Large Scale and Long Standing Simultaneous Reconstruction and Segmentation

Keisuke Tateno[1,2] and Federico Tombari[1,3] and Nassir Navab[1,4]

[1] *Chair for Computer Aided Medical Procedures (CAMP), TU Munich, Munich (Germany)*

[2] *Canon Inc., Shimomaruko, Tokyo (Japan)*

[3] *Dipartimento di Informatica: Scienza e Ingegneria (DISI), University of Bologna, Bologna (Italy)*

[4] *Johns Hopkins University, Baltimore (USA)*

Abstract

This work proposes a method to segment a 3D point cloud of a scene while simultaneously reconstructing it via Simultaneous Localization And Mapping (SLAM). The proposed method incrementally merges segments obtained from each input depth image in an unified global model leveraging the camera pose estimated via SLAM. Differently from other approaches, our method is able to yield segmentation of scenes reconstructed from multiple views in real-time and with a complexity that does not depend on the size of the global model. Moreover, we endow our system with two additional contributions: a loop closure approach and a failure recovery and re-localization approach, both specifically designed so to enforce global consistency between merged segments, thus making our system suitable for large scale and long standing reconstruction and segmentation. We validate our proposal against the state of the art in terms of computational efficiency and accuracy on several benchmark datasets, as well as by showing how our method enables real-time reconstruction and segmentation of diverse real indoor environments.

Keywords: Dense SLAM, Segmentation, Real-time, Scalable, Long standing, Relocalization, Loop-closure

1. Introduction and Related Work

Scene segmentation is one of the most important and researched topics in the field of robotic perception, since segmentation is typically a pre-requisite for several robotic tasks such as object modeling and object recognition [1], autonomous grasping and manipulation of objects [2], object tracking [3], and scene understanding and object discovery of unknown environments [4]. Within the robotic perception and computer vision communities, a great effort has been made to develop efficient 3D segmentation algorithms, i.e. real-time processing of depth maps obtained from RGB-D or 3D sensors. The focus on 3D data is motivated by the additional insight that geometry and shape provide, with respect to texture and color, for the task of segmentation, as well as the opportunity to determine segments that lie in the 3D space in front of the robot and not just on the image plane. Fast real-time segmentation of depth maps has been recently investigated by the works of Uckermann et al. [5, 6], Pieropan et al. [7] and Abramov et al. [8].

Recently, 3D reconstruction methods, which aim at a real-time registration of depth maps from multiple viewpoints obtained from a moving sensor, are becoming increasingly exploited for higher level robotic perception tasks, since they offer additional information for the surrounding environment and are fundamental for robot navigation tasks: this is the case of Kinect Fusion [9], as well as dense SLAM [10, 11, 12]. While the former method yields a 3D mesh of the reconstructed environment by exploiting a specific data representation internally deployed, the output of SLAM methods is generally in the form of a 3D point cloud.

As a consequence, in addition to segmentation methods aimed at processing single depth maps, some works have recently addressed the problem of segmenting 3D reconstructions obtained via Kinect Fusion or SLAM. Toward this goal, segmentation methods specifically devised to work on 3D meshes [13] or point clouds [14, 15, 16] are generally deployed to yield a segmentation of such 3D representations, as proposed in the object discovery approach of [4]. One main limitation of such methods is clearly the computational cost, since they cannot run in real-time. This aspect strongly limits their use in those application scenarios characterized by real-time constraints, as it is the case in most of the aforementioned robotic perception tasks. In addition, their computational burden tends to increase with the size of the 3D mesh or point cloud. Hence, to limit the overall computational requirements, only up to a certain number of merged depth maps can be deployed with an off-the-shelf hardware.

Aiming at the same goal, [17] proposes incremental object segmentation in a dense RGB-D SLAM framework. The method is based on Kintineous [10], a dense RGB-D SLAM approach which builds upon KinectFusion [9], and relies on a 3D representation called Truncated Sign Distance Function (TSDF). In this method, newly merged depth frames in the TSDF are, from time to time, extracted in the form of "slices" (i.e., a 3D mesh) according to the estimated camera position, segmented via graph-based segmentation [13], and merged into a global segmentation map. Although this yields a much higher

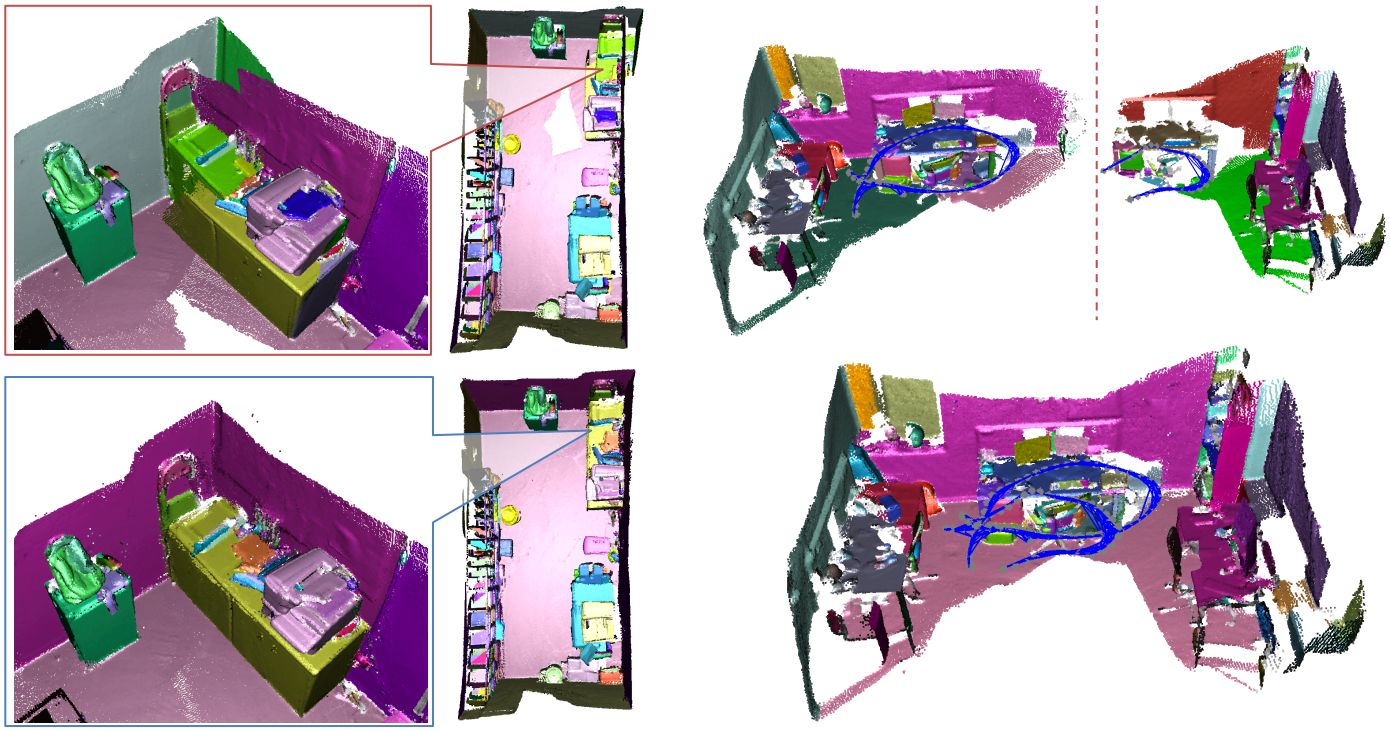

Figure 1: The proposed framework is capable of providing a long-standing SLAM reconstruction embedding a globally consistent segmentation of the scene (bottom, left and right). It involves specific approaches for loop closure and failure recovery, which can enforce global label consistency among the segments while, respectively, globally optimizing the camera poses and handling tracking failures. This can be seen by comparing the top left image (before loop closure) with the bottom right one (after loop closure). In addition, an example of our failure recovery is portraryed in the bottom right image, which shows the merged result of the two independent sequences shown in the top right image. Blue lines depicts the edges of the key-frame graph, while grey points represent the estimated camera poses.

efficiency than the previous approaches, the use of a segmentation method such as [13] on each "slice" (represented as a 3D mesh), as well as the fact that the segments extracted from each slice are successively merged in the global segmentation map, still does not allow this method to produce the segmentation of the current input data in real-time, and, as stated in [17], generates an overall computational complexity that grows with the size of the global segmentation map.

