAM research. An RGB-D sensor provides two concurrent, synchronized image streams—colour images and depth maps, which represent

*Corresponding author

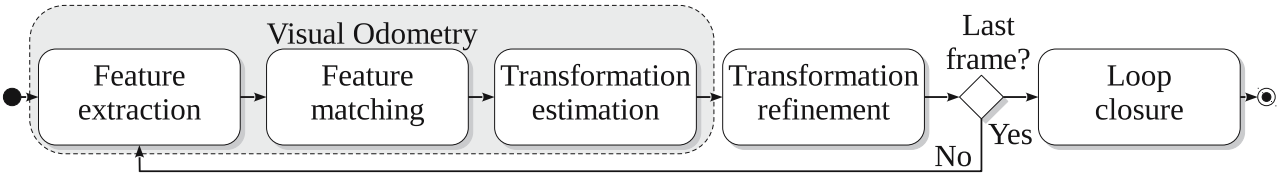


Fig. 1. Components of an example RGB-D-SLAM system.

the colour of observed scene points and their relative distance to the camera—that subsequently can be combined into point clouds. Those clouds contain typically hundreds of thousands of points, acquired with with a high-framerate, while the mapped areas are usually complex. Thus, several efficiency and quality issues in V-SLAM must be addressed:

- precision and computational efficiency of point cloud registration,
- global coherence of maps,
- efficient and precise representation of a huge number of points.

In this paper, the focus is on the third issue. There is a proposition developed for an efficient representation of a precise three-dimensional map of the environment. An "efficient representation" means it is compact (there are few spurious points) and can be rapidly accessed and updated.

A 3D occupancy map can be represented in several ways:

1. "voxel" representation—the 3D space is divided into small "cubes" (Marder-Eppstein *et al*., 2010),
2. "surfel" representation—the space is modelled by an arrangement of surface elements (Krainin *et al*., 2010),
3. MLS (multi-level surface)—contains a list of obstacles (occupied voxels) associated with each element of a 2D grid (Triebel *et al*., 2006),
4. octal trees (octrees)—constitute an effective implementation of the voxel map using the octal tree structure (Wurm *et al*., 2010),
5. combined voxel-surfel representation (e.g., the multi-resolution surfel map).

The core of the proposed system is a map of small surface patches of uniform size (called surfels), integrating readings of an RGB-D sensor. The utilization of a surfel map has several advantages (Weise *et al*., 2009; Henry *et al*., 2014). Firstly, it is a compact representation of a sequence of point clouds being merged into surfels. Secondly, the surfel update procedure takes the sensor's current pose into account, allowing both visibility checks and reduction of depth quantization noise—the most prominent type of noise in cheap RGB-D sensors. Finally, every surfel can be processed independently, simplifying and speeding up the map update procedure in comparison with a mesh-like representation.

The existing solutions for surfel map creation are not speed-optimized. A single update of the map might take even several seconds (Henry *et al*., 2012). In this paper, two substantial efficiency improvements are proposed:

1. utilization of the "frustum culling" method, based on an octree representation, to accelerate surfel updating;
2. replacement of a dense ICP algorithm by a much more efficient sparse-ICP method, as proposed by Dryanovski *et al*. (2013).

The article is organized as follows. In Section 2 the recent developments in the field of RGB-D-based mapping are presented. A structure of a typical V-SLAM system is described (Section 2.1), followed by a survey of its key steps: the feature extraction (Section 2.2), feature matching (Section 2.3) and the estimation of camera motion (Section 2.4) in visual odometry. Then, solutions to dense point clouds alignment (Section 2.5) and global map coherence (Section 2.6) are explained. In Section 3 the most popular 3D mapping systems based on RGD-D V-SLAM are reviewed. Section 4 describes the proposed solution, starting from its important data structures, followed by a short overview of the system and operation details. In Section 5, the ROS-based system implementation is shortly outlined. Section 6 presents experiments performed for the validation of developed solution. Conclusions are given in Section 7. Appendix contains a theoretical primer on "frustum culling".

2. RGB-D-based V-SLAM

2.1. Structure of the RGB-D-SLAM system. A state diagram of an example RGB-D-SLAM system is presented in Fig. 1. First, the camera motion (the transition between consecutive poses) is estimated. Typically, it is based on matching features extracted from both current and previous images, finding parts of both images corresponding to the same scene elements. There

are algorithms based only on RGB image features (either for a single camera (Nistér *et al*., 2004) or a stereo-camera case (Konolige *et al*., 2011)), but the additional depth information makes the matching process much easier. As a result, a sparse 3D cloud of features is generated, allowing robust estimation of the transformation between camera poses.

Usually, a subset of detected features is sufficient to estimate the transformation. Thus, when taking numerous subsets, different estimates of the transformation are found. Typically this step is based on the RANSAC (random sample consensus) algorithm (Fischler and Bolles, 1981), which selects the transformation that is the most consistent with the majority of feature pairs.

The next step, transformation refinement is typically based on the ICP (iterative closest point) algorithm (Zhang, 1994). This is an iteration of least-square-error (LSE) solutions to the problem of dense point cloud correspondence.

The four steps above (feature extraction and matching, transform estimation and refinement) are repeated for every next RGB-D image. However, the optimal alignments between every two consecutive images can eventually lead to a map that is globally inconsistent. This is especially visible in the case of closed loops in the sensor's trajectory. In order to handle this issue, a loop closure procedure is performed. It refines both the 3D coordinates describing the scene geometry and the motion of the camera, using several or all views in the sequence.

In the following subsections, algorithms used in the above mentioned steps are presented.

2.2. Feature extraction. Two major categories of image features can be distinguished: global and local. The former represent certain characteristics calculated on the basis of a whole image. In the case of objects, global features can be extracted from the entire point cloud constituting the object or all image segments making it up (however, one must *a priori* determine which elements belong to a given object). In contrast, local features are associated with locations of the so-called keypoints—pixels in the image (or points in a 3D space) considered to be characteristic, hence important. Additionally, local features express the characteristics of the neighbourhood of this point in the form of a descriptor. Global features allow quick determination of the similarity of entire images or objects, while the local ones are useful for the comparison of individual parts. With respect to the alignment of two point clouds, only local features are of interest.

Furthermore, in the case of RGB-D images, another orthogonal division of features can be proposed: features extracted from the appearance (i.e., colour) and features extracted from the 3D shape (a depth map or a point cloud). An example of a global feature calculated from the RGB image is a color histogram of all the object points. A good example of a global feature extracted from the point cloud is the VFH (viewpoint feature histogram) (Rusu *et al*., 2010), which creates a histogram of angles between surface normal vectors for pairs of points constituting the object cloud.

In the case of local features, one should consider keypoint detectors and feature descriptors. Recently proposed detectors locating key points in RGB images include the FAST (features from accelerated segment test) (Rosten and Drummond, 2006) and the AGAST (adaptive and generic accelerated segment test) (Mair *et al*., 2010)), whereas sample novel descriptors are BRIEFs (binary robust independent elementary features) (Calonder *et al*., 2010) and the FREAK (fast retina keypoint) (Alahi *et al*., 2012).

There are also several features having their own detectors and descriptors, such as SIFT (Lowe, 1999) or BRISKs (binary robust invariant scalable keypoints) (Leutenegger *et al*., 2011). The latter possess the so-called "binary descriptor"—a vector of boolean values instead of real or integer numbers. It is worth noting that in recent years binary descriptors have become more popular, especially in real-time applications due to their very fast matching ability (Miksik and Mikolajczyk, 2012; Figat *et al*., 2014; Nowicki and Skrzypczyński, 2014).

Several approaches explore the depth information only, aiming at detecting keypoints (i.e., corners) by adapting well-known 2D methods, like the "Harris 3D"' detector (Sipiran and Bustos, 2011). Examples of local features extracted from point clouds include the SHOT (signature of histograms of orientation) (Tombari *et al*., 2010) and SC (3-D shape context) features (Frome *et al*., 2004). The NARF (normal aligned radial feature) (Steder *et al*., 2011) is both a keypoint detector and a descriptor, designed to work with depth maps.

If the information to be processed consists of an RGB image along with the aligned depth information, classic keypoint detectors can be used to extract points of interest based on color information and then employ their coordinates to extract descriptors from the depth data. Such an example is the "5D Harris" detector, which detects corners based on the gradient image (computed from the intensity image) and an image containing normal vectors to the surface (determined on the basis of a depth map). An example of an RGB-D-based descriptor is "color-SHOT" (CSHOT) (Tombari *et al*., 2011), an extension of the SHOT feature. The CSHOT descriptor consists of a histogram of orientations of points constituting the neighborhood of a given keypoint (identically as in the original SHOT approach), extended by the color histogram computed on the basis of the colour of these points.

2.3. Finding correspondences. The problem of finding the correspondences in two sets of points is typically solved by a "k-nearest neighbors" (k-NNs) algorithm. However, because feature descriptors are typically vectors of large, constant sizes (e.g., the SIFT descriptor is a single 128-dimensional vector), this is a high-dimensional problem. The most popular solution is the FLANN (Fast Library for Approximate Nearest Neighbors) (Muja and Lowe, 2009), an approximate nearest neighbour search algorithm. It utilizes multiple randomized kd-trees or a combination of hierarchical k-means trees with a priority search order (depending on input data). The same authors also proposed an improvement of the FLANN for fast approximate matching of binary features using parallel search of randomized hierarchical trees (Muja and Lowe, 2012).

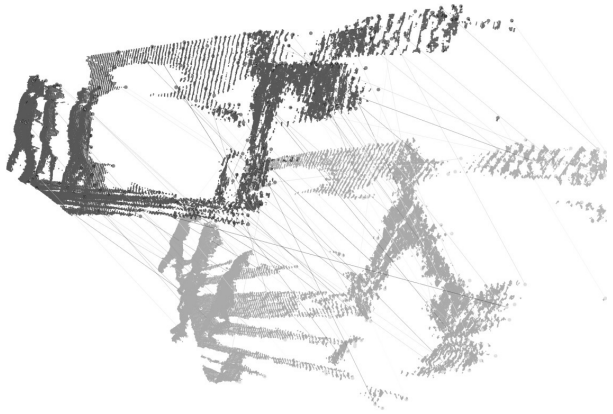


Fig. 2. Example of VO-based alignment of the "five people" cloud.

Figure 2 presents VO-based alignment of a sample cloud from the Point Cloud Library (PCL) (Rusu and Cousins, 2011), with marked features and feature correspondences.

2.4. Transformation estimation. A single point in the Cartesian space has only a "position" attribute; thus, for a given pair of corresponding points a translation vector can be computed. Hence, in contrary to the estimation of the epipolar geometry from 2D point correspondences, the estimation of the full transformation (i.e., translation along with rotation) between two point clouds requires three 3D point correspondences (assuming non-collinear points). To find which of the many possible correspondences is the best, a RANSAC-based (Fischler and Bolles, 1981) algorithm is typically used. In every iteration, the transformation is computed on the basis of three randomly selected point correspondences. Next, the remaining correspondences are validated as to whether they fit into this transformation, thus forming two sets: inliers (the fitting ones) and outliers (the correspondences that do not fit). The procedure is repeated for a given number of iterations or until the sum of estimation errors falls below a given threshold.

2.5. Transformation refinement. The initial transformation, computed on the basis of sparse clouds of features, is typically refined with the use of the "iterative closest point" (ICP) algorithm (Zhang, 1994). It improves the initial transformation while evaluating the alignment of dense point clouds created on the basis of RGB-D images. The principle of ICP is to refine the transformation iteratively by minimizing the mean square error between the new cloud transformed into the original reference frame and the original cloud. There are many different flavours of ICP (Pomerleau *et al*., 2013), like utilizing additional information for both the nearest neighbour candidate selection of a given point (e.g., utilizing normal vectors (Censi, 2008)) or between point distance computation (e.g., metrics using both position and colour information (Men *et al*., 2011; Łępicka *et al*., 2016)).

2.6. Loop closure. When only two consecutive clouds are matched against each other, the matching errors tend to accumulate over time, giving significant discrepancies at the end of given sequence. This is especially visible when returning to the point of origin (closing a loop). Several approaches have been proposed to handle this so-called "drift" problem. In the following, three most popular methods are described.

Global path optimization using graph approaches. Vertices in a graph represent sensor positions and are associated with the collected point clouds. Edges represent constraints and are created when correspondences between particular clouds are obtained. To improve efficiency, only some selected point clouds are memorized, or only keypoints are stored, instead of full clouds. The graph is then optimized to find globally the best set of transformations between memorized point clouds. For this purpose, a global graph optimization method is applied, e.g., TORO optimization (Henry *et al*., 2012) or *g2o* optimization (Kummerle *et al*., 2011). The set of sensor positions is optimized by introducing artificial error measures to compare actual relations between sensor poses (as they are described by the parameter vector) and recorded pose differences acting as constraints.

Bundle adjustment. In the case of "bundle adjustment" methods, the optimization criterion is the reprojection error as used in Section 2.4. The basic idea is to perform the *transformation estimation* step with the correspondences extended to several consecutive frames

(images). If k is the average number of point correspondences between two frames and n is the number of frames used in bundle adjustment, the total number of point correspondences filling the optimization process is around $\frac{n(n-1)}{2}k$. Thus, the number of constraints grows quadratically with that of frames, while the number of parameters grows linearly only. This leads to a robust optimization task and an expected improvement of the camera pose path. One of the most successful algorithms is the "sparse bundle adjustment" (Konolige, 2010). The bundle adjustment procedure can be part of the *transformation estimation* step, or it can be a separate post-processing step.

Matching against the full map. An alternative solution for maintaining map consistency is to match the current frame features with the full map constructed so far, instead of matching them with features of one or several most recent frames (Dryanovski *et al*., 2013). Efficiency is ensured by storing in the map only selected keypoints from each frame. However, this approach deteriorates for loops of arbitrary length, and therefore it is usually accompanied by a global graph optimization step.

3. RGB-D-based mapping systems

Recently, several systems have been proposed for V-SLAM and 3D occupancy mapping using RGB-D sensors. The most successful ones include

- RGB-D mapping (Henry *et al*., 2012) (Peter Henry, University of Washington, Seattle),
- KinectFusion (Izadi *et al*., 2011) (Microsoft Research, UK) and Kintinuous (MIT) (Whelan *et al*., 2013),
- RGB-D SLAM (Endres *et al*., 2012) (University Freiburg, TU Munchen),
- fast visual odometry and mapping from RGB-D data (Dryanovski *et al*., 2013) (The City University of New York).

In the case of the latter three, there are implementations available in the ROS (Quigley *et al*., 2009) or PCL (Rusu and Cousins, 2011). Short descriptions of the listed solutions follow below.

3.1. RGB-D mapping. A characteristic feature of the RGB-D mapping system presented by Henry *et al*. (2012) is the surfel representation of the occupancy map. The initialization of a two point cloud correspondence is based on SIFT descriptors of key image points. After the initial alignment, the solution is fine-tuned using the ICP algorithm, which optimizes a double criterion encompassing the point-to-plane distance for the dense cloud and the distance of matched SIFT points (using elastic weights for both the components).

During map creation, a graph of key frames is prepared containing transformations between frames and probabilistic constraints. Graph optimization (loop closure) is performed with the TORO method (which is actually a maximum likelihood estimator). It is worth mentioning that sparse bundle adjustment is another option evaluated in this work. System performance is described to be satisfactory; however, it is far from real-time: cloud alignment takes about 1 s, whereas the surfel update step takes 3 s per frame.

3.2. KinectFusion. The KinectFusion method is optimized for utilization of GPU and real-time processing. It uses a simplified point-to-plane version of the ICP algorithm, which results from GPU processing restrictions. This boosts the performance, but clouds must be sampled frequently enough.

The map is represented as a set of voxels. However, the voxels are interpreted in terms of the truncated signed distance function (TSDF) and contain the truncated distance from the closest surface. The voxel map is stored in GPU; thus the map size is limited by the GPU memory. This limitation was overcome in the Kintinuous method (Whelan *et al*., 2013), where the map is transferred to and from external memory when necessary. In the latter approach, ICP is supported also by the visual odometry system FOVIS.

The reduction of noise and measurement errors is performed by averaging TSDF values. The map is represented as a set of voxels; however, for visualization, ray-casting is used to extract surfaces (by detecting zero-crossing of the TSDF). The method works in real time (thanks to GPU).

The results of some of our experimental mapping with the KinectFusion algorithm are given in Fig. 3.

3.3. RGB-D SLAM. The RGB-D SLAM system is organized in a fairly uncomplicated way. It uses only matchings between keypoint descriptors SIFT, SURF, ORB (and their GPU versions) in order to compute transformation between two frames. Application of keypoints cannot provide very accurate results, thus loop closure algorithms are used extensively to achieve a globally optimal solution. New frames are compared with a group of previous frames (some of them are selected deterministically and some randomly). The frames and relations between them constitute vertices and edges of the graph. The graph is then optimized using the *g2o* framework. The method was implemented in the ROS framework (Quigley *et al*., 2009) and cooperates with the ROS OctoMap (Hornung *et al*., 2013) package for creation of the occupancy map. A sample map, which

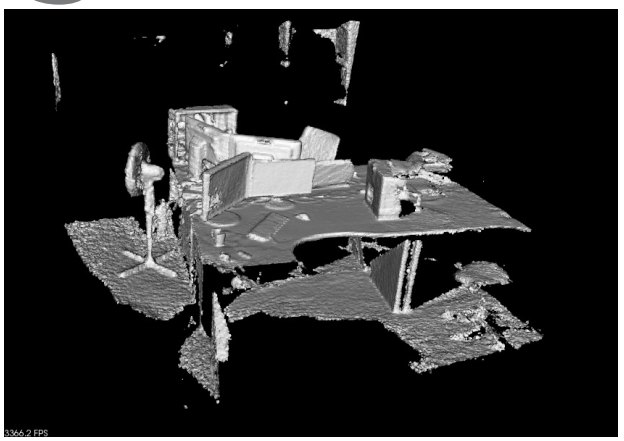


Fig. 3. 3D scene of "a desk in a room" registered from multiple viewpoints using the KinectFusion algorithm.

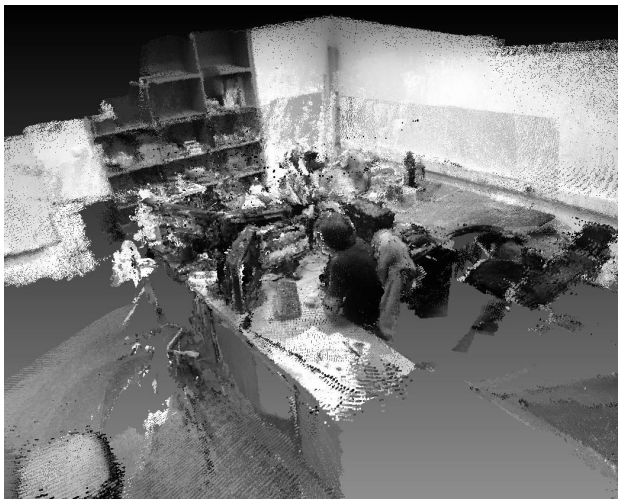


Fig. 4. On-line updating of the depth map for the "a desk in a room" scene using the RGB-D SLAM method.

is the result of our mapping experiments with RGB-D SLAM, is presented in Figs. 4 and 5.

3.4. FVOM: Fast visual odometry and mapping. A characteristic feature of the approach proposed by Dryanovski *et al*. (2013) is that the ICP algorithm does not work on two most recent points clouds, but instead aligns the newest point cloud to the complete model of the scene constructed from point clouds collected so far. In order to do it efficiently, only selected keypoints from each frame are taken into account and incorporated into the model (the full clouds can be stored in external memory).

For each frame, Shi–Tomasi keypoints (Shi and Tomasi, 1994) are extracted from the RGB image (also other keypoints are supported), but the original descriptor is replaced with a new one, containing the

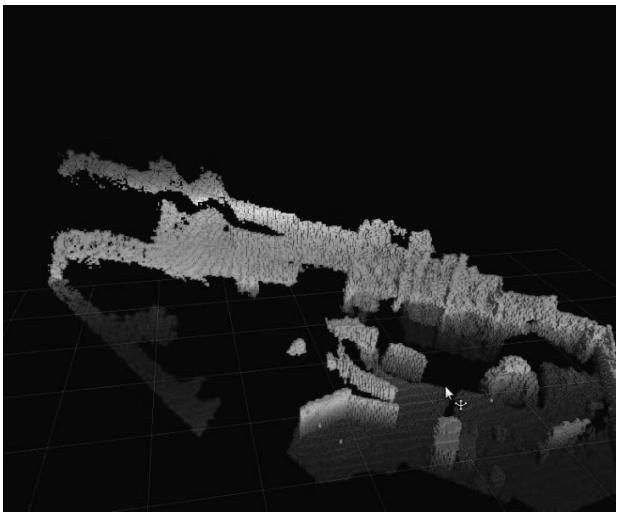


Fig. 5. Depth map exported to OctoMap for the "a desk in a room" scene using the RGB-D SLAM method.

mean Cartesian position of the keypoint along with the position variance, computed on the basis of the proposed model of sensor measurement uncertainties (depending on distance measurements). The resulting sparse cloud is aligned against the current model of the scene using the ICP. After alignment, the new cloud is incorporated into the model. The assimilation process takes into account point uncertainties and uses Kalman filter update rules to propagate uncertainties from measurements into the model. This probabilistic framework is to some extent used also for computation of distances between features in the ICP algorithm.

The alignment of the new cloud against the whole model decreases the extent of alignment error propagation (the drift). However, in new implementations (under the ROS), the basic algorithm is supplemented also with *g2o* global optimization for off-line processing. On-line processing is very fast. The average processing time of a single frame is about 16.1 ms.

Mapping results that we obtained using the FVOM method are presented in Figs. 6 and 7. Figure 6 presents keypoints captured during the on-line processing, whereas Fig. 7 presents the final integrated cloud optimized with the *g2o* algorithm.

4. Proposed 3D mapping system

In this section the architecture and steps of the proposed 3D mapping system are presented. It creates a 3D surfel map and makes updates of it using the incoming RGB-D sensor frames. The main advantage of such a surfel map is that it provides significant data compression with only a very small data loss. What is even more important, the quantization noise of the sensor is also kept under control,

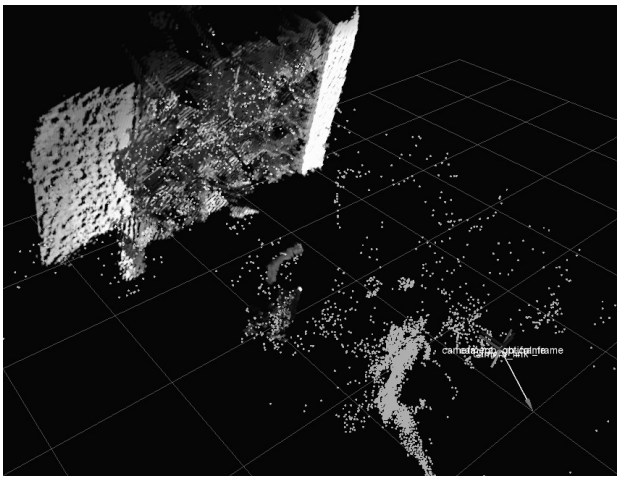


Fig. 6. Keypoints generated for the "a desk in a room" scene during on-line registration using the fast visual odometry algorithm.

since the surfel-scan averaging procedure is performed mainly along the depth axis.

4.1. System architecture. The data-flow diagram of our system is given in Fig. 8. The data obtained from the RGB-D sensor are first processed by the *Visual odometry* module in order to obtain a sequence of sensor poses. The next module, the *Keyframe generator*, selects keyframes from the input sequence of frames. The *Loop-closure* module takes as inputs the generated keyframes and estimated poses, and performs global sensor path optimization.

The generated keyframes and refined sensor poses

Fig. 7. G2o-optimized depth map for the "a desk in a room" scene using the fast visual odometry algorithm.

are input data for the *Surfel update* and *Surfel addition* modules, which modify the *Surfel map* data container. The *Surfel update* module modifies existing scene surfels based on sensor readings, while the *Surfel addition* module adds new surfels in unoccupied locations.

The *Frustum culling* module works as an intermediary between the *Surfel map* container and the *Surfel update* module and efficiently filters surfels that cannot be currently observed by the sensor, thus reducing the workload of the *Surfel update* module.

The above mentioned system modules are described further on in this section.

4.2. Novelty of the approach. The proposed approach to surfel mapping follows to some extent solutions described by Krainin *et al*. (2010) and Henry *et al*. (2012); however, it contains some novel elements that provide an efficiency improvement. The first improvement is the acceleration of the surfel map update procedure thanks to "octree-based frustum culling".

Octree-based frustum culling applied for map updating. As the utilization of frustum culling in registration is not new, it is necessary to point out the novelty of the proposed approach. Luck *et al*. (2000) utilize a frustum-based segmentation technique in order to outline the portions of the scene visible from the range sensor in the currently estimated position, which are next registered with the newly added point cloud with the use of the ICP. Hence, in this case, the frustum (modelled here simply as a set of triangles extending from the laser scanner) is used to accelerate the ICP. An extension of this approach is proposed by May *et al*. (2009), who embed frustum culling in the ICP, by employing the pose estimate of the previous iteration step. As a result, the scene points outside the model frustum are filtered out by testing against clipping planes before the nearest neighbour search, which, analogically, fastens the computations. Hence both of the mentioned approaches utilize frustum culling for the estimation (or refinement) of the current sensor pose.

On the contrary, in the proposed system, frustum culling is employed in a different step of the registration process, i.e., in scene model update. This is related to the fact that the update of surfel map requires transforming (even) millions of surfels into camera coordinate frame and subsequently projecting them onto image plane, which is much slower than the merging of two point clouds. The utilization of octree-based frustum culling for rapid outlining of portions of a map to be updated fastened this step and enabled the system to work in near real-time.