Moreover, the work by Salas-Moreno et al. [18] aims at real-time plane segmentation and SLAM reconstruction. In this method, planes are segmented from each input depth map, then they are incrementally merged into a global model. The main limitation of such a method in the context of segmentation is the fact that it considers only planar surfaces, while curved surfaces are not segmented, posing a limit to the generality of the processed shapes especially in the presence of arbitrary objects.

To tackle this problem, [19] proposed a SLAM-based place recognition method based on object detection. The method recognizes the same place on different maps by detecting object segments that are common between partially overlapping maps. Although this method can attain label consistency of the objects appearing in different maps, it is still limited to specific shapes, since the object detection method relies on the objectness which cannot detect relatively flat surfaces. Moreover, the segment merging procedure is only partial, since it is carried out only on those segments which are recognized as objects.

Our approach aims at overcoming the limitations of the methods currently available in literature by simultaneously performing reconstruction and segmentation. In particular, while reconstruction is carried out within a point-based fusion SLAM framework so to deal with noisy 3D data, segments are extracted from the current depth frame and incrementally merged together so to build a Global Segmentation Map (GSM) of the reconstructed environment. As a consequence, it achieves the important advantage of a constant runtime regardless of the size of the GSM and the number of merged depth maps in the global 3D model, which makes our approach particularly suited to large-scale and long-standing reconstruction scenarios. Furthermore, at each frame, the update procedure is efficient enough to be carried out in real-time.

In addition to this approach, our work also includes two additional contributions, aimed at making our system apt to deal with large-scale and long-standing acquisitions. The first is a peculiar loop closure stage, that, when loop closures are detected, aims at obtaining a globally refined alignment of the camera poses and of the segment labels so to yield a reconstructed point cloud which is globally consistent not only in terms of 3D geometry, but also in terms of 3D segments. An example is shown in Fig. 1, which illustrates the difference before (top image) and after (bottom image) applying the proposed label-consistent loop closure approach. The second contribution is the ability to detect when tracking fails to register

the current depth map to the global model due to the presence of significant occlusions, rapid camera motions or presence of big planar surfaces, and to perform re-localization of the newly tracked sequences via key-frame matching. Also in this case, the proposed approach aims at enforcing global consistency in terms of detected segments. This yields a completely automatic approach for failure recovery, suited to long-standing reconstruction and segmentation of large-scale environments.

With regards to the experimental results, we first evaluate our simultaneous reconstruction and segmentation framework against the state-of-the-art segmentation algorithms for 3D meshes and point clouds in terms of efficiency and accuracy on a benchmark dataset, as well as qualitatively on diverse indoor environments. Moreover, we evaluate the proposed loop closure and failure recovery approaches on a benchmark dataset suited to long-term reconstruction and characterized by multiple tracking failures. Finally, we show qualitative results of the large-scale reconstruction and segmentation obtained by our approach.

The paper is structured as follows. In Section 2, we describe the framework for simultaneous reconstruction and segmentation. Successively, in Sections 3 and 4, we illustrate the proposed loop closure approach and failure recovery algorithm, respectively. In Section 5, we provide experimental evaluation of the three main contributions of the paper. Finally, in Section 6, we draw the conclusions.

2. Simultaneous Reconstruction and Segmentation

This section describes the proposed framework that performs simultaneous reconstruction and segmentation at each input depth frame. The flow diagram in Fig. 2 sketches the pipeline of the algorithm. In this flowchart, the stages regarding SLAM reconstruction are shown as blue boxes, while the stages regarding segmentation are shown as red.

The SLAM algorithm employed by our system is based on the point-based fusion method of Keller et al. [20]. Instead of the standard voxel-based representations, such an approach uses a point-based representation with normal information referred to as *surfel*, i.e. a disk-shaped entity which describes locally planar regions without connectivity information. The advantage of using surfel-based representations is that common operations of map entities such as data association, insertion, averaging and removal can be performed at a lower memory footprint compared to voxel-based representations like Kinect Fusion [9]. Throughout this approach, the camera pose of each input depth map is estimated and each depth map is merged into the global model.

At the segmentation stages, the objective is to incrementally (and synergically with respect to the SLAM reconstruction) build up a Global Segmentation Map (GSM) in real-time by properly propagating and merging segments extracted from each depth map. Towards this goal, each depth map is segmented first (Depth Map Segmentation stage). Although any segmentation method devised to work on depth maps can be employed at this stage, our approach uses an adapted version of the real-time method proposed in [6] (described in Sec. 2.2.1).

Successively, during the Segment Label Propagation stage, the segments building up the GSM and associated with each surfel on the global model are propagated to the current depth map by means of the estimated camera pose obtained via SLAM. In particular, propagated segments are only those whose surfaces overlap with the current geometry of the depth map. This allows us to assign a label to each segment of the depth map, which is consistent with that of the GSM (detailed in Sec. 2.2.2). The goal of the successive Segment Merging stage is to detect and merge segments in the GSM that have the same correspondent in the current depth map, a circumstance that typically occurs when, in the previous views, the underlying surface had occluding foreground objects. We thus exploit new vantage points along the SLAM sequence to improve the consistency of the GSM (illustrated in SubSec. 2.2.3). Finally, in the Segment Update stage, the labels of the GSM are updated with the labels computed from the current depth map during the Segment Label Propagation stage. Since the frame-wise depth map segmentation might be noisy, it is undesirable to update the segment label directly on the GSM from the corresponding segment label on the depth map. Therefore, we assign each element of the GSM to a confidence based on the various label observations, so that the update process can be carried out only after each propagated label has reached a certain confidence level, thus removing noisy label associations. This is explained in Sec. 2.2.4.

In the next Subsection, we introduce notation and the stages of our pipeline devoted to SLAM reconstruction. Moreover, the stages carrying out the incremental segmentation are described in Subsection 2.2.

2.1. SLAM Reconstruction

As proposed in [20], a *global model* (i.e., the output of the SLAM reconstruction) consists of an unorganized set of surfels $s_1, \dots, s_k \in \mathcal{S}$, where each surfel s_k is characterized by a 3D position $v_k \in \mathbb{R}^3$, a normal $n_k \in \mathbb{R}^3$, a radius $r_k \in \mathbb{R}$, a confidence $c_k \in \mathbb{R}$ and a time stamp $t_k \in \mathbb{N}$. In the Preprocessing stage, a depth image $\mathcal{D}_t$ at current frame t is transformed into a metric vertex map $\tilde{\mathcal{V}}_t(\boldsymbol{u}) = \boldsymbol{P}^{-1}\dot{\boldsymbol{u}}\mathcal{D}_t(\boldsymbol{u})$, with the camera intrinsic matrix $\boldsymbol{P}$, a depth map element $\boldsymbol{u} = (x, y)^\top$ in the image domain $\boldsymbol{u} \in \Omega \subset \mathbb{R}^2$ and its homogeneous representation $\dot{\boldsymbol{u}}$. The vertex map $\tilde{\mathcal{V}}_t$ is then smoothened by applying a bilateral filter [21] so as to generate a version of the vertex map with less noise, $\mathcal{V}_t$. Thereafter, the normal map of current frame $\mathcal{N}_t$ is simply generated from the vertex map $\mathcal{V}_t$ by central differences.

In the Camera Pose Estimation stage, the current 3D camera pose $\boldsymbol{T}_t = [\boldsymbol{R}_t, \boldsymbol{t}_t] \in \mathbb{SE}(3)$ at frame t, which is composed of a 3×3 rotation matrix $\boldsymbol{R}_t \in \mathbb{SO}(3)$ and a 3D translation vector $\boldsymbol{t}_t \in \mathbb{R}^3$, is updated by incrementally aligning the filtered vertex map $\mathcal{V}_t$ with the global model in the form of the vertex map rendered from the previously estimated camera pose, $\mathcal{V}^m_{t-1}$ (hereinafter, we use the superscript "m" to indicate the rendered version of a 3D point cloud with respect to a particular camera pose). The alignment is obtained using dense ICP with a point-to-plane error metric computed by means of the rendered normal map, $\mathcal{N}^m_{t-1}$, and the fast projective data association algorithm proposed in [9].