Sparse ICP in visual odometry. The second novelty concerns the utilization of a visual odometry method based on a sparse ICP algorithm (Dryanovski *et al*., 2013) that is more efficient than the one proposed in another

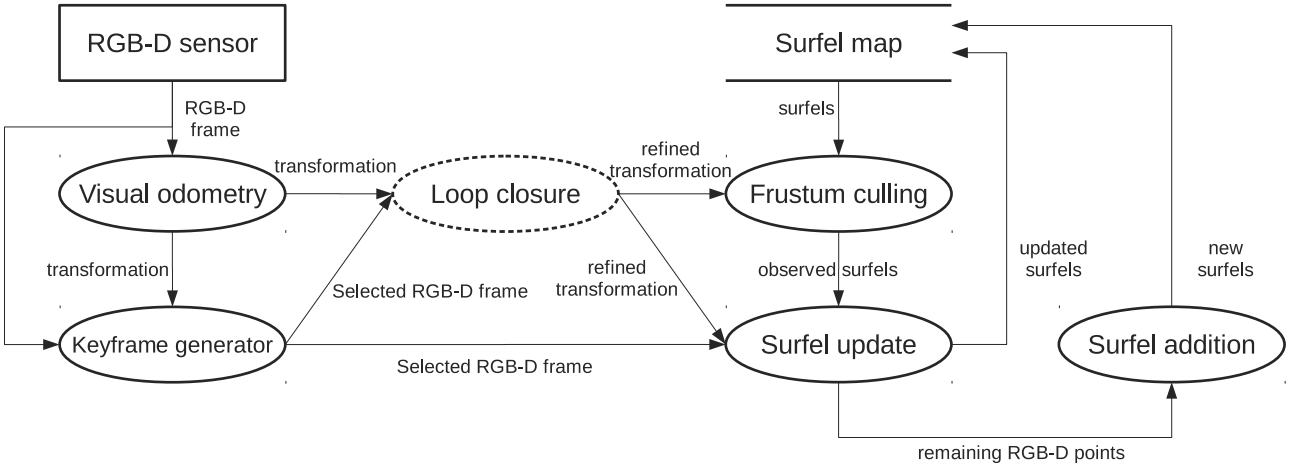


Fig. 8. Data flow in the proposed system.

surfel-based approach (Henry *et al*., 2012). In addition to the sole sparsity, the new scan is matched against multiple previous ones, eliminating the need for a loop closure of small areas. The implementation of this method is able to work at the rate of even 60 fps.

4.3. Visual odometry. The *Visual odometry* module is responsible for computation of the pose increment of the Kinect sensor between consecutive frames. The approach by Dryanovski *et al*. (2013) is followed and the algorithm proposed by Shi and Tomasi (1994) is used for detection of keypoints in the image, whereas the mean and the covariance matrix of the Cartesian positions of each keypoint *constitute* the feature descriptor. The mean and covariances are computed by applying the Gaussian mixture model in the keypoint Cartesian position. The 3D uncertainty used in those computations is derived from the depth uncertainty model of the camera with additional assumptions that not only z, but also x and y coordinates are prone to errors and that there are dependencies between those measurements.

The resulting sparse cloud is right after the passed to the ICP, with the assumption that the initial transformation is equal to the transformation computed for the previous frame. Hence, although it operates on features, there is in fact no feature matching step. Instead, the new sparse cloud is aligned with the one containing the whole model of the scene (instead of the cloud extracted from the last frame), with additional mechanism for forgetting the oldest points. As result, the algorithm is capable of creating quite coherent maps even without the bundle adjustment procedure.

4.4. Keyframe generation. The inputs to the *Keyframe generation* module are the transformation information from the *Visual odometry* module and the RGB-D image. Raw RGB-D frames coming from the sensor usually contain redundant data captured from similar sensor positions, which do not deliver much new information. Hence the main goal of the module is to select the most important RGB-D frames in order to reduce the volume of data processed by subsequent modules. This is achieved by selecting only the frames obtained from sensor positions significantly different from the corresponding previous sensor positions.

The algorithm defines two thresholds: t_{alpha} for an angle between two sensor poses and t_{dist} for distances between two sensor poses. When the incoming sensor pose differ by no more than t_{alpha} regarding the orientation and t_{dist} regarding the the position from the sensor pose associated with the last keyframe, no new keyframe is created. Otherwise, a new keyframe is constructed and submitted to further processing.

4.5. Loop closure. The loop closing procedure is performed off-line and consists of two steps. In the first one it creates a graph, with vertices corresponding to the selected keyframes and edges corresponding to transformations between the overlapping keyframes. In the second step, the sensor poses are globally optimized using the *g2o* framework (Kummerle *et al*., 2011).

In order to identify overlapping keyframes, the procedure extracts SURF features for every pair of keyframes and finds the best matching between their features. Next, the transformation between each two keyframes is established using the RANSAC algorithm. A pair of keyframes is identified as overlapping, when enough inlier correspondences vote for the specific transformation. The transformation is then associated with the newly created edge in the graph.

The *g2o* algorithm uses the created graph as a source of constraints in the optimization problem.

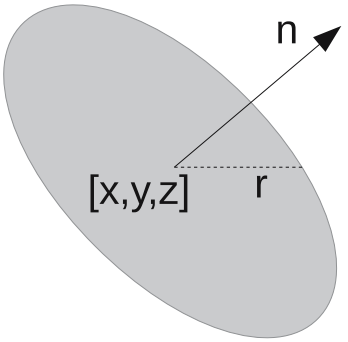


Fig. 9. Illustration of a surfel with its attributes.

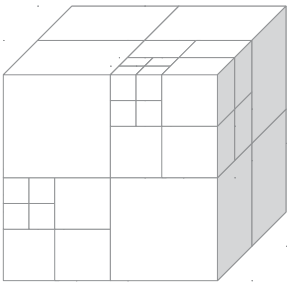


Fig. 10. Octree example.

The optimized parameters are sensor poses associated with the keyframes. For each value of the parameter vector between-keyframe pose, transformations may by computed. These transformations are subsequently compared against the constraint transformations computed from SURF feature matchings. The error function thus obtained forms an optimization criterion for the *g2o* procedure. As an outcome, a vector of optimized sensor poses for keyframes is given.

4.6. Surfel map. The surfel map is composed of elements called surfels. The name *surfel* is an abbreviation for *surface element* and has its origin in computer graphics (Pfister *et al*., 2000). A surfel represents a surface patch (usually circular) in a 3D space, characterised by its position, radius and a normal vector describing its orientation (Fig. 9). Additionally, it may contain other useful attributes, e.g., color, curvature or a measure of confidence. The measure of confidence may be defined as the number of views where the surfel was observed, but also (in a more complex solution) as a histogram of viewing directions. Surfels became useful as building blocks of 3D maps, as proposed by Weise *et al*. (2009) and followed by others, e.g., Krainin *et al*. (2010) or Henry *et al*. (2012), mainly due to point compression and noise reduction capabilities of surfel-based mapping algorithms.

In the proposed approach, every i-th surfel s_i contains the following attributes: position in the Cartesian space $p_i = [X_i, Y_i, Z_i]^T$, radius r_i, orientation $\mathbf{n}_i$, color c_i and confidence v_i, being equal to the number of views in which the surfel was observed.

4.7. Octree storage. The surfel map is stored in an octree, which is a tree data structure used for storing data in three dimensions. Each octree node represents a cube in a 3D space. The cube is divided into 8 octants (by bisecting the space along each of the 3 dimensions), and each of the node children represents one of the octants. Thus, each of the nodes in the tree has an associated bounding box. When the surfel is added to the tree, it either occupies one of the existing leaf nodes, or a new one is created to accommodate the new surfel. An image presenting division of a 3D space by an octree is presented in Fig. 10. After addition of all surfels to the tree, the structure of occupied voxels resembles the general structure of the underlying surfel cloud. The octree is especially useful for performing volume queries with a complexity independent of the existing point configuration. For instance, it enables fast rejection of large subsets of surfels from further deliberations as lying outside of the area of interest.

4.8. Frustum culling. The most computationally expensive operation in the surfel map creation is the surfel update step. During this step, the surfels in the scene need to be reprojected onto the sensor image plane, so their coordinates might be compared with the new scan. In the work of Henry *et al*. (2012), all surfels were transformed into the current camera coordinate frame and projected onto the image plane. This approach is not very efficient, since the number of surfels that need to be transformed and projected may reach several millions. As a result, Henry *et al*. (2012) report that the surfel processing step

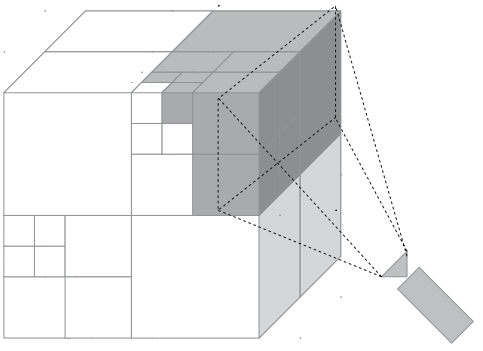


Fig. 11. Octree frustum culling example.

takes on average 3 seconds per frame.

In the proposed approach, frustum culling is applied immediately to the surfels to efficiently prune surfels outside the sensor cone of view, called the *frustum*. The adopted approach is similar to the one used in computer graphics, e.g., in the Point Cloud Library (Rusu and Cousins, 2011) visualization module, with the exception that we develop the real camera model instead of a simplified artificial one. The frustum culling algorithm can be divided into two major steps: the computation of clipping planes equations and using clipping planes to filter out 3D data. For brevity, the theoretical details of the computation of clipping planes for the Kinect camera at a specific position in world coordinates are skipped here and included in Appendix.

In the implementation of the second step, an octree is utilized for efficient organization of the surfel map in a 3D space. When applying a hierarchical search to the octree, one can easily prune certain segments of the space at coarse resolution, thus saving computation time. More precisely, in the case when the whole octree node is outside or inside the frustum, the same applies to all of its children, so they must not be further verified. The idea of combining frustum culling with an octree representation to select the subset of surfels that must be reprojected is presented in Fig. 11.

4.8.1. Frustum culling for a voxel. With a 3D scene organized in voxels, we must be able to decide whether a voxel lies within the frustum or not. The simplest way is to check each vertex of the voxel against frustum clipping planes. However, this can be done more efficiently by approximating the voxel with the smallest circumscribing sphere. Then, we say that the voxel lies within the frustum if the corresponding sphere lies within it. This is achieved by a simple *sphere test*, defined as follows.

Let us describe a sphere S by its center and radius (p_0, r). The sphere coordinates are compared against all frustum clipping planes, and for each plane a decision is made whether the sphere lies within a positive halfspace of the plane, a negative halfspace of the plane, or intersects the plane. Let us assume that each plane P is described by the equation in Hessian normal form:

$$\mathbf{n} \cdot p = -t, \tag{1}$$

where the vector $\mathbf{n}$ normal to the plane is standardized and t is the distance from the plane to the origin of the coordinate system. We also assume that the normal vector points in the direction of the frustum interior, thus dividing the space into positive and negative halfspaces. This enables fast computation of the "signed" distance between the center of the sphere and the plane:

$$d(S, P) = \mathbf{n} \cdot p_0 + t. \tag{2}$$

Now,

- if $d(S, P) > r$, then the sphere lies within the positive half-space of the plane;
- if $-r \leq d(S, P) \leq r$, then the sphere intersects the plane;
- if $d(S, P) < -r$, then the sphere lies in the negative half-space of the plane.

By applying these simple decision rules to all planes constituting the frustum, we may decide whether the sphere (hence the inscribed voxel) lies within, outside or intersects the frustum.

4.8.2. Frustum culling for a cloud organized in an octree. If the cloud is organized in an octree, the frustum culling technique can be performed quite efficiently since most of the pruning affects higher tree levels. The idea of a hierarchical, logarithmic culling procedure was first proposed in the classic work of Clark (1976). In our system, in order to perform the culling, we traverse the surfel octree in a depth-first manner. For each node, the following actions are undertaken based on the sphere circumscribing its bounding box:

- if the node is a leaf node and its bounding box is within the frustum or intersects the frustum, add all its points to the *observed cloud*;
- if the node is a leaf node and its bounding box is outside the frustum, skip all the points associated with the node;
- if the node is a branch node and its bounding box is within the frustum, add all the points to the *observed cloud*;
- if the node is a branch node and its bounding box intersects the frustum, recursively process the children of the node,
- if the node is a branch node and its bounding box is outside the frustum, skip the children of the node.

The procedure above implies that for points from leaf nodes that intersect the frustum one cannot conclude whether they lie inside or outside the frustum, and the decision must be made individually for each point. Therefore, the size of the leaf node determines the efficiency of frustum culling. There emerges a trade-off between the efficiency of the frustum culling method and the octree depth, and the overheads associated with management of large data structures.

4.9. Surfel update. With all the surfels constituting the *observed cloud*, determined they can be updated with the new scan. The map update consists of two steps: surfel update and surfel addition. Before updating

and adding new surfels, normal vectors are computed for the new scan. For this purpose, an effective normal computation method for RGB-D cameras was implemented, as described by Holzer *et al*. (2012).

In the model update procedure, the method proposed by Weise *et al*. (2009) and Krainin *et al*. (2010) is followed quite closely. For each subsequent j-th sensor scan from visual odometry, the j-th sensor pose ${}^{G}_{C_j}T$ with respect to the global reference frame G is obtained. This information is subsequently utilized in transforming the position ${}^{G}p_i$ and normal vector ${}^{G}\mathbf{n}_i$ of every surfel from the current map into the current camera frame, obtaining ${}^{C_j}p_i$ and ${}^{C_j}\mathbf{n}_i$. Thus, assuming the right-handed coordinate system, the surfel depth is simply a z-coordinate of ${}^{C_j}p_i$, i.e., ${}^{C_j}z_i$.

Then, the surfel is projected onto the image plane and a sensor reading associated with the surfel is obtained (for efficiency the "nearest-neighbor interpolation" is used). If, based on the interpolated coordinates, no reading can be associated with the surfel (the surfel projection is out of bounds of the RGB-D image, or there is no valid reading), the surfel is not updated.