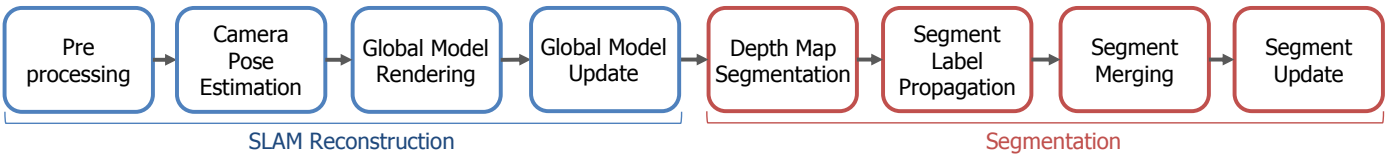

Figure 2: Flow diagram of the proposed incremental segmentation pipeline applied at each input depth map.

During the Global Model Rendering stage, to compute correspondences between the points on the current depth map and the surfels on the global model, a index map $\mathcal{I}$ is created which maps each surfel vertex of the global model $\boldsymbol{v} \in \mathcal{S}$ with its projection on the current depth map at element $\boldsymbol{u} = \pi(\boldsymbol{P}\boldsymbol{T}_t^{-1}\boldsymbol{v})$ via standard pin-hole projection function π based on the updated camera pose $\boldsymbol{T}_t$. Additionally, the model vertex map $\mathcal{V}_t^m$ and model normal map $\mathcal{N}_t^m$ are rendered to be used for the camera pose estimation of the successive frame.

Finally, in the Global Model Update stage, the new surfel measurements $\boldsymbol{v}$ obtained from the current depth map are either added as *unstable* surfels or merged with the already present surfels. The criteria of adding or merging is based on the difference of the 3D position of corresponding surfels, as well as on the angle of their associated normals. If the new surfel measurement is matched to certain already present surfels, the new surfel is merged to the matched surfel. Merging $\boldsymbol{v}$ with a surfel already present in $\mathcal{S}$ increments the associated confidence c. After a certain number of stable measurements, unstable surfels change their status to stable: this occurs when the associated confidence grows above a threshold (set to 5 measurements). Finally, if stable surfels in $\mathcal{S}$ are observed in front of a new surfel measurement, the surfel is removed from the global model and considered as an unstable point belonging to a dynamic object.

2.2. Incremental Segmentation

These stages assume, as input, the current depth map $\mathcal{D}_t$ as well as the global model reconstructed via SLAM up to the current frame, $\mathcal{S}$, as described in Sec. 2.1. Each depth map is also associated with a vertex map, $\mathcal{V}_t$, that stores the 3D vertices $\boldsymbol{v}(\boldsymbol{u})$ of each element $\boldsymbol{u}$ in the depth map $\mathcal{D}_t$. The goal is to incrementally build up and update a GSM $\mathcal{L}$, which has the same number of elements as the global model $\mathcal{S}$, in which each element represents a segment label. To this end, $\mathcal{L}$ is updated, at each new frame, with the segmentation information associated with the current depth map through the four stages illustrated in the succeeding Subsections.

2.2.1. Depth map segmentation

To segment each input depth map $\mathcal{D}_t$, we employ a fast segmentation method inspired by the *normal edge* analysis carried out in [6], where, at each frame, a binary *edge map* is computed by comparing nearby normal angles, these edges representing the segment boundaries. In contrast to [6], we propose to extract only convex-shape segments by explicitly detecting concave boundaries. Indeed, this choice is motivated by recent works exploiting graph-based segmentation [4, 17, 31], that introduced a penalty for concave regions based on the assumption that real-world objects mainly consist of convex shapes.

Figure 3: Normal map (left), normal and geometrical edge map (middle) and resulting depth map segmentation with shading (right)

Inspired by these methods, we adapt the concave region penalty to the normal edge-based segmentation deployed in our pipeline. First, for each element $\boldsymbol{u}$ in the depth map $\mathcal{D}_t$, we detect concave boundaries by computing the dot product between the normal at $\boldsymbol{u}$, denoted as $\boldsymbol{n}(\boldsymbol{u})$, and each normal of the 8-connected neighboring points of $\boldsymbol{u}$, $\boldsymbol{n}(\boldsymbol{u}_i)$ for $i = 1, \ldots, 8$. In particular, we define an operator, $\Phi_i(\boldsymbol{u})$, as follows:

$$\Phi_i(\boldsymbol{u}) = \begin{cases} 1 & (\boldsymbol{v}(\boldsymbol{u}_i) - \boldsymbol{v}(\boldsymbol{u})) \cdot \boldsymbol{n}(\boldsymbol{u}) > 0 \\ \boldsymbol{n}(\boldsymbol{u}) \cdot \boldsymbol{n}(\boldsymbol{u}_i) & \text{otherwise} \end{cases} \quad (1)$$

where $\boldsymbol{v}(\boldsymbol{u})$ and $\boldsymbol{v}(\boldsymbol{u}_i)$ being the associated 3D vertices from the vertex map $\mathcal{V}_t$. This operator takes the value 1 (i.e., its maximum value) when $\boldsymbol{n}(\boldsymbol{u})$ and $\boldsymbol{n}(\boldsymbol{u}_i)$ lie on a convex surface, while it takes the dot product between such normals if they lie on a concave shape. Hence, the higher the concavity, the lower the value of $\Phi_i(\boldsymbol{u})$. We compute such an operator for all 8 neighbors, then take its minimum:

$$\Phi(\boldsymbol{u}) = \min_{i=1,\ldots,8} \{\Phi_i(\boldsymbol{u})\} \ . \quad (2)$$

Such an operator highlights concavities along at least one of the eight directions around element $\boldsymbol{u}$. We thus compute our concavity-aware normal edge map by thresholding $\Phi(\boldsymbol{u})$ (we set the threshold 0.94).

Another cue that we take into consideration is the distance between two vertices in the 3D space: indeed, a commonly deployed assumption is that depth borders define 3D segment borders. To reach this goal, we define another operator, $\Gamma(\boldsymbol{u})$, that takes into account the maximum 3D point-to-plane distance between an element $\boldsymbol{u} \in \mathcal{D}_t$ and its 8 neighbors:

$$\Gamma(\boldsymbol{u}) = \max_{i=1,\ldots,8} \left\{ \left| (\boldsymbol{v}(\boldsymbol{u}_i) - \boldsymbol{v}(\boldsymbol{u})) \cdot \boldsymbol{n}(\boldsymbol{u}) \right| \right\} \ . \quad (3)$$

To threshold $\Gamma(\boldsymbol{u})$, we use an uncertainty measure $\sigma_d(\boldsymbol{u})$ computed following the noise model proposed in [22]. Such a measure adaptively takes into account the noise level of each point

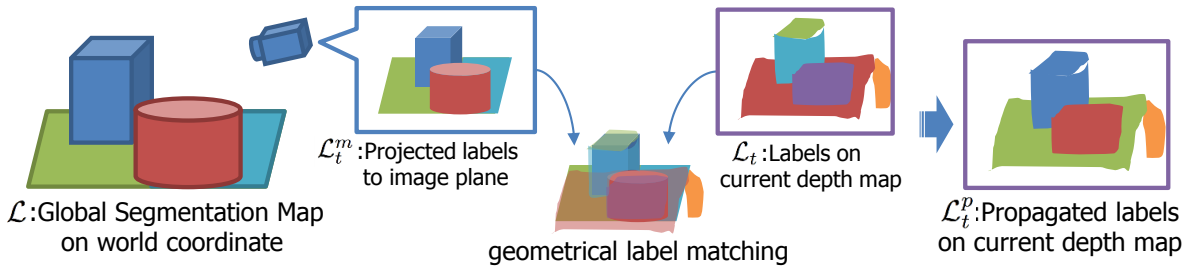

Figure 4: Toy example explaining the proposed Segment Propagation stage: first segments from the GSM are re-projected onto the current camera plane; then, they are compared with those of the current depth map, so to identify corresponding segments between GSM and the camera plane.

of the depth map, assuming a noise model that increases with the distance from the sensor. By thresholding $\Gamma(\boldsymbol{u})$, we obtain a set of geometrical edges, which are added to the previously computed normal edges to yield the final edge map (shown in Fig. 3, middle). Finally, we apply a connected component analysis algorithm to the edge map obtained to yield a label map $\mathcal{L}_t$, where each element $\boldsymbol{u}$ is associated with a segment label $\mathcal{L}_t(\boldsymbol{u}) = l_j$ (shown in Fig. 3, right). In this map, the label 0 (unlabeled segment) is assigned to all points lying within the detected concave regions and depth border regions.