Let us denote the depth currently acquired from the sensor as ${}^{C_j}z'_i$. The depth reading is regarded as valid if the depth is within the bounds of a reliable reading $[z_{r,\min}, z_{r,\max}]$ and the z-component of $\mathbf{n}'_i$ is above a certain threshold. If the reading is regarded as valid, then, similarly to Weise *et al*. (2009), different update rules are used depending on the difference ${}^{C_j}z'_i - {}^{C_j}z_i$:

- if $\left|{}^{C_j}z'_i - {}^{C_j}z_i\right| \leq z_{\max}$, then the surfel is updated. The running averages are computed to update the point position, normal direction and color; if the new reading is taken from a closer location, also the surfel radius is updated; with each new reading associated with the surfel, the visibility confidence v_i of the surfel is incremented;

- if ${}^{C_j}z'_i - {}^{C_j}z_i < -z_{\max}$, then the reading is behind the surfel, and therefore it becomes invalid or the surfel is removed, depending on the value of surfel confidence (a static threshold is used here);

- if ${}^{C_j}z'_i - {}^{C_j}z_i > z_{\max}$, then the sensor reading is ahead of the surfel, so the surfel is not updated, and the reading may be subsequently used to form a new surfel.

An illustration of different cases of surfel update is shown in Fig. 12. Here $z_{\max}$ is a constant describing the maximum depth difference between the surfel and the reading still allowing the merge operation. In order to speed up computations, surfels that are outside depth bounds of the reliable reading (increased by $z_{\max}$, i.e., $[z_{r.\min} - z_{\max}, z_{r.\max} + z_{\max}]$) are not updated.

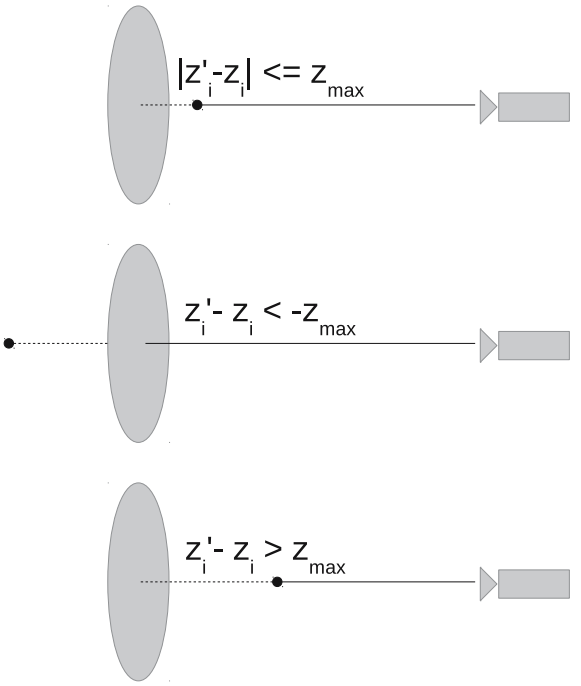


Fig. 12. Different cases of the surfel update procedure. The large disc is the surfel to be updated (the black dot denotes the actual reading).

4.10. Surfel addition. For each sensor reading with a valid depth that was not used for surfel update, a new surfel is created by making a copy of the reading position, normal vector and color $({}^{C_j}p'_k,\ {}^{C_j}\mathbf{n}'_k,\ {}^{C_j}c'_k)$. The projection of the surfel onto the image plane approximately corresponds to the camera pixel size, so the radius must take into account both the distance and normal vector orientation. The radius computation formula is adopted from the work of Weise *et al*. (2009) as

$$r_k = \sqrt{2}\frac{{}^{C_j}z'_k/(\alpha+\beta)}{\mathbf{n}'_k(z)}, \tag{3}$$

where α and β denote the focal lengths of the sensor along its x and y axes.

5. Implementation

The above described mapping system was implemented in the Robot Operating System (ROS) framework (Quigley *et al*., 2009) with the use of the Point Cloud Library routines (Rusu and Cousins, 2011). The system is constructed from 3 main ROS nodes: the *visual_odometry* node, the *keyframe_mapper* node, and the *surfel_mapper* node. The data flow between nodes (presented in the form of ROS topics) is given in Fig. 13. The characteristic of each node is given below.

- *visual_odometry* node—the main node responsible

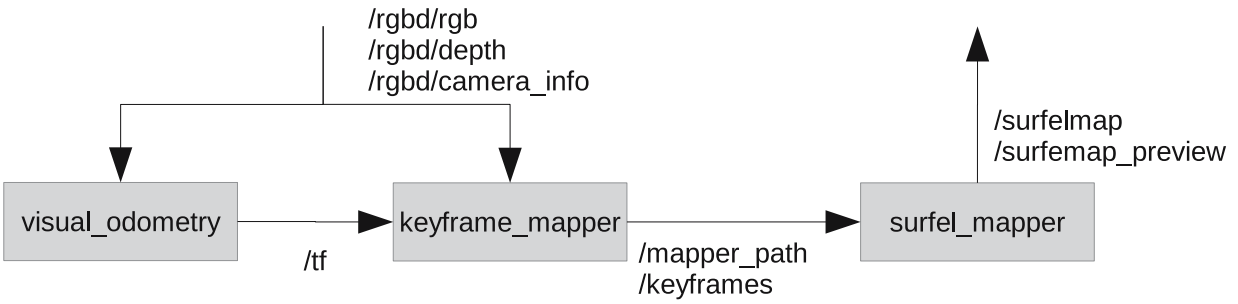


Fig. 13. Implementation-specific system architecture.

for establishing sensor position based on sensor readings. The node subscribes a sequence of RGB and depth images from the sensor and publishes the odometry information (current sensor pose with respect to the first sensor pose):

  - /rgbd/rgb—rectified image from RGB camera,
  - /rgbd/depth—rectified image from Depth camera, the image is registered in the RGB coordinate frame,
  - /camera_info—RGB (and Depth) the camera information regarding camera reference frame and camera parameters,
  - /tf—estimated transformation between a fixed coordinate frame and the current sensor coordinate frame:

- *keyframe_mapper* node—an auxiliary node used for keyframe generation and sensor path optimization. The node takes as an input visual odometry messages and RGB-D data, and outputs registered keyframes generated for sufficiently distinct sensor positions. It also performs an off-line loop closing and provides a new set of registered keyframes together with a new set of positions as /mapper_path data;

  - /mapper_path—an ROS topic with path messages containing sensor positions, which positions either mirror positions provided by *visual_odometry* (at start) or represent a corrected set of positions (after loop closure);
  - /keyframes—keyframe point clouds registered in the fixed odometry frame. Registration uses sensor positions identical to /mapper_path;

- *surfel_mapper* node—the main node used for surfel map creation. Its input: registered point clouds keyframes and estimated sensor positions /mapper_path. Its output: snapshots of the constructed surfel map;

  - /surfelmap—surfel information from the bounding box specified, published in the form of ROS markers,
  - /surfelmap_preview—fast low resolution preview of the full constructed surfel map (for each octree leaf only a single point is generated (see Fig. 14)).

The ROS implementation of the system described in this paper is available at `github.com/piappl/robrex_mapping.git`. For the *visual_odometry* and *keyframe_mapper* nodes, there was used an implementation provided at `github.com/ccny-ros-pkg/ccny_rgbd_tools`.

6. Experiments

6.1. Experimental setup. The experiments were performed on sequences of synchronized RGB and Depth images. The first set of three sequences was recorded using the Kinect camera in PIAP laboratories. The three sequences used are: the "PIAP Office Room" (PIAP OR for short) sequence, the "PIAP Room and Corridor" (PIAP RC) sequence (containing the same room with the corridor fragment and the stairway) and the "PIAP Hangar" (PIAP H) sequence (representing a walk around a 17 $\times$ 8 m robot hangar). The sequences here are given in the order of the increasing size of the mapped area. Additionally, for demonstration purposes, some preliminary experiments were also performed on a shorter "PIAP Desk" sequence.

In addition to our own sequences, we wanted to experimentally verify our system on some publicly available datasets, which would enable other researchers to compare their results with ours. We decided to use the dataset proposed by Handa *et al*. (2014) (Imperial College London, ICL), containing two artificially generated scenes, being dedicated in principle for the purpose of benchmarking visual SLAM. The dataset is partitioned into two subsets: "Living Room" (ICL LR for short)

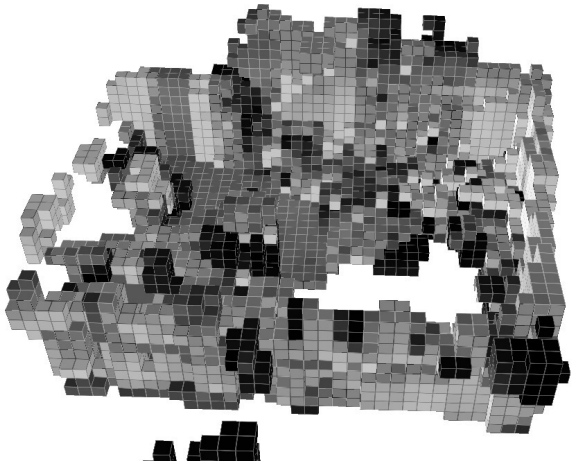


Fig. 14. Preview of the "room" sequence. Leaf nodes of the surfel map octree are diplayed.

and "Office Room" (ICL OR), each containing four sequences. Each sequence comes in two flavors—the pure rendered one and that with noise added.

During the experiments, the *visual_odometry* module was used to provide visual odometry information and the *keyframe_mapper* module generated a sequence of keyframes. The keyframes were subsequently used by the *surfel_mapper* module for surfel map creation. The octree preview of the "PIAP OR" sequence is given in Fig. 14.

In a typical scenario, all three modules, as well as the data acquisition process, work simultaneously. For our PIAP test sequences, we were able to process incoming data in real time. The *surfel_mapper* module, the *visual_odometry* module and the *keyframe_mapper* module were all run in separate threads.

6.2. Visual odometry performance. Although the evaluation of the third-party *fast visual odometry* algorithm is not the main topic of this paper, some crude quality assessment turned out to be necessary in order to ensure reliability of visual odometry data for subsequent benchmarking of our surfel representation. Each sequence was used for generating the map with explicit loop closing turned on or off. The results were visually inspected and are given in Table 1. In addition, for sequences with ground truth trajectories available we computed absolute trajectory errors (ATEs) and relative pose errors (RPEs). These results are given in Table 4. Visualizations of trajectories together with ground truth and residuals are given in Fig. 15. The analysis of quantitative results confirms the visual inspection of map quality.

During the experiments it was established that *fast visual odometry and mapping* is prone to failures in the case of mapping large surfaces with a small number of features (such as large walls or empty corridors). This is basically

Table 1. Visual evaluation of VO quality (without and with loop closing turned on) for the test sequences. The following notation is used: "–" (scene geometry is visually wrong), "+/–" (scene geometry is generally fine, but some artifacts can be easily spotted, e.g., "doubled" surfaces), "+" (scene geometry is fine, there are no easily perceivable artifacts).

	no loop	loop
"PIAP OR"	+	+
"PIAP RC"	+/–	+/–
"PIAP H"	–	+/–
"ICL LR 0"	+	+
"ICL LR 0 (with noise)"	+/–	+
"ICL LR 1"	+/–	+
"ICL LR 1 (with noise)"	+/–	+
"ICL LR 2"	+/–	+
"ICL LR 2 (with noise)"	–	–
"ICL LR 3"	–	–
"ICL LR 3 (with noise)"	–	–
"ICL OR 0"	+/–	+/–
"ICL OR 0 (with noise)"	+/–	+
"ICL OR 1"	+/–	–
"ICL OR 1 (with noise)"	+/–	+
"ICL OR 2"	+	+
"ICL OR 2 (with noise)"	+/–	+
"ICL OR 3"	+	+
"ICL OR 3 (with noise)"	+/–	+

a result of the fact that FVOM performance is based on utilization of sparse features. This applied both to basic mapping (using ICP on a sparse feature map) and mapping with loop closing (which matches features globally).

For larger sequences (e.g., "PIAP H"), the basic ICP mapping fails to properly close the sequence loop, which is to a large extent amended by application of the explicit loop closing feature. What is more, the utilization of the off-line loop closing increased mapping accuracy for all sequences, which is reflected, e.g., in a smaller (by a few percent) number of surfels constituting the resulting scene (due to relatively more surfel updates).

For the "ICL LR" test sequences it was possible to evaluate the odometry quality by comparing the resulting map with the ground truth model of the room. Such comparison was performed on two sequences ("ICL LR 0 (with noise)" and "ICL LR 1 (with noise)"; cf. Table 2). It must be noted that the mean distance between the ground truth model and the map estimated using visual odometry is very close to the map obtained based on the ground truth trajectory. For ground truth trajectories the error is between 6 and 9 mm, whereas for visual odometry the error is between 9 or 11 mm, which we found acceptable. The results suggest also some noise reduction capabilities of surfel update over simple point summation, as the map constructed using surfel update gives 1–2 mm smaller average distance-to-ground-truth than the reconstruction

Table 4. Generated trajectories compared to ground truth trajectories. The RMSE of the absolute trajectory error (ATE) and the relative pose error (RPE) (rotational (t-RPE) and translational components (r-RPE)) are given. Procedures proposed by Handa *et al*. (2014) were used for benchmarking.

	no loop RMSE			loop RMSE		
	ATE [m]	t-RPE [m]	r-RPE [deg]	ATE [m]	t-RPE [m]	r-RPE [deg]
ICL LR 0	0.025	0.039	2.765	0.012	0.021	1.062
ICL LR 0 (with noise)	0.036	0.066	2.931	0.018	0.030	1.487
ICL LR 1	0.103	0.147	4.924	0.011	0.022	0.647
ICL LR 1 (with noise)	0.143	0.204	5.968	0.013	0.020	0.752
ICL LR 2	0.068	0.105	3.983	0.013	0.023	0.761
ICL LR 2 (with noise)	1.063	1.770	86.633	1.057	1.762	86.627
ICL LR 3	0.413	0.610	11.179	0.080	0.155	5.405
ICL LR 3 (with noise)	0.498	0.727	8.926	0.134	0.245	5.434
ICL OR 0	0.025	0.040	1.855	0.022	0.033	0.971
ICL OR 0 (with noise)	0.068	0.120	7.475	0.027	0.043	1.355
ICL OR 1	0.041	0.070	1.823	0.306	1.017	74.496
ICL OR 1 (with noise)	0.061	0.106	2.691	0.182	0.368	8.868
ICL OR 2	0.021	0.031	1.074	0.012	0.019	0.397
ICL OR 2 (with noise)	0.098	0.152	3.535	0.022	0.036	1.059
ICL OR 3	0.028	0.048	0.760	0.011	0.023	0.889
ICL OR 3 (with noise)	0.059	0.092	2.527	0.038	0.059	1.935

Table 2. Comparison of the map with a ground truth 3D model. The result is presented as the mean distance between map points and the ground truth cloud. The comparison was made using real (ground truth) trajectories (denoted as GT) as well as visual odometry trajectories (denoted as VO). Map types comprise points-based maps and surfel-based maps. Procedures proposed by Handa *et al*. (2014) with modifications given at `www.doc.ic.ac.uk/~ahanda/VaFRIC/living_room.html` (accessed July 2015) were used for benchmarking.

	traj.	type	mean dist. [m]
"ICL LR 0 (with noise)"	GT	points	0.008
	GT	surfels	0.006
	VO	surfels	0.009
"ICL LR 1 (with noise)"	GT	points	0.010
	GT	surfels	0.009
	VO	surfels	0.011

Table 3. Basic data describing benchmarking sequences. "keyframes" is the number of keyframes generated by the *keyframe_mapper* module, "points" is the total number of scans considered usable for map creation.

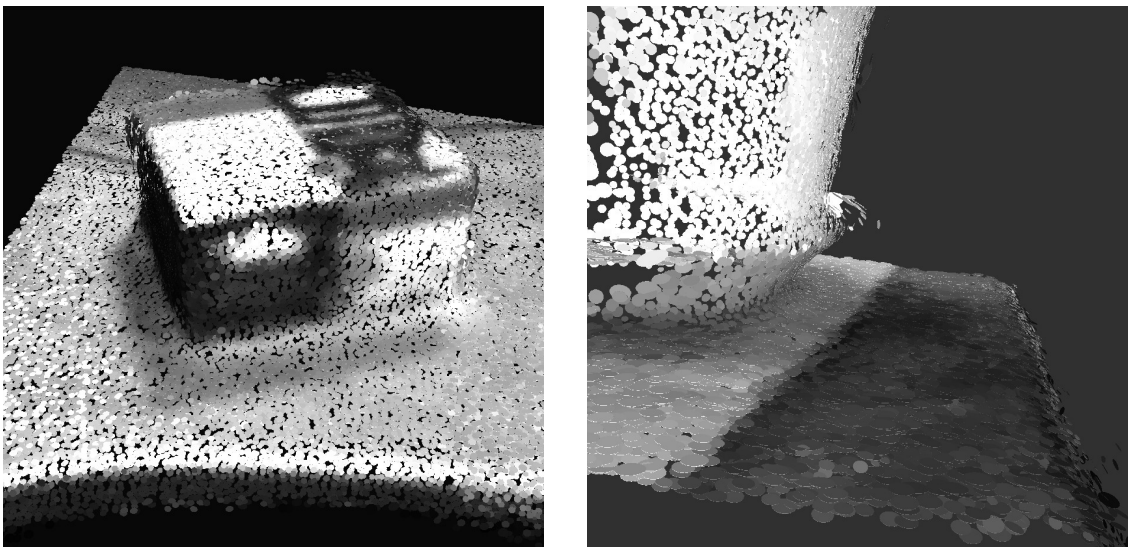
	keyframes	points
"ICL OR 2"	86	14 963 707
"ICL OR 2 (with noise)"	84	13 392 091
"ICL LR 2"	81	13 751 353
"PIAP OR"	171	26 613 260
"PIAP RC"	355	39 271 937
"PIAP H"	367	37 844 437

utilizing simple point summation.

Eventually, for detailed benchmarking of the surfel map and frustum culling there were selected 6 sequences: "PIAP OR", "PIAP RC", "PIAP H", "ICL OR 2", "ICL OR 2 (with noise)", "ICL LR 2". From among the synthetic sequences, those covering the largest portion of the "virtual room" were selected. A brief characteristic of the sequences used in benchmarks is given in Table 3.

6.3. Surfel data compression. One of the most important features of a surfel map representation is compression of incoming point clouds with as low information loss as possible. Some point compression was provided by the utilization of keyframes, captured only when the sensor position significantly changed. However, even so, each surface in the map was captured from several sensor positions and thousands of data merging procedures were applied for each frame.

During the experiments it was observed that the system performs well concerning compression of incoming points, and redundant surfels appear very rarely. In Fig. 15 we may see small details of the mapped scenes. In both the figures we observe small cracks between surfels and the number of overlapping surfels is relatively small. A larger overlap can be observed on surfaces that are almost parallel to the sensor optical axis (e.g., a table in Fig. 15(a)). This is mainly due to the fact that the projection of the (circular) pixels on the surface under such circumstances results in elongated ellipses, which must be approximated by circles.



(a) (b)

Fig. 15. Selected details of surfel maps for the sequences "PIAP Desk" and "PIAP OR": teabox on the desktop (a), flower pot (b).

In our experiments we set the scan merging distance to quite a large value of $d_{\max} = 5$ cm. We believe that this value is appropriate for mapping larger spaces such as rooms, since it enables surfel-scan merging even for imperfectly aligned scans, especially at a larger distance from the sensor. However, such a value may lead to merging finer details of objects into more prominent surfaces. Therefore, for individual objects, smaller values are recommended.

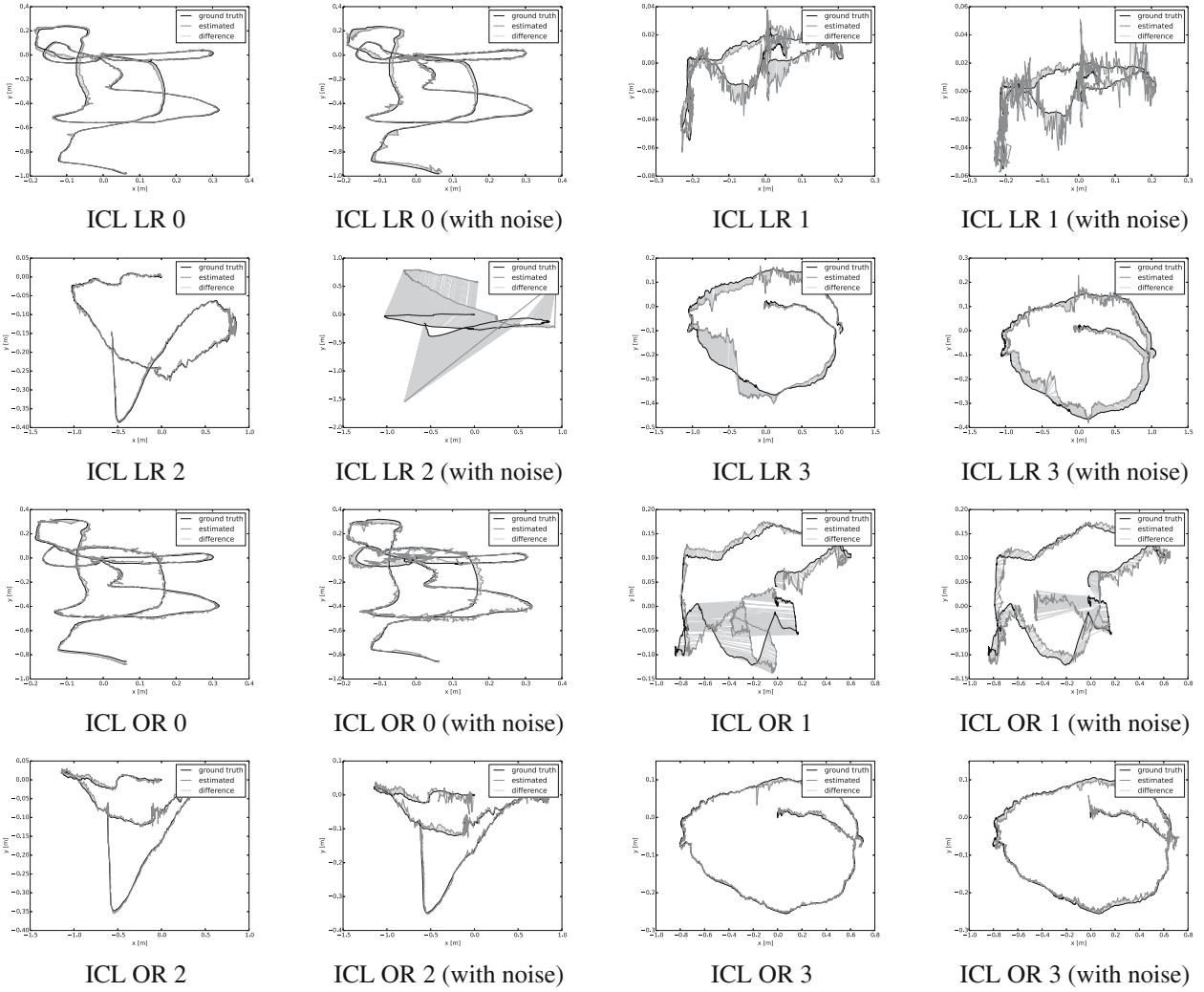
Our intuitive notion of good compression capabilities of the surfel representation are also supported by quantitative experiments performed on the benchmark sequences. In Fig. 17 we compare a scenario where all the readings from keyframes become part of the map with the scenario of using our surfel mapping approach. The results are the numbers of points/surfels in the map as a function of the frame number. As may be seen by the end of the sequence the number of surfels is from 4.5 to 6.5 times smaller than that of cloud points in the simple method, depending on the sequence. We will later show that such discrepancy may have a serious impact on the complexity of subsequent processing as well as the quality of results.

The level of participation of scans in the surfel update procedure varies between sequences and also inside sequences. In our benchmarks, the average level of participation was about 84% for artificial ICL sequences without noise and between 70–80% for real sequences and artificial sequences with noise added. In most cases, the level of participation was above 60% with occasional valleys.

6.4. Frustum culling. We performed several benchmarks to see if and to what extent the frustum culling method improves the surfel update step.

The first experimental results, presented in Fig. 18, concern the number of surfels that need to be transformed into the camera frame in order to perform reprojection onto the image plane at a new sensor position. In the case of the update without utilization of frustum culling, the number of surfels grows steadily with each subsequent input frame (as it depends on the total number of surfels in the scene). On the contrary, the number of surfels transformed using the culling method is generally much smaller and depends on the complexity of the actual scene fragment observed by the sensor, rather than the actual frame index. More precisely, peaks in the chart correspond to exploration of already known areas of the map, whereas valleys correspond to exploration of new areas. The ratio between necessary surfel transformations with and without frustum culling is the lowest for sequences in small confined spaces (e.g., the "ICL OR" sequences that produce only slightly above 2 mln surfels), and the largest when mapping larger areas with only seldom "remapping" of already known space ("PIAP RC" and the "PIAP H" sequence, which produce 8–9 mln surfels). For instance, for the "PIAP H" sequence the average ratio is over 18.7, while for "ICL OR 2" it is only 3.9.

The time of single surfel processing varies. It depends mainly on how many processing steps are applied to each surfel. The surfel processing chain may consist of surfel rigid transformation, surfel projection onto the image plane and surfel update. Therefore, in addition to the number of surfels that are taken into account during processing, it is worth comparing the actual processing time of each keyframe when using, or not, the frustum culling method. In Fig. 19 we present a comparison



ICL LR 0 | ICL LR 0 (with noise) | ICL LR 1 | ICL LR 1 (with noise)

ICL LR 2 | ICL LR 2 (with noise) | ICL LR 3 | ICL LR 3 (with noise)

ICL OR 0 | ICL OR 0 (with noise) | ICL OR 1 | ICL OR 1 (with noise)

ICL OR 2 | ICL OR 2 (with noise) | ICL OR 3 | ICL OR 3 (with noise)

Fig. 16. Comparison of estimated trajectories (gray) with ground truth trajectories (black). The difference is marked in light-gray. In all cases, loop closing was explicitly utilized. Procedures proposed by Handa *et al*. (2014) were used for benchmarking.

of the time spent on the update and addition step for each subsequent frame of benchmark sequences, while in Fig. 20 we compare the total time of frame processing for a surfel mapper with frustum culling turned on and off.