2.2.2. Segment Label Propagation

As anticipated, the goal of this stage is to propagate the segments of the GSM that are visible from the current camera viewpoint onto the label map of the current depth map. With this in mind, correspondences between the visible segments of the GSM, $l_i \in \mathcal{L}$, and those on the current depth map, $l_j \in \mathcal{L}_t$, are estimated by checking whether the underlying surface of both segments are the same. The label propagation procedure is illustrated in Fig. 4. As previously mentioned, given the currently estimated camera pose at time t, we re-project the global model onto its image plane, yielding a vertex map $\mathcal{V}_t^m$ and a normal map $\mathcal{N}_t^m$. At the same time, we also compute the vertex map and the normal map associated with the current depth map, i.e. $\mathcal{V}_t$, $\mathcal{N}_t$ (note the absence of superscript "m" to distinguish them). In a similar fashion, we re-project the GSM, $\mathcal{L}$, on the same image plane, obtaining a label depth map $\mathcal{L}_t^m$. Note that this re-projection allows us to focus only on the elements of the GSM currently visible from the current camera viewpoint and to discard all GSM elements that are either occluded or outside of the camera's field of view, thus resulting in efficiency and scalability with respect to the global model size.

To efficiently determine segment correspondences between $\mathcal{L}_t$ and $\mathcal{L}_t^m$, first the number of corresponding points $\Pi(l_i, l_j)$ between all points with label $l_i \in \mathcal{L}_t^m$ and all points with label $l_j \in \mathcal{L}_t$ are determined. This is done by considering each pair of elements $l_i = \mathcal{L}_t^m(\boldsymbol{u}), l_j = \mathcal{L}_t(\boldsymbol{u}), \forall\, \boldsymbol{u} \in \mathcal{D}_t$, and by thresholding the distance of the corresponding vertices along the viewing ray and the angle between the corresponding normals:

$$\left| (\mathcal{V}_t(\boldsymbol{u}) - \mathcal{V}_t^m(\boldsymbol{u})) \cdot \frac{\mathcal{V}_t(\boldsymbol{u})}{|\mathcal{V}_t(\boldsymbol{u})|} \right| < \sigma_d(\boldsymbol{u}) \tag{4}$$

$$\cos^{-1}(\mathcal{N}_t(\boldsymbol{u}) \cdot \mathcal{N}_t^m(\boldsymbol{u})) > \tau_{\mathcal{N}} \tag{5}$$

where $\sigma_d(\boldsymbol{u})$ is the depth uncertainty measure previously used, and $\tau_{\mathcal{N}}$ is the normal angle threshold (in our experiments, $\tau_{\mathcal{N}} = 20°$). When both conditions are satisfied, $\Pi(l_i, l_j)$ gets incremented. We normalize this term by the size of the corresponding segment on $\mathcal{L}_t$:

$$\tilde{\Pi}(l_i, l_j) = \frac{\Pi(l_i, l_j)}{\#(l_j)} \tag{6}$$

(we refer to # as the cardinality operator for a segment).

Hence, $\tilde{\Pi}(l_i, l_j)$ represents the percentage of 3D overlap of segment $l_j \in \mathcal{L}_t$ with $l_i \in \mathcal{L}_t^m$, computed as the intersection between the two segment point sets and normalized by the magnitude of the point set of l_j. It can be regarded as a confidence measure for both segments to lie on the same 3D surface. We can thus easily associate each segment on $\mathcal{L}_t$ with the maximally-overlapping segment on $\mathcal{L}_t^m$ as follows:

$$\tilde{\Pi}_{max}(l_j) = \max_{l_i \in \mathcal{L}_t^m} \left\{ \tilde{\Pi}(l_i, l_j) \right\} . \tag{7}$$

It is noteworthy to mention that, since we only take into account those labels of the GSM that appear on the current label map $\mathcal{L}_t^m$, the complexity of the operation in Equation (7) is independent from the size of the GSM.

Based on $\tilde{\Pi}_{max}(l_j)$, we build up a *propagated* label map $\mathcal{L}_t^p$ (depicted in Fig. 4, right) by applying to each element $l \in \mathcal{L}_t^p$ the following rule:

1. If $\tilde{\Pi}_{max}(l_j) \geq \tau_\Omega$, then $l = l_i$, i.e. we propagate the label of the GSM segment that yielded the highest overlap. τ_Ω is set, in our experiments, to 0.3 — i.e., we require at least 30% segment overlap;
2. otherwise, $l = l_j$, i.e. we propagate the label l_j directly from $\mathcal{L}_t$.

Finally, to avoid including in $\mathcal{L}_t^p$ new segments derived from possible data artifacts or bordering regions, we assign the label 0 to l every time the newly propagated segment from $\mathcal{L}_t$ has a size smaller than a certain threshold (50 pixels in our experiments).

2.2.3. Segment merging

During the label propagation stage, different segments on the GSM might correspond to the same segments on the depth map. This is typically the case for a surface which is initially separated into multiple segments due to occluding foreground objects: when the camera viewpoint changes, the occlusion might

Figure 5: Benefits of the proposed Segment Merging procedure: an object initially segmented into two parts due to an occluding foreground surface (left), is then merged into the same segment when a different camera view reveals the right connectivity (middle and right)

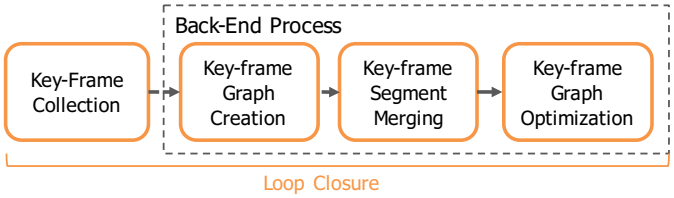

Figure 6: Flow diagram of the proposed pipeline for loop closure with global label consistency.

disappear to reveal that such multiple segments are indeed part of the same surface. An example of such a situation is depicted in Fig. 5. The goal of this stage is to identify and merge together these corresponding segments on the GSM.

To determine whether the underlying surface of the two segments is the same based on their 3D overlap, we rely on the same criterion used in the previous Segment Propagation stage. In particular, during the computation of term $\tilde{\Pi}(l_j)$, we identify all segments $l_i \in \mathcal{L}_t^m$ yielding an overlap higher than a certain threshold with $l_j \in \mathcal{L}_t$ (set to 0.2 in our experiments). Such segments form the label set L_{l_j}.

Since the segments obtained from the depth map are often noisy, it is not robust to perform such a merging procedure only from the observations derived from a single depth map. Therefore, we introduce a pairwise confidence that estimates to what extent a pair of segments are part of the same surface based on all input data seen so far, inspired by the confidence measure used on each surfel with the SLAM method [20]. Specifically, all possible label pairs (l_a, l_b), where $l_a, l_b \in L_{l_j}$ and $l_a \neq l_b$, are associated with a confidence $\Psi_t^m(l_a, l_b)$. If a pair (l_a, l_b) is identified for the first time, its associated confidence is initialized as follows: $\Psi_t^m(l_a, l_b) = 0$. Instead, when such a segment pair has been already observed, its associated confidence is updated by incrementing it:

$$\Psi_t^m(l_a, l_b) = \Psi_{t-1}^m(l_a, l_b) + 1 \,. \tag{8}$$

Finally, for all those segment pairs which are not observed at the current time t, but for which a confidence was already initialized in the previous frames, their confidence is updated as follows:

$$\Psi_t^m(l_a, l_b) = \max\left(0, \Psi_{t-1}^m(l_a, l_b) - 1\right) \,. \tag{9}$$

In addition, when the confidence associated to a certain segment pair grows higher than a specific threshold (set to 3 in our experiments), the segment pair (l_a, l_b) is merged by replacing the label l_a with label l_b in $\mathcal{L}$.