Those figures show that the surfel update method using frustum culling offers superior performance over the method that does not use it. The difference, however, is not as prominent as in the case of comparing pure numbers of processed surfels. The main reason is that most surfels can be quickly rejected shortly after transformation using z-range check and the processing of "valid" surfels consumes a significant part of the resources for both culling and no-culling methods. However, for larger scenes, especially when new areas are scanned (so only a limited number of surfels actually participate in the update), the surfel transformation overhead must eventually dominate. This is confirmed by analyzing the processing times for different sequences. It can be noticed that for longer sequences (such as "PIAP H" or "PIAP RC") the frame processing time can be even considered constant (disregarding fluctations) when applying frustum, while the processing time in the case of no-frustum culling is a constantly growing function (again disregarding fluctuations).

Taking as an example the "PIAP H" sequence, in the case of frustum culling-based update the average surfel update and addition time for the whole sequence was about 24 ms and the total frame processing time was 58 ms (with 27 ms spent on normals' computation). For no-frustum culling method the processing times were equal to 63 ms and 96 ms, respectively (with comparable time for normals' computation). Thus, the speedup gained due to using frustum culling was 2.62 regarding only surfel update and addition and 1.65 regarding total frame process time.

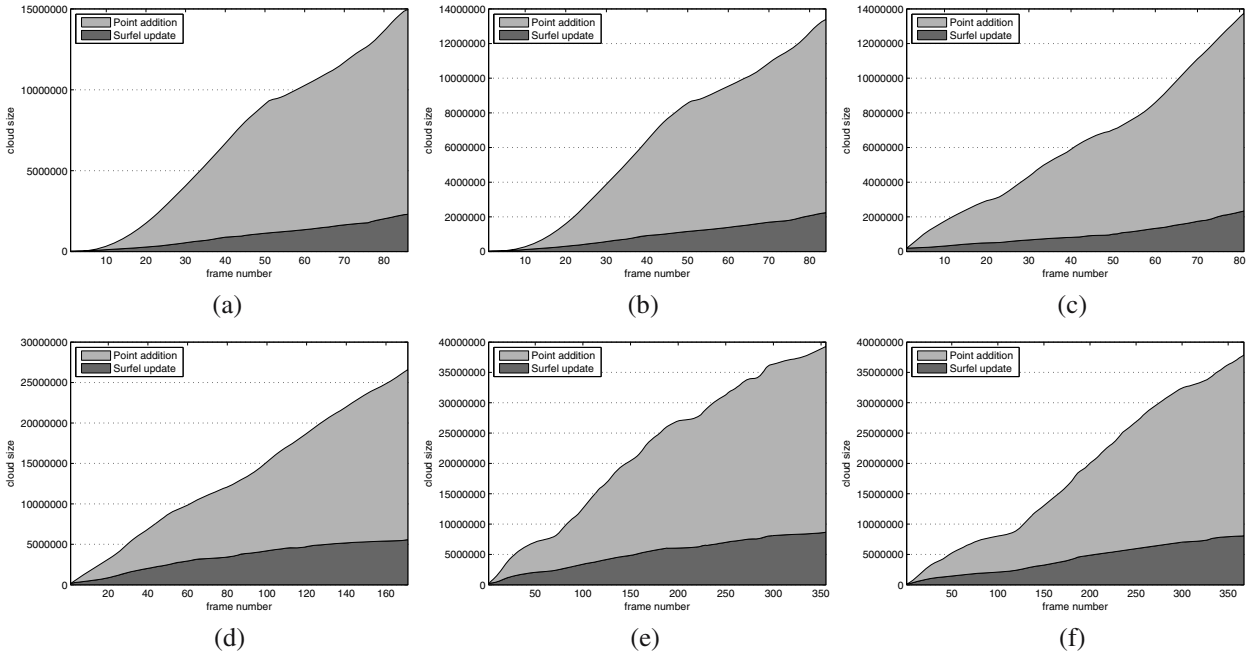


Fig. 17. Comparison of cloud sizes when using the simple point addition model and the surfel update model. Sequences "ICL OR 2" (a), "ICL OR 2 (with noise)"(b), "ICL LR 2" (c), "PIAP OR" (d), "PIAP RC" (e), "PIAP H" (f).

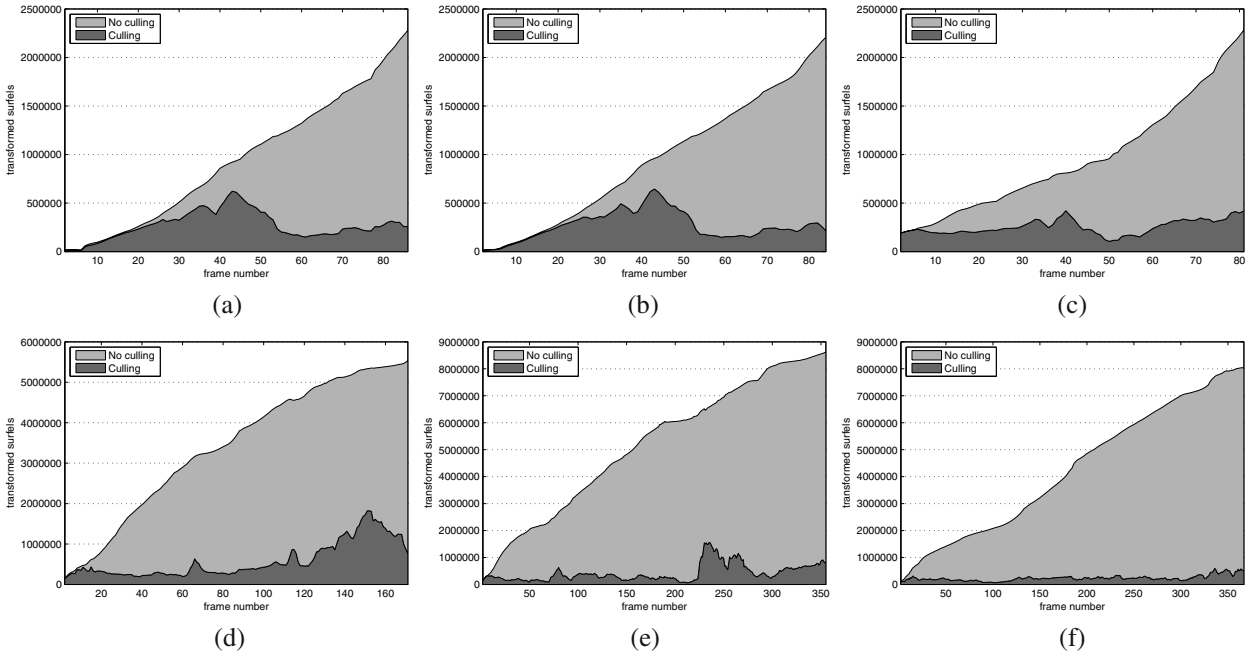


Fig. 18. Comparison of the number of surfels that need to be transformed into a sensor frame (and possibly projected) with frustum culling and without it. Sequences "ICL OR 2" (a), "ICL OR 2 (with noise)" (b), "ICL LR 2" (c), "PIAP OR" (d), "PIAP RC" (e), "PIAP H" (f).

6.5. Frustum efficiency. In order to apply the frustum culling technique, we used an octree with a leaf size of 20 cm. To evaluate the impact of this parameter on our test sequence "PIAP OR" we collected information on the current number of surfels in the scene, the number of surfels pruned using frustum culling and the number of surfels within the frustum, which gave valid projections onto the image plane. As a result, we extracted two

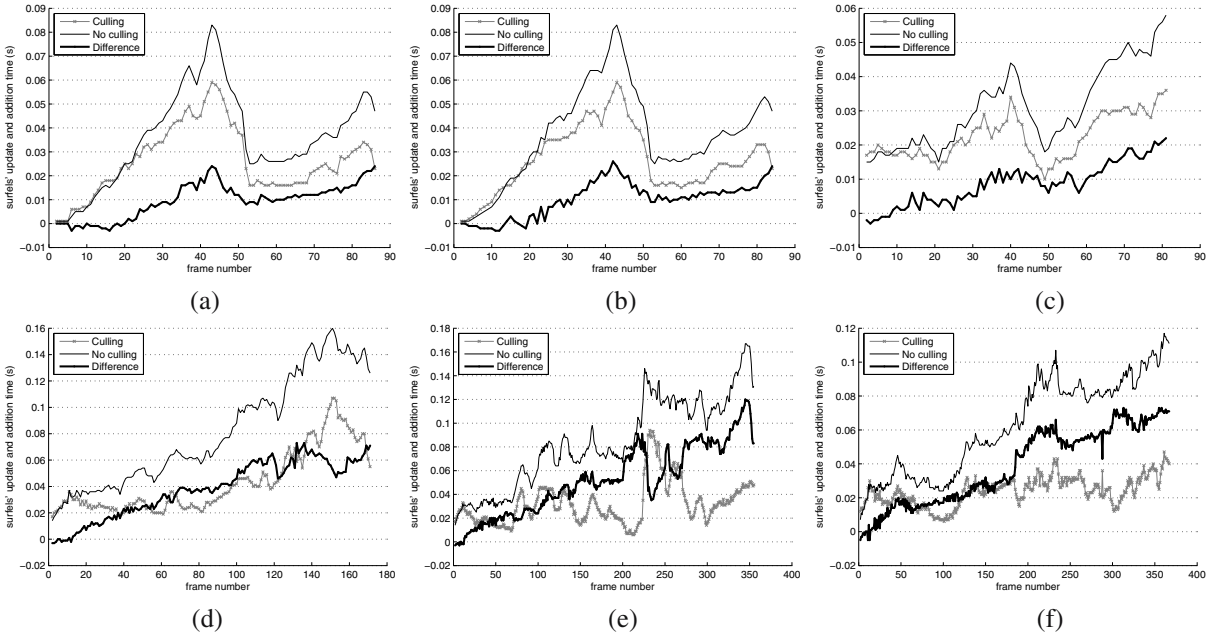


Fig. 19. Surfel processing times (comprising surfel update and addition steps). Sequences "ICL OR 2" (a), "ICL OR 2 (with noise)" (b), "ICL LR 2" (c), "PIAP OR" (d), "PIAP RC" (e), "PIAP H" (f).

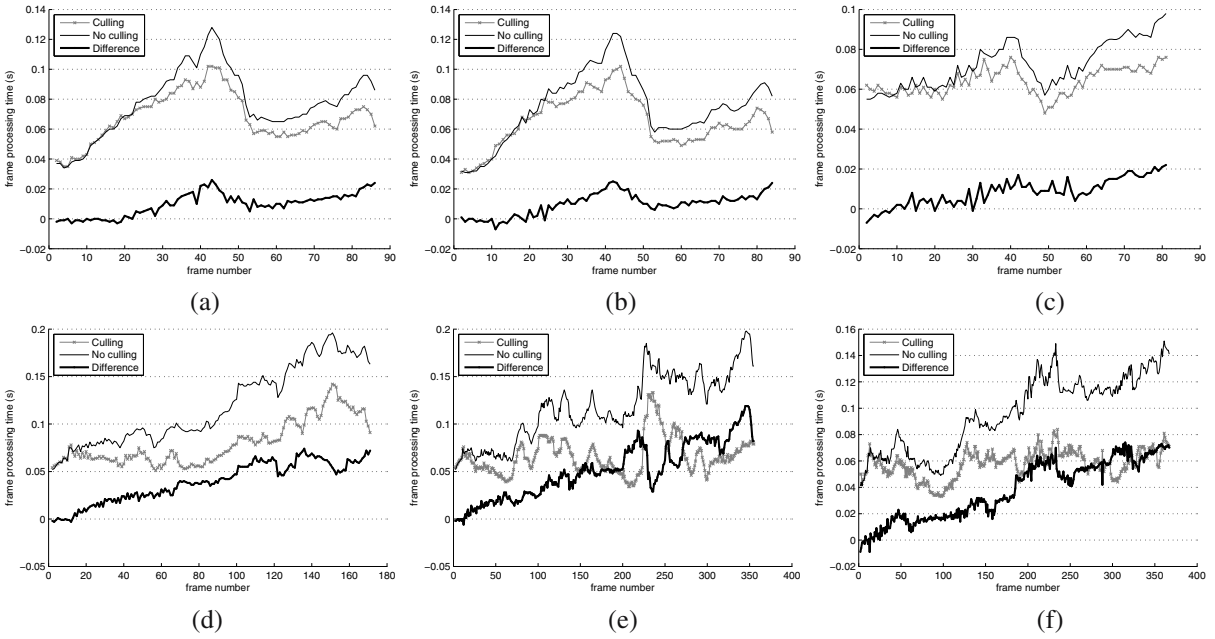


Fig. 20. Total frame processing times. Sequences "ICL OR 2" (a), "ICL OR 2 (with noise)" (b), "ICL LR 2" (c), "PIAP OR" (d), "PIAP RC" (e), "PIAP H" (f).

measures: *culling efficiency* (surfels pruned vs. all surfels in the scene) and *frustum precision* (surfels correctly projected on the image plane vs. surfels in the frustum). The results are given in Fig. 21.

Based on the results we may say the *frustum precision* measure rarely falls below 70% for all sequences. For the "PIAP OR" sequence, the average was 83% for the whole sequence, which means that only 17% surfels had to be unnecessarily transformed into the camera frame. The figures are similar for the remaining sequences (81–88%). This means that there is little justification for further decreasing the octree leaf size, since the expected benefits might be insignificant compared with possible overheads related to management of a larger octree structure.