2.2.4. Segment Update

The last step of the incremental segmentation procedure concerns updating the GSM with the label map obtained at the end of the Segment Label Propagation stage, i.e. $\mathcal{L}_t^p$. Analogously to the Segment Merging stage, it is not robust to directly modify our GSM based on the label indications contained in $\mathcal{L}_t^p$, since such a label map, being based on one single depth frame, usually contains noisy information. Hence, and similarly to the previous stage, we follow a confidence based approach, by associating each element of the GSM map $\mathcal{L}(v)$ with a confidence $\Psi_t^u(v)$.

To initialize and update such a confidence, for each element $\mathcal{L}_t^p(\boldsymbol{u})$, we compute the corresponding GSM element $\mathcal{L}(v)$ by means of the estimated camera pose, and follow these three cases:

1. If $\mathcal{L}(v)$ is unlabeled (e.g., the corresponding surfel has been just added due to the new observation), it is set to the label of the new observation: $\mathcal{L}(v) = \mathcal{L}_t^p(\boldsymbol{u})$, while the confidence is initialized as $\Psi_t^u(v) = 0$.
2. If the labels of $\boldsymbol{u}$ and v are the same, the confidence is incremented:

$$\Psi_t^u(v) = \min\left(\Psi_{t-1}^u(v) + 1, \Psi_{max}^u\right) \tag{10}$$

   where Ψ_{max}^u is the cap value for the confidence associated surfels (set to 10 in our experiments).
3. If the labels of $\boldsymbol{u}$ and v are different, the label confidence is decreased:

$$\Psi_t^u(v) = \max\left(0, \Psi_{t-1}^u(v) - 1\right) \,. \tag{11}$$

When the confidence associated with a point $\mathcal{L}(v)$ drops to 0, it means that the segment is frequently corresponded to different label, and the current label is not confident. In such a case, the point is assigned the corresponding label on $\mathcal{L}_t^p$, i.e. $\mathcal{L}(v) = \mathcal{L}_t^p(\boldsymbol{u})$.

3. Loop Closure with Global Label Consistency

A typical problem of SLAM reconstruction algorithms is the accumulation of the camera pose estimation error from frame-to-frame, resulting in remarkable drifts in time. To alleviate such problems when dealing with large-scale reconstructions, graph optimization and loop closure are usually exploited. The objective is to detect when the current key-frame matches a previously seen one and perform a global optimization on the key-frame graph so to reduce the accumulated drift error in the overall reconstruction [12, 23]. Importantly, when applying the loop closure in our framework to produce a globally consistent segmentation map, we also need to enforce a global optimization in terms of overlapping segments between pairs of connected key-frames.

To this goal, we propose a loop closure and graph optimization approach based on four stages, as depicted in Fig. 6. Here,

the input is represented by a stream of depth maps acquired from a moving 3D/RGB-D sensor, which are processed one at a time. These four stages are run on an independent thread as a back-end process, in order to keep reconstruction and segmentation real-time. The proposed method is based on the pose-graph based loop closure of [12, 23]. To detect loop-closure, we use a place recognition approach based on matching the entire image associated to each key-frame [24]. In contrast to [12, 23, 24], our loop-closure enforces global label consistency on the GSM by matching segments on key-frames. In addition, we employ a set of synthesized depth map rendered from the Global Model to improve key-frame matching, as illustrated in the following.

To achieve this goal, we utilize the input variables given as a set of key-frames $k_1, .., k_n \in \mathcal{K}$ as well as the global model $\mathcal{S}$ and GSM $\mathcal{L}$. Each key-frame k_n consists of the key-frame pose $\boldsymbol{T}_{k_n}$, the model vertex map $\mathcal{V}^m_{k_n}$, the model normal map $\mathcal{N}^m_{k_n}$, the model label map $\mathcal{L}^m_{k_n}$ and the index map of its surfels $\mathcal{I}_{k_n}$, which are rendered from the global model (see Sec. 2.1).

3.1. Key-frame collection

While the system is carrying out reconstruction and segmentation, a subset of frames is collected as key-frames. To determine whether the current frame should be considered as a key-frame, we threshold the relative translation and angle distance between the current estimated pose and nearest pose from the accumulated set of key-frames $\mathcal{K}$. If both the angle and translation distance are larger than the threshold, the current frame is selected as key-frame. Note that the key-frame only relates to the surfels that are updated within the last 150 frames: such limitation aims to prevent a contamination of the key-frame due to inaccurate surfels caused by the presence of drift error.

3.2. Key-frame Graph Creation

After the insertion of a key-frame, key-frame pairs having overlapping viewing volumes and overlapping segments are detected by means of a spatial criterion and a similarity criterion, respectively. As for the former criterion, given a newly inserted key-frame k_i and the accumulated key-frames $k_1, .., k_n \in \mathcal{K}$, key-frame pair sharing overlapping viewing volume are detected by evaluating the similarity between their respective poses. The distance between two poses is the same as the one deployed for key-frame collection, i.e. based on thresholding the angle and translation distances. In particular, all key-frame pairs with a pose angle lower than 20 degrees are sorted by their translation distance. The best n_l key-frames are selected as potentially connected key-frames $k_1, .., k_{n_l} \in \mathcal{K}_l$ (n_l is set to 5 in our experiments).

This spatial criterion, based solely on the key-frame pose, can fail due to the presence of severe tracking drift affecting the pose of each key-frame. For this reason, we apply an additional similarity criterion based on matching the rendered depth maps associated to each key-frame. To compare them, we calculate the Sum of Absolute Distances (SAD) of the depth components of the vertex map between k_i and all other key-frames:

$$\mathrm{SAD}(k_i, k) = \sum_{\boldsymbol{u} \in \mathcal{D}_t} \left| \mathcal{V}^m_{k_i}(\boldsymbol{u}).z - \mathcal{V}^m_{k}(\boldsymbol{u}).z \right|, \quad k \in \mathcal{K} \tag{12}$$

where the operator $.z$ indicates the depth component of the vertex map. All key-frame pairs are sorted based on the obtained SAD distance, and the best n_s are retained as potential loop-closing key-frames $k_1, .., k_{n_s} \in \mathcal{K}_s$ (n_s is set to 5).

After detecting potential key-frames connected with the newly inserted key-frame k_i, their relative pose is refined via ICP. If ICP fails to converge, then the potential key-frame is discarded. All remaining key-frames are marked as connected with k_i, and an edge is created for each key-frame pair $E(k_i, k_j) \in \mathcal{E}, k_i \in \mathcal{K}, k_i \in \mathcal{K}$ and inserted in the key-frame graph $\mathcal{E}$. The edge of key-frame graph is associated with the index of the two key-frames, as well as with their relative pose $E(i, j) = \{k_i, k_j, \boldsymbol{T}_{i,j}\}$.

3.3. Key-frame Segment Merging

In general, the obtained key-frame pairs will present inconsistencies in terms of the labels associated to the segment appearing in each respective key-frame. To solve this, a global label consistency stage is carried out on the key-frame graph, by matching segment shapes on connected key-frames and merging labels on those segments presenting a significant overlap.

Contrary to the case described in Subsection 2.2.2 where label propagation was performed between two already aligned vertex maps, the pair of key-frames in this case is not aligned to each other. The alignment of the vertex map $\mathcal{V}^m_{k_j}$, the normal map $\mathcal{N}^m_{k_j}$ and the label map $\mathcal{L}^m_{k_j}$ is performed by means of the relative pose of the key-frame pair $\boldsymbol{T}_{i,j} = [\boldsymbol{R}_{i,j}, \boldsymbol{t}_{i,j}]$ such that:

$$\tilde{\mathcal{V}}^m_{k_j}(\tilde{\boldsymbol{u}}) = \boldsymbol{T}_{i,j}\mathcal{V}^m_{k_j}(\boldsymbol{u}) \tag{13}$$

$$\tilde{\mathcal{N}}^m_{k_j}(\tilde{\boldsymbol{u}}) = \boldsymbol{R}_{i,j}\mathcal{N}^m_{k_j}(\boldsymbol{u}) \tag{14}$$

$$\tilde{\mathcal{L}}^m_{k_j}(\tilde{\boldsymbol{u}}) = \mathcal{L}^m_{k_j}(\boldsymbol{u}) \tag{15}$$

where each transformed and reprojected pixel location is obtained as: $\tilde{\boldsymbol{u}} = \pi(\boldsymbol{K}\boldsymbol{T}_{i,j}\mathcal{V}^m_{k_j}(\boldsymbol{u}))$. Then, the same algorithm for label propagation described in Subsection 2.2.2 is applied on the aligned vertex, normal and label maps. Therefore, by applying such procedure for all edges associated to the newly inserted key-frame, the segment labels on all key-frame pairs with overlapping viewing volume are kept globally consistent.