While *frustum precision* is not dependent on the

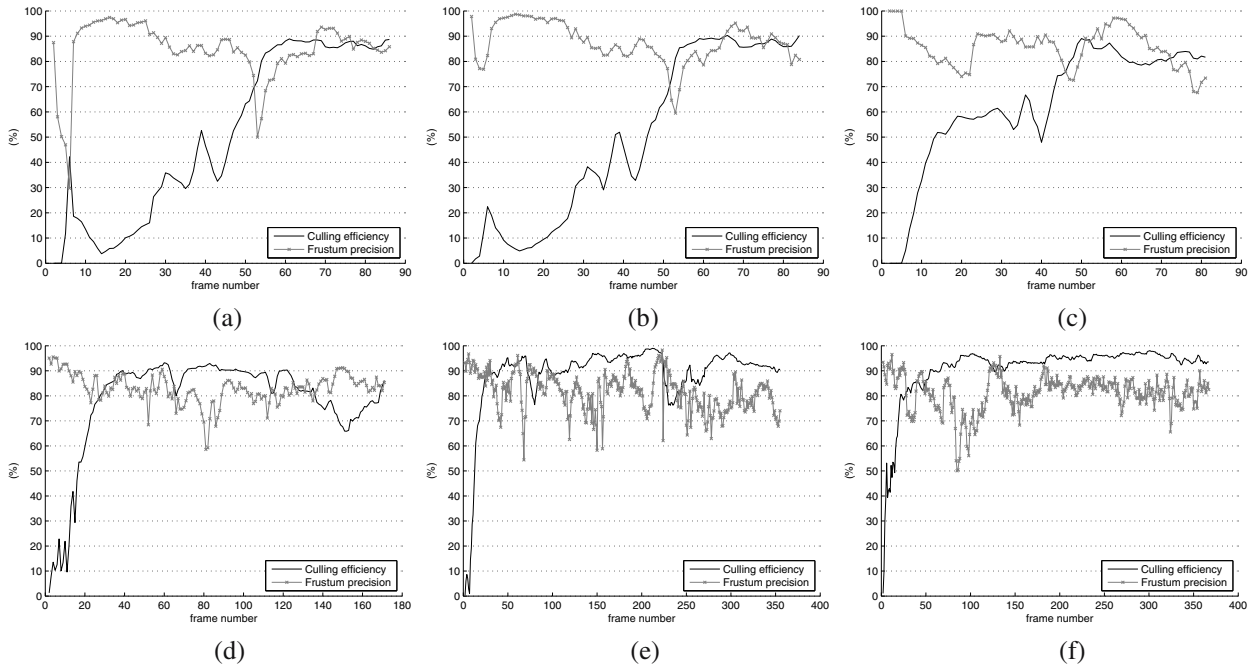


Fig. 21. Evaluation of octree frustum efficiency for the octree leaf size equal to 20 cm. *Culling efficiency* is computed as a ratio of the number of surfels pruned to all surfels in the scene. *Frustum precision* is computed as a ratio of the surfels correctly projected on image plane to all the surfels in the frustum. Sequences "ICL OR 2" (a), "ICL OR 2 (with noise)" (b), "ICL LR 2" (c), "PIAP OR" (d), "PIAP RC" (e), "PIAP H" (f).

actual keyframe index, the *culling efficiency* value quickly grows when new keyframes are acquired. Taking the "PIAP OR" sequence as an example, *culling efficiency* reaches the level of about 90% and stabilizes when we start to reinspect areas already mapped. It is interesting to note that around the 150th frame the value suddenly drops, when the sensor reaches an area with view densely packed with objects. This is accompanied by a peak in frame processing time charts (see Fig. 20).

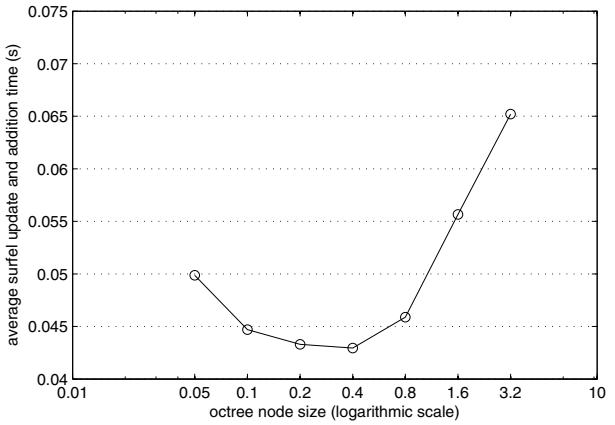


Fig. 22. Octree leaf node size vs. time spent on the surfel update and surfel addition steps for the "PIAP OR" sequence.

For the "PIAP OR" sequence the average value of *culling efficiency* is about 78%, whereas the average for the last 130 frames is 85%. The value is even higher for large mapping areas (91% average for the whole "PIAP H" sequence), but lower for small mapping areas (51% for "ICL OR 2" sequence).

The *cullling efficiency* and *frustum precision* measures may not reflect well the actual performance gain obtained by using frustum culling, given the specific octree leaf size. For instance, using a fine-grained octree will result in very high *frustum precision* scores and also higher *culling efficiency*.

However, the overall impact on performance may be negative due to the overhead associated with management of a large octreeindex. Therefore, for the "PIAP OR" sequence we conducted a performance experiment, measuring the average time of surfel update and surfel addition steps for different sizes of the octree leaf node. The results are presented in Fig. 22. Note that the best performance was obtained for an octree leaf size varying between 20 and 40 cm. The former value was consistently applied in other experiments.

7. Conclusions

Typical approaches to surfel map creation (e.g., Henry *et al.*, 2012) suffer from efficiency problems, which question their applicability to robotic tasks, where on-line processing is important. Two sources of these problems were identified. The first one is the typical use of an

incremental, dense point cloud registration algorithm (for the purpose of visual odometry) based on the ICP method, which is highly resource consuming. The second reason is the surfel update procedure, which is slow for sufficiently large data clouds.

In the proposed system, both of above problems are addressed and resolved. The incremental ICP is replaced by a sparse ICP algorithm, as proposed by Dryanovski *et al*. (2013). The algorithm is more efficient that its dense version and is able to work at the rate of even 60 fps. What is more important, it aligns the new scan against multiple previous ones, eliminating the need for a loop closure of small areas.

Secondly, a frustum culling procedure is applied in order to eliminate as many surfels as possible from the computationally expensive surfel update step. The surfel map is indexed by an octree and the culling procedure exploits the hierarchical map structure.

Taking into account the proposed improvements, a mapping system was constructed and implemented under the popular Robot Operating System (ROS). Finally, an experimental validation of this system was performed. The experiments carried out on RGB-D image sequences obtained by the Kinect sensor showed superior efficiency performance of the proposed surfel-level mapping over simple point cloud summation.

The surfel update step using the proposed culling method performs on average over two times faster than a step without it. More likely, this ratio will grow when larger scenes are processed. For the test sequences, the frame processing time rarely exceeded 100 ms, which allows on-line processing, since the acquisition rate of key frames is usually lower than 0.1 fps.

For those reasons the proposed surfel map representation and processing methods constitute very promising tools for a 3D occupancy map representation in V-SLAM systems based on modern, high-framerate sensors such as Microsoft Kinect or ASUS XTion. Their most important advantages are strong data compression and noise reduction abilities, accompanied by a minimum data loss.

The main area for further improvement targets the problem of global optimization. So far the approach has included a loop-closure method, supplementing the FVOM module in an off-line manner (graph generation in the case of longer test sequences takes several minutes). Recently, new approaches emerged that enable fast, on-line loop closure (Kawewong *et al*., 2013). It would be beneficial to incorporate such an approach in the mapping system, to obtain coherent large-scale maps in real-time. Another interesting area of improvements is the utilization of higher-level knowledge concerning surface smoothing and noise reduction of the acquired surfel map.

Acknowledgment

The authors gratefully acknowledge the financial support of the National Centre for Research and Development (Poland), grant no. PBS1/A3/8/2012. The initial version of this work was presented at the special session on *Robotic perception employing RGB-D images* during the 13th National Conference on *Robotics* in Kudowa Zdrój, Poland, 2014.

References

Alahi, A., Ortiz, R. and Vandergheynst, P. (2012). FREAK: Fast retina keypoint, *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Providence, RI, USA*, pp. 510–517.

Calonder, M., Lepetit, V., Strecha, C. and Fua, P. (2010). BRIEF: Binary robust independent elementary features, *Computer VisionECCV 2010, Heraklion, Greece*, pp. 778–792.

Censi, A. (2008). An ICP variant using a point-to-line metric, *IEEE International Conference on Robotics and Automation, ICRA 2008, Pasadena, CA, USA*, pp. 19–25.

Clark, J.H. (1976). Hierarchical geometric models for visible surface algorithms, *Communications of the ACM* **19**(10): 547–554.

Dryanovski, I., Valenti, R. and Xiao, J. (2013). Fast visual odometry and mapping from RGB-D data, *2013 IEEE International Conference on Robotics and Automation (ICRA), Karlsruhe, Germany*, pp. 2305–2310.

Endres, F., Hess, J., Engelhard, N., Sturm, J., Cremers, D. and Burgard, W. (2012). An evaluation of the RGB-D SLAM system, *2012 IEEE International Conference on Robotics and Automation (ICRA), St Paul, MN, USA*, pp. 1691–1696.

Figat, J., Kornuta, T. and Kasprzak, W. (2014). Performance evaluation of binary descriptors of local features, *in* L.J. Chmielewski *et al.* (Eds.), *Proceedings of the International Conference on Computer Vision and Graphics*, Lecture Notes in Computer Science, Vol. 8671, Springer, Berlin/Heidelberg, pp. 187–194.

Fischler, M.A. and Bolles, R.C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography, *Communications of the ACM* **24**(6): 381–395.

Frome, A., Huber, D., Kolluri, R., Bülow, T. and Malik, J. (2004). Recognizing objects in range data using regional point descriptors, *Computer Vision (ECCV 2004), Prague, Czech Republic*, pp. 224–237.

Gribb, G. and Hartmann, K. (2001). Fast extraction of viewing frustum planes from the world-view-projection, `http://gamedevs.org`.

Handa, A., Whelan, T., McDonald, J. and Davison, A.J. (2014). A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM, *2014 IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, China*, pp. 1524–1531.

Henry, P., Krainin, M., Herbst, E., Ren, X. and Fox, D. (2012). RGB-D mapping: Using kinect-style depth cameras for dense 3D modeling of indoor environments, *International Journal of Robotic Research* **31**(5): 647–663.

Henry, P., Krainin, M., Herbst, E., Ren, X. and Fox, D. (2014). RGBD mapping: Using depth cameras for dense 3D modeling of indoor environments, *in* O. Khatib *et al.* (Eds.), *Experimental Robotics*, Springer, Berlin/Heidelberg, pp. 477–491.

Holzer, S., Rusu, R., Dixon, M., Gedikli, S. and Navab, N. (2012). Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images, *Intelligent Robots and Systems (IROS), Vilamoura-Algarve, Portugal*, pp. 2684–2689.

Hornung, A., Wurm, K.M., Bennewitz, M., Stachniss, C. and Burgard, W. (2013). OctoMap: An efficient probabilistic 3D mapping framework based on octrees, *Autonomous Robots* **34**(3): 189–206.

Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A. and Fitzgibbon, A. (2011). KinectFusion: Real-time 3D reconstruction and interaction using a moving depth camera, *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology, UIST'11, Santa Barbara, CA, USA*, pp. 559–568.

Kasprzak, W. (2010). Integration of different computational models in a computer vision framework, *2010 International Conference on Computer Information Systems and Industrial Management Applications (CISIM), Cracow, Poland*, pp. 13–18.

Kasprzak, W., Pietruch, R., Bojar, K., Wilkowski, A. and Kornuta, T. (2015). Integrating data- and model-driven analysis of RGB-D images, *in* D. Filev *et al.* (Eds.), *Proceedings of the 7th IEEE International Conference Intelligent Systems IS'2014*, Advances in Intelligent Systems and Computing, Vol. 323, Springer, Berlin/Heidelberg, pp. 605–616.

Kawewong, A., Tongprasit, N. and Hasegawa, O. (2013). A speeded-up online incremental vision-based loop-closure detection for long-term SLAM, *Advanced Robotics* **27**(17): 1325–1336.

Konolige, K. (2010). Sparse bundle adjustment, *Proceedings of the British Machine Vision Conference, Aberystwyth, UK*, pp. 102.1–102.11, DOI: 10.5244/C.24.102.

Konolige, K., Agrawal, M. and Sola, J. (2011). Large-scale visual odometry for rough terrain, *in* M. Kaneko and Y. Nakamura (Eds.), *Robotics Research*, Springer, Berlin/Heidelberg, pp. 201–212.

Koshy, G. (2014). Calculating OpenGL perspective matrix from opencv intrinsic matrix, `http://kgeorge.github.io`.

Krainin, M., Henry, P., Ren, X. and Fox, D. (2010). Manipulator and object tracking for in hand 3D object modeling, *Technical Report UW-CSE-10-09-01*, University of Washington, Seattle, WA.

Kummerle, R., Grisetti, G., Strasdat, H., Konolige, K. and Burgard, W. (2011). g2o: A general framework for graph optimization, *2011 IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China*, pp. 3607–3613.

Leutenegger, S., Chli, M. and Siegwart, R. (2011). Brisk: Binary robust invariant scalable keypoints, *2011 IEEE International Conference on Computer Vision (ICCV), Barcelona, Spain*, pp. 2548–2555.