3.4. Key-frame Graph Optimization

Once the key-frame graph is created, the poses of each key-frame are refined via graph optimization [25]. Using the relative key-frame pose $\boldsymbol{T}_{i,j}$ constrained on key-frame edges and the absolute key-frame poses $\boldsymbol{T}_i, \boldsymbol{T}_j$, the following error is minimized:

$$\boldsymbol{E} = \left(\boldsymbol{T}_{i,j}\boldsymbol{T}_j\boldsymbol{T}_i^{-1}\right)^{\top} \Sigma_{i,j}^{-1}\, \boldsymbol{T}_{i,j}\boldsymbol{T}_j\boldsymbol{T}_i^{-1}\,. \tag{16}$$

Here, $\Sigma_{i,j}$ is the uncertainty of relative pose and it is calculated from the Jacobian of the alignment error provided by dense ICP.

After pose optimization, the vertex $\boldsymbol{v}$ and associated normal $\boldsymbol{n}$ of each surfel $\boldsymbol{s}$ are also updated by averaging the effects of the 6DOF increments $\Delta\boldsymbol{T}_n = [\Delta\boldsymbol{R}_n, \Delta\boldsymbol{t}_n]$ across the key-frames:

$$\boldsymbol{v} = \sum_{\boldsymbol{k} \in \mathcal{K}_v} \frac{\Delta\boldsymbol{T}_k\boldsymbol{v}}{|\mathcal{K}_v|}, \quad \boldsymbol{n} = \sum_{\boldsymbol{k} \in \mathcal{K}_v} \frac{\Delta\boldsymbol{R}_k\boldsymbol{n}}{|\mathcal{K}_v|} \tag{17}$$

Figure 7: Flow diagram of the proposed pipeline for failure recovery.

where $\mathcal{K}_v$ is the set of key-frames observing the same surfel $\boldsymbol{s}$. At the end of this process, the pose and associated surfels at each key-frame are globally optimized.

4. Failure recovery

While large-scale reconstruction can typically be achieved via graph optimization and loop closure as explained in the previous Section, another typical limitation of SLAM-based reconstruction approaches is represented by tracking failure. When tracking fails, a specific failure recovery algorithm has to be employed to first detect the failure, then to match the newly acquired sequence with the previously obtained reconstruction.

To recover from tracking failures, we deploy an approach based on place recognition and on matching the entire image of the stored key-frames [24]. Contrary to [24], our method introduces a *temporary map*, used while tracking is lost. After a tracking failure, the reconstructed map is stashed, and a temporary map is initialized with the current depth frame, from which the system restarts performing simultaneous reconstruction and segmentation. At the same time, the system performs failure recovery via key-frame matching: once matching with a previous key-frame is successful, the stashed map is merged with the temporary map, applying label propagation to enforce global segmentation consistency between the two maps. Due to this approach, the system can keep on performing reconstruction and segmentation after tracking failures, guaranteeing long-standing functioning to our framework. The stages carrying out failure recovery are depicted in Fig. 7.

4.1. Failure Detection

Failure recovery is initiated when tracking fails and the current camera pose cannot be retrieved. A tracking failure is detected by thresholding the mean residual between the vertex map $\mathcal{V}_t(\boldsymbol{u})$ on current frame and model vertex map $\mathcal{V}_t^m(\boldsymbol{u})$ with model normal map $\mathcal{N}_t^m(\boldsymbol{u})$ on point to plane error metrics:

$$\sum_{\boldsymbol{u}\in\mathcal{D}_t} \frac{\left|(\mathcal{V}_t(\boldsymbol{u}) - \mathcal{V}_t^m(\boldsymbol{u})) \cdot \mathcal{N}_t^m(\boldsymbol{u})\right|}{|\mathcal{D}_t|} < \tau_{\mathcal{V}} \tag{18}$$

where $|\mathcal{D}_t|$ is the number of point on input vertex map, and $\tau_{\mathcal{N}}$ is the normal angle threshold. when this averaging residual error exceeds a threshold, the tracking is considered to be failed.

Once a tracking failure is detected, the current map $\mathcal{M} = \{\mathcal{S}, \mathcal{L}, \mathcal{K}\}$ – consisting of the global model $\mathcal{S}$, the GSM $\mathcal{L}$ and their set of key-frames $\mathcal{K}$ – is stashed as a *background map* $\mathcal{M}_b = \mathcal{M}, \mathcal{M}_b = \{\mathcal{S}_b, \mathcal{L}_b, \mathcal{K}_b\}$. Then, a *temporary map* is initialized with the current depth frame, and used thereafter as current map.

4.2. Relocalization

After the creation of the temporary map, each new depth frame is matched with the key-frames in the background map. The objective is to recover the camera pose with respect to the coordinate frame of the background map. Similarly to the key-frame matching carried out within the proposed loop closure in Subsection 3.2, we calculate the SAD between the depth components of the vertex map of each key-frame and that of the current frame, and determine the key-frame yielding the minimum SAD, i.e. $\tilde{k}$:

$$\tilde{k} = \arg\min_{k\in\mathcal{K}_b} \sum_{\boldsymbol{u}\in\mathcal{D}_t} \left|\mathcal{V}_t^m(\boldsymbol{u}).z - \mathcal{V}_k^m(\boldsymbol{u}).z\right| . \tag{19}$$

The absolute pose $\boldsymbol{T}_{\tilde{k}}$ associated to $\tilde{k}$ is used as the initial guess of the camera pose with respect to the background map. Then, this pose is refined by computing the relative pose $\boldsymbol{T}(b, \tilde{k})$ via ICP. If the residual error on the estimation of such relative pose is higher than a certain threshold, the estimated pose is considered incorrect and the current frame is discarded. Otherwise, the camera pose of the current frame with respect to the background map is computed as: $\boldsymbol{T}_b = \boldsymbol{T}_{b,\tilde{k}}\boldsymbol{T}_{\tilde{k}}$.

4.3. Temporary Map Merging

After failure recovery is successful, the system needs to merge the temporary map with the background map. Since the camera pose on the temporary map $\boldsymbol{T}_p$ has already been estimated by the SLAM reconstruction stages, the transformation from the coordinate system of the temporary map to that of the background map can be computed as:

$$\boldsymbol{T}_{b,p} = \boldsymbol{T}_b\boldsymbol{T}_p^{-1}, \quad \boldsymbol{T}_{b,p} = [\boldsymbol{R}_{b,p}, \boldsymbol{t}_{b,p}] . \tag{20}$$

Then, the surfel vertex and normal $\boldsymbol{v}_p, \boldsymbol{n}_p \in \mathcal{S}_p$, and key-frame position $\boldsymbol{T}_{k_p} \in \mathcal{K}_p$ on the temporary map are transformed into the coordinate system on the background map and merged together:

$$\tilde{\boldsymbol{v}}_p = \boldsymbol{T}_{b,p}\boldsymbol{v}_p \tag{21}$$

$$\tilde{\boldsymbol{n}}_p = \boldsymbol{R}_{b,p}\boldsymbol{n}_p \tag{22}$$

$$\tilde{\boldsymbol{T}}_{k_p} = \boldsymbol{T}_{b,p}\boldsymbol{T}_{k_p} . \tag{23}$$

While merging, label propagation between overlapping segments among the key-frames of the temporary map and those of the background map is necessary to obtain a globally consistent segmentation. This is done by applying the key-frame label propagation algorithm described in Subsection 3.2.