Lowe, D.G. (1999). Object recognition from local scale-invariant features, *Proceedings of the International Conference on Computer Vision, ICCV'99, Kerkyra, Greece*, Vol. 2, pp. 1150–1157.

Luck, J., Little, C. and Hoff, W. (2000). Registration of range data using a hybrid simulated annealing and iterative closest point algorithm, *ICRA'00: IEEE International Conference on Robotics and Automation, San Francisco, CA, USA*, Vol. 4, pp. 3739–3744.

Łępicka, M., Kornuta, T. and Stefańczyk, M. (2016). Utilization of colour in ICP-based point cloud registration, *in* R. Burduk *et al.* (Eds.), *Proceedings of the 9th International Conference on Computer Recognition Systems (CORES 2015)*, Advances in Intelligent Systems and Computing, Vol. 403, Springer, Berlin/Heidelberg, pp. 821–830.

Mair, E., Hager, G.D., Burschka, D., Suppa, M. and Hirzinger, G. (2010). Adaptive and generic corner detection based on the accelerated segment test, *Computer Vision (ECCV 2010), Hersonissos, Greece*, pp. 183–196.

Marder-Eppstein, E., Berger, E., Foote, T., Gerkey, B. and Konolige, K. (2010). The office marathon: Robust navigation in an indoor office environment, *2010 IEEE International Conference on Robotics and Automation (ICRA), Anchorage, AK, USA*, pp. 300–307.

Martínez Mozos, O., Triebel, R., Jensfelt, P., Rottmann, A. and Burgard, W. (2007). Supervised semantic labeling of places using information extracted from sensor data, *Robotics and Autonomous Systems* **55**(5): 391–402.

May, S., Dröschel, D., Fuchs, S., Holz, D. and Nuchter, A. (2009). Robust 3D-mapping with time-of-flight cameras, *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009, St. Louis, MO, USA*, pp. 1673–1678.

Men, H., Gebre, B. and Pochiraju, K. (2011). Color point cloud registration with 4D ICP algorithm, *2011 IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China*, pp. 1511–1516.

Miksik, O. and Mikolajczyk, K. (2012). Evaluation of local detectors and descriptors for fast feature matching, *21st International Conference on Pattern Recognition (ICPR), Tsukuba, Japan*, pp. 2681–2684.

Muja, M. and Lowe, D.G. (2009). Fast approximate nearest neighbors with automatic algorithm configuration, *International Conference on Computer Vision Theory and Application VISSAPP'09, Lisbon, Portugal*, pp. 331–340.

Muja, M. and Lowe, D.G. (2012). Fast matching of binary features, *9th Conference on Computer and Robot Vision (CRV), Toronto, Canada*, pp. 404–410.

Nistér, D., Naroditsky, O. and Bergen, J. (2004). Visual odometry, *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2004, Washington, DC, USA*, Vol. 1, pp. I–652.

Nowicki, M. and Skrzypczyński, P. (2014). Performance comparison of point feature detectors and descriptors for visual navigation on android platform, *2014 International Wireless Communications and Mobile Computing Conference (IWCMC), Nicosia, Cyprus*, pp. 116–121.

Pfister, H., Zwicker, M., van Baar, J. and Gross, M. (2000). Surfels: Surface elements as rendering primitives, *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH'00, New Orleans, LA, USA*, pp. 335–342.

Pomerleau, F., Colas, F., Siegwart, R. and Magnenat, S. (2013). Comparing ICP variants on real-world data sets, *Autonomous Robots* **34**(3): 133–148.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R. and Ng, A. (2009). ROS: An open-source robot operating system, *Proceedings of the Open-Source Software Workshop at the International Conference on Robotics and Automation (ICRA), Kobe, Japan*.

Rosten, E. and Drummond, T. (2006). Machine learning for high-speed corner detection, *in* A. Leonardis *et al.* (Eds.), *Computer Vision—ECCV 2006*, Lecture Notes in Computer Science, Vol. 3951, Springer, Berlin/Heidelberg, pp. 430–443.

Rusu, R.B., Bradski, G., Thibaux, R. and Hsu, J. (2010). Fast 3D recognition and pose using the viewpoint feature histogram, *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Taipei, Taiwan*, pp. 2155–2162.

Rusu, R.B. and Cousins, S. (2011). 3D is here: Point Cloud Library (PCL), *International Conference on Robotics and Automation, Shanghai, China*, pp. 1–4.

Shi, J. and Tomasi, C. (1994). Good features to track, *1994 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR'94, Seattle, WA, USA*, pp. 593–600.

Sipiran, I. and Bustos, B. (2011). Harris 3D: A robust extension of the Harris operator for interest point detection on 3D meshes, *The Visual Computer* **27**(11): 963–976.

Skrzypczyński, P. (2009). Simultaneous localization and mapping: A feature-based probabilistic approach, *International Journal of Applied Mathematics and Computer Science* **19**(4): 575–588, DOI: 10.2478/v10006-009-0045-z.

Song-Ho, A. (2013). OpenGL projection matrix, `http://www.songho.ca/opengl/gl_projectionmatrix.html`.

Steder, B., Rusu, R.B., Konolige, K. and Burgard, W. (2011). Point feature extraction on 3D range scans taking into account object boundaries, *2011 IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China*, pp. 2601–2608.

Thrun, S. and Leonard, J.J. (2008). Simultaneous localization and mapping, *in* B. Siciliano and O. Khatib (Eds.), *Handbook of Robotics*, Springer, Berlin/Heidelberg, pp. 871–890.

Tombari, F., Salti, S. and Di Stefano, L. (2010). Unique signatures of histograms for local surface description, *in* K. Daniilidis *et al.* (Eds.), *Computer Vision—ECCV 2010*, Lecture Notes in Computer Science, Vol. 6314, Springer-Verlag, Berlin/Heidelberg, pp. 356–369.

Tombari, F., Salti, S. and Di Stefano, L. (2011). A combined texture-shape descriptor for enhanced 3D feature matching, *18th IEEE International Conference on Image Processing (ICIP), Brussels, Belgium*, pp. 809–812.

Triebel, R., Pfaff, P. and Burgard, W. (2006). Multi-level surface maps for outdoor terrain mapping and loop closing, *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, Beijing, China*, pp. 2276–2282.

Weise, T., Wismer, T., Leibe, B. and Van Gool, L. (2009). In-hand scanning with online loop closure, *IEEE 12th International Conference on Computer Vision Workshops (ICCV Workshops), Kyoto, Japan*, pp. 1630–1637.

Whelan, T., Johannsson, H., Kaess, M., Leonard, J. and McDonald, J. (2013). Robust real-time visual odometry for dense RGB-D mapping, *2013 IEEE International Conference on Robotics and Automation (ICRA), Karlsruhe, Germany*, pp. 5724–5731.

Wurm, K. M., Hornung, A., Bennewitz, M., Stachniss, C. and Burgard, W. (2010). OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems, *ICRA 2010 Workshop, Taipei, Taiwan*.

Zhang, Z. (1994). Iterative point matching for registration of free-form curves and surfaces, *International Journal of Computer Vision* **13**(2): 119–152.

Artur Wilkowski received his M.Sc. degree in applied computer science in 2004 and a Ph.D. degree in computer science in 2012 from the Warsaw University of Technology (both with honors). Since 2006 he has been a research and teaching employee at that university, and from 2013 to 2015 has was also engaged by the Industrial Research Institute for Automation and Measurements. His research interests cover the area of computer vision, pattern recognition and artificial intelligence, and their applications in robotics, biometrics and surveying. He is an author of over 30 scientific papers and research reports concerning these topics.

Tomasz Kornuta graduated from the Warsaw University of Technology (WUT) in 2003 (B.Sc.) and 2005 (M.Sc). In 2013 he received the Ph.D. degree from the same university (with honors). Since 2008 he has been with the Institute of Control and Computation Engineering (ICCE) at the WUT, as of 2009 as the head of the Laboratory of the Foundations of Robotics. His scientific interests include robot control, software engineering for robotics, robot ontologies, intelligent robot behaviours and active perception (visual and RGB-D-based perception

combined with force sensing). He is the author/coauthor of over 50 publications regarding those subjects. He has also served as a reviewer for several national and international robotic conferences.

Maciej Stefańczyk graduated from the Faculty of Electronics and Information Technology at the Warsaw University of Technology, Poland, from which he obtained his B.Sc. and M.Sc. degrees, in 2010 and 2011, respectively (both with honors). He is currently a Ph.D. student at the same university, interested in the use of active vision-based systems extended by knowledge-base in robot control systems. His main research interests include utilization of the visual information in both robotics and computer entertainment systems.

Włodzimierz Kasprzak received a doctoral degree in pattern recognition (1997) from the University of Erlangen–Nuremberg, and a Ph.D. in computer science (1987) as well as a D.Sc. in automatic control (2001) from the Warsaw University of Technology (WUT). In 2014 he was awarded the professorial title. Since 1997 he has been with the Institute of Control and Computation Engineering at the WUT. His research interests are in pattern recognition and artificial intelligence methods, and their applications for image and speech analysis, robot perception and task planning. He is the author of 3 books as well as over 150 papers and research reports. Professor Kasprzak is a member of the IEEE and national societies of the IAPR (TPO, DAGM).

Appendix

Theoretical primer on frustum culling

A1. Frustum culling in computer graphics

Frustum culling is a common method used in computer graphics to restrict a visualization only to volumes actually observed by the virtual camera. Frustum culling may be performed in two ways. Firstly, it may be done after transforming all scene points into the camera frame and further into the *clipping space*. Secondly, it may be performed in the original scene space by computing clipping planes and comparing points against them. In both cases, it is useful to compute the so-called *projection matrix* first.

A2. Projection matrix

In the OpenGL library, the projection matrix $M_{4\times4}$ is the matrix transforming 3D point coordinates (in a camera coordinate system) lying in the frustum view into a box of extent $[-1,1]^3$. The transformation is performed using homogeneous coordinates:

$$k\begin{pmatrix}{}^{F}X\\ {}^{F}Y\\ {}^{F}Z\\ 1\end{pmatrix}=M\begin{pmatrix}{}^{C}X\\ {}^{C}Y\\ {}^{C}Z\\ 1\end{pmatrix}, \tag{A1}$$

where $({}^{C}X,\ {}^{C}Y,\ {}^{C}Z,1)^T$ are point coordinates in the camera space and $({}^{F}X,\ {}^{F}Y,\ {}^{F}Z,1)$ are the resulting *clipping space* coordinates. Now, after transforming each point, it is easy to decide whether or not it lies in the frustum, since all points inside the frustum are mapped into $[-1,1]^3$, and all points outside it are mapped out of this box.

A3. 3D clipping planes

The projection matrix allows finding also the parameters of the clipping planes, which is of more interest in the solution presented in this paper. In the OpenGL formulation of the projection matrix, the parameters of the clipping planes are formed by the sums and differences of adequate rows of the projection matrix M, as given by Gribb and Hartmann (2001). Thus, one obtains plane parameters in the camera reference frame. Because parameters of planes are required with respect to the global reference frame, corresponding rows of the *projection-view matrix* must be used, which is a matrix resulting from multiplication of the rigid global-to-camera transformation and the *projection-matrix*, i.e., $M\,{}^{C}_{G}T$. The resulting parameters of the clipping planes require only standardization into Hessian normal form.

A4. Projection matrix for real-world cameras

The formulation of projection matrix M in the OpenGL library is given as follows (Song-Ho, 2013):

$$M=\begin{pmatrix}\frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0\\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0\\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n}\\ 0 & 0 & -1 & 0\end{pmatrix}, \tag{A2}$$

where f and n are respectively the z-coordinates of the near and far clipping plane and, l, r, t, b are left, right, top and bottom bounds of the visible image plane (it is assumed that the optical axis goes through the point $(0,0)$ in the image plane).

In the case of real-world cameras, the optical axis is assumed to cross the point (c_x,c_y) in the image plane, and the camera intrinsics are described by the following matrix:

$$K=\begin{pmatrix}\alpha & 0 & c_x\\ 0 & \beta & c_y\\ 0 & 0 & 1\end{pmatrix}, \tag{A3}$$

where α and β are the focal lengths in the x and y directions, respectively. Using these symbols and taking into account that in OpenGL the z-axis points towards the camera, one can rewrite the projection matrix M as

follows:

$$M = \begin{pmatrix} \frac{2\alpha}{w} & 0 & \frac{2c_x}{w} - 1 & 0 \\ 0 & \frac{2\beta}{h} & \frac{2c_y}{h} - 1 & 0 \\ 0 & 0 & \frac{(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{pmatrix}, \tag{A4}$$

where w and h are respectively the image width and height in pixels. It is easy to observe that each point located between the far and the near plane, which is projected inside the image bounds, is mapped into the cube $[-1, 1]^3$ when transformed by the M matrix. For instance, the ${}^{C}X$ coordinate is effectively computed as

$${}^{F}X = \left(\frac{{}^{C}X}{{}^{C}Z} \alpha + c_x \right) / w \cdot 2 - 1. \tag{A5}$$

Noting the fact that the expression in parentheses is simply the point's x-coordinate in the image, we see that all points projecting into the camera image must have "clip space" coordinates within the $[-1, 1]$ bounds. A more detailed discussion regarding the relations between camera intrinsics and projection matrices (although for non-eccentric cameras) is given by Koshy (2014).

Received: 13 November 2014
Revised: 1 May 2015
Re-revised: 24 August 2015