5. Experimental Results

In this section, we provide quantitative and qualitative experimental results to validate the three main contributions of our method. In Subsection 5.1 we evaluate the simultaneous reconstruction and segmentation framework in terms of computational efficiency and segmentation accuracy, and we demonstrate how it can enable real-time and accurate reconstruction and segmentation of diverse indoor environments. Successively, in Subsection 5.2, we test the proposed loop closure and

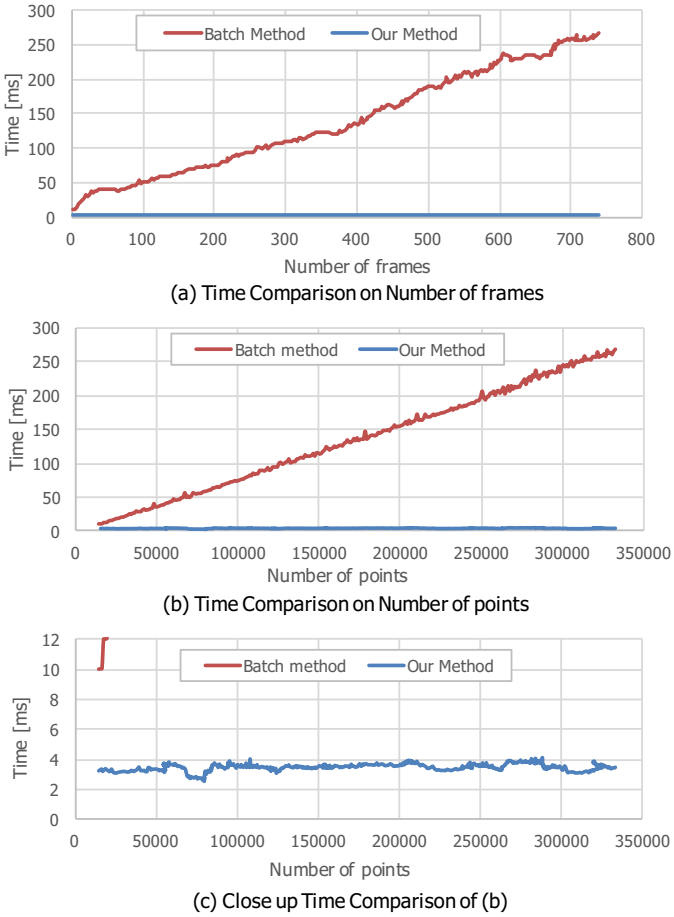

Figure 8: Comparison between the proposed segmentation framework and [13] in terms of measured execution times as a function of, respectively, (a) the number of input frames merged into the SLAM sequence, and (b) the total number of points of the reconstructed scene.

failure recovery approaches on a benchmark dataset designed to include tracking failures and loop closures.

We wish to point out that an implementation of the proposed incremental segmentation framework is publicly available[1].

5.1. Evaluation in terms of efficiency and accuracy

We compare the proposed simultaneous reconstruction and segmentation framework against the graph-based segmentation algorithm of Felzenszwalb and Huttenlocher [13], since it is a widely used segmentation method for reconstructed 3D data [4], whose implementation is publicly available[2]. Since this approach relies on the 3D mesh topology in order to build the segment graph, to test it on our point clouds reconstructed via SLAM, we have employed the fast meshing algorithm proposed in [26], whose implementation is publicly available in the Point Cloud Library (PCL)[3].

For the comparison, we have used one sequence (*fr1/room*) from the public *TUM RGB-D SLAM* benchmark dataset [27].

[1] `campar.in.tum.de/Chair/ProjectInSeg`
[2] `cs.stanford.edu/people/karpathy/discovery`
[3] `www.pointclouds.org`

Table 1: Measured execution times of each stage of the proposed incremental segmentation averaged on the *fr1/room* sequence [27] at different resolutions of the input frames (all stages are implemented on CPU)

Number of frames	739	739	739
Resolution of depth map	160×120	320×240	640×480
Depth Map Segmentation [ms]	1.81	7.63	32.39
Segment Label Propagation [ms]	1.5	5.58	29.2
Segment Merging [ms]	0.08	0.09	0.11
Segment Update [ms]	0.13	0.51	1.92
Segmentation Total [ms]	3.52	13.82	63.58
Segmentation Frame-Rate [fps]	284	72	15

Figure 9: Reconstruction results with texture at different resolutions of the input depth map (from left to right: 160×120, 320×240, 640×480 pixel resolution).

Both algorithms have been tested on the same platforms, a standard laptop PC equipped with an Intel Core i7 CPU at 2.6GHz with 16GB of RAM. Both algorithms run totally on CPU processing (no GPU optimization of any part). The resolution of the input depth maps is converted to 160x120 to allow the dense SLAM framework employed to run in real-time on such CPU.

Fig. 8 shows the comparison in terms of efficiency between the two algorithms. For both algorithms, only the time required for segmentation is measured, while the time spent on SLAM is not taken into account. In the Figure, the measured execution times of both methods are plotted as a function of, respectively, the number of input frames merged into the SLAM sequence (Fig. 8,a), and (the total number of points of the reconstructed scene (Fig. 8, b), so to compare the two methods in terms of scalability with the size of reconstructed point clouds. As shown in the Figure, while [13] exhibits a clearly linear complexity with the point cloud size, the proposed algorithm demonstrates a constant complexity with such size, thanks to the incremental nature of its approach. Also, the proposed method demonstrates real-time capabilities, running at an average of 3.5 ms per frame.

Also, we wish to point out here the advantage with respect to the method in [17], which, as mentioned in Sec. 1, also performs incremental segmentation of 3D reconstructions. As stated in Sect. IV of [17], the complexity of this method grows on the size of the map, and its run-time exhibits a linear dependency with the number of points of the 3D reconstruction, as reported by the results in Fig. 5 in [17]. Our method, instead, due to the properties outlined in Subsection 2.2, has a constant run-time with respect to the global model size, as shown in Fig. 8.

The execution time measured relatively to each stage involved in our incremental segmentation, averaged over the benchmark *fr1/room* sequence, is summarized in Table 1, while

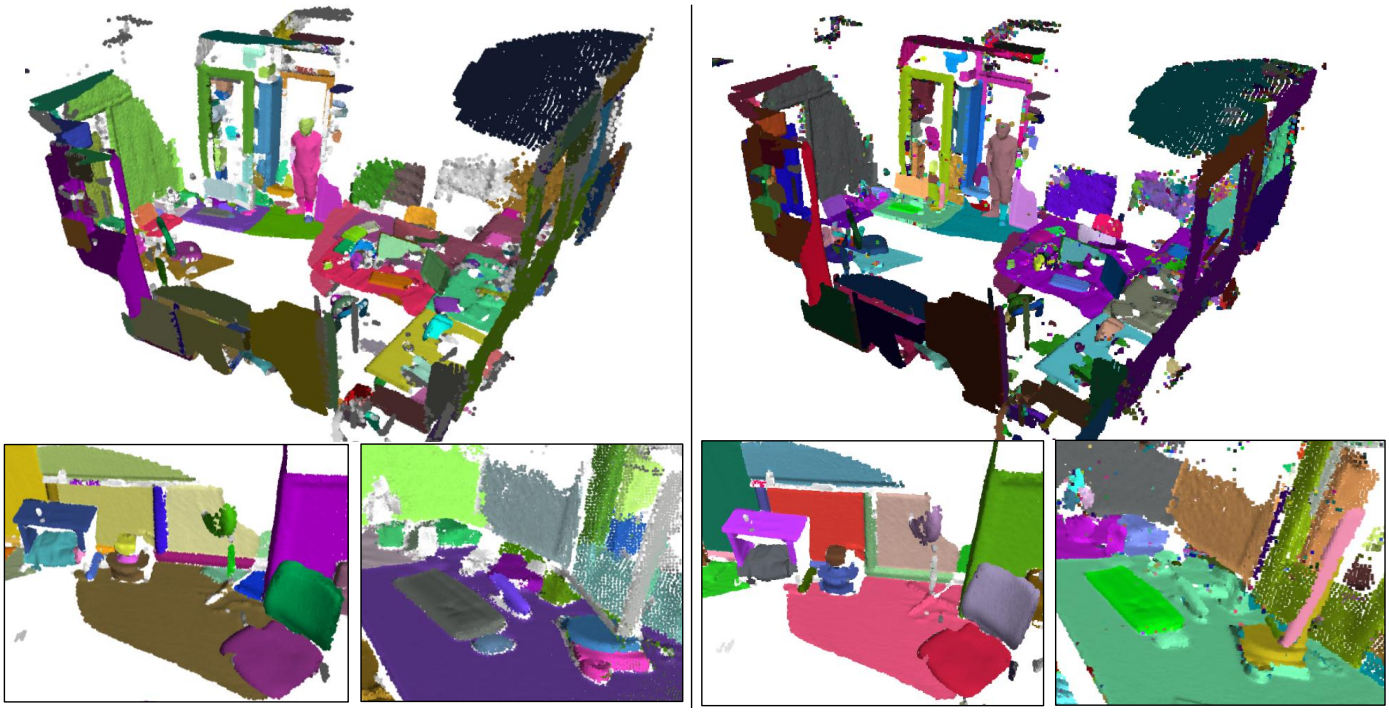

Figure 10: Qualitative comparison between the segmentation obtained by the proposed method (left) and that obtained by the method in [13] (right).

the final reconstruction result on each input resolution is shown in Fig. 9, and the final segmentation of the whole sequence is shown in Fig. 10.

As shown by the table, the whole segmentation framework can run in real-time at different resolutions of the input depth map, yielding 284 frame per second on the 160×120 resolution, and running at 72 and 15 frames per second at a resolution of, respectively, 320×240 and 640×480. We expect that even higher frame rates could be achieved if the segmentation stages were to be processed on GPU. Also interestingly, the Table shows that the most time consuming stage is the initial frame-wise segmentation step: hence, an even higher efficiency can be achieved by plugging in a different, more efficient frame-wise segmentation algorithm with respect to the one currently being deployed.

As for the reconstruction quality, we wish to point out that different resolutions of the input depth map simply affect the density and detail of the reconstructed point cloud, as shown by the Fig. 9. Hence, the input resolution should be chosen so as to satisfy the required trade-off between expected level of details of the final reconstruction and computational cost.

A snapshot of the results of reconstruction and segmentation obtained with our method, together with the ground truth, are shown in Fig. 11. The obtained weighted average overlap scores and unweighted one are shown in Table 2.

In addition to previous results, we also compare our method with [13] in terms of the segmentation accuracy of the same benchmark sequence, i.e. *fr1/room* from the *TUM RGB-D SLAM* benchmark. Fig. 10 shows a qualitative comparison in terms of segmentation yielded by the proposed framework (left) and that given by [13] (right). Overall, while the segmentation accuracy appears to be comparable, the proposed method seems able to better segment small objects on top of planar surfaces, as witnessed also by the right close-up snapshot shown at the bottom of the Figure.

Additionally, we show some quantitative results in terms of segmentation accuracy with human annotated ground truth of labeled point clouds. Specifically, we used 14 sequences from the public *RGB-D Scenes Dataset v2* [32]. The dataset provides manually labeled point clouds with RGB-D image sequences, although the ground truth segmentation relates to semantic categories (i.e., semantic segmentation). To provide quantitative results, we calculated a weighted average overlap score between our segmented point clouds and the ground truth, which is an extended version of the area-weighted average overlap score proposed in [33].

The weighted average overlap score for a reconstructed point cloud is computed as

$$Score_{weighted} = \frac{\sum_i |G_i| \max_j \text{Overlap}\left(G_i, S_j\right)}{\sum_i |G_i|} \tag{24}$$

where G_i is the ith ground truth segment, $|G_i|$ is the number of points of G_i, S_j is the jth segment from our incremental segmentation, and $\text{Overlap}\left(G_i, S_j\right) = \frac{G_i \cap S_j}{G_i \cup S_j}$. The overlap between the reconstructed point clouds and the ground truth is calculated using a fast Nearest Neighbor Search method such as a kd-tree. Moreover, we compute also the unweighted average overlap score as:

$$Score_{unweighted} = \sum_i \frac{\max_j \text{Overlap}\left(G_i, S_j\right)}{N_g} \tag{25}$$

Table 2: Weighted and unweighted average overlap scores yielded by our approach on the 14 sequences of the public *RGB-D Scenes Dataset v2* [32]

Scene	01	02	03	04	05	06	07	08	09	10	11	12	13	14	Total
Weighted score [%]	54.4	52.9	55.4	51.7	78.1	77.6	83.9	83.5	63.8	62.1	59.0	60.2	65.9	67.5	65.4
Unweighted score [%]	70.0	69.1	70.0	68.1	85.7	67.0	87.0	77.0	80.4	75.2	66.9	65.0	87.3	80.8	74.9

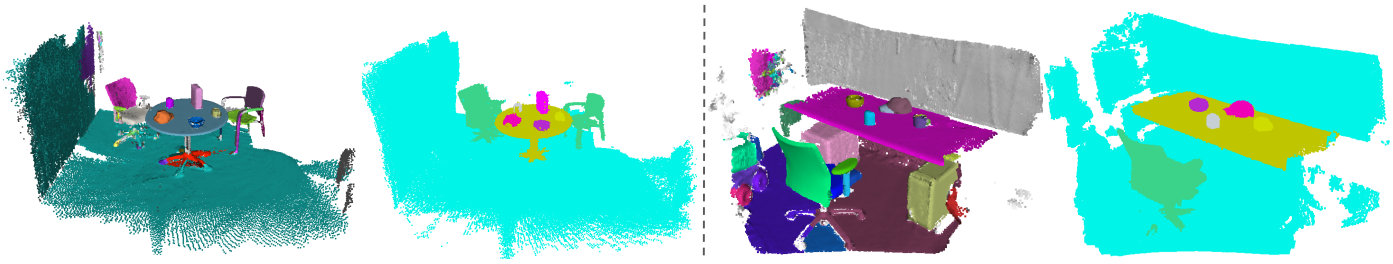

Figure 11: Results of reconstruction and segmentation obtained with our method and compared to the ground truth on the public *RGB-D Scenes Dataset v2* [32]. From left to right: our result and the ground truth on the sequence of (*scene_07*) , our result and the ground truth on the sequence of (*scene_14*).

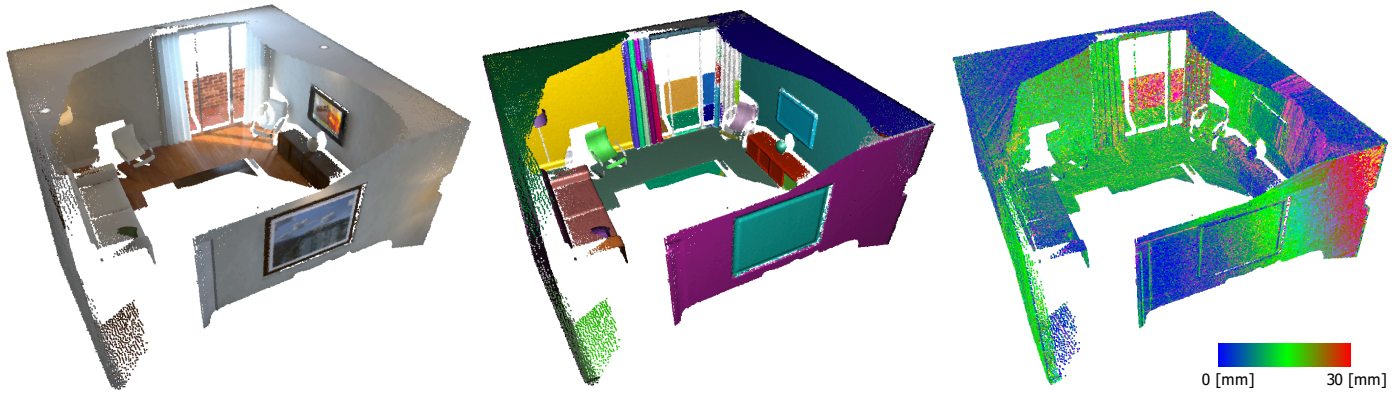

Figure 12: Reconstruction and segmentation result on one sequence of the *ICL-NUIM* dataset [30]. From left to right: reconstruction result with texture, segmentation result, point-wise reconstruction error (ranging from 0 mm(blue) to 30 mm (red)).

where N_g is the number of ground truth segments.

Two examples of the result of reconstruction and segmentation obtained with our method, together with the ground truth, are shown in Fig. 11. The obtained weighted and unweighted average overlap scores over all 14 dataset sequences are shown in Table 2. Overall, we can notice that, although providing an accurate segmentation, our method tends to lean towards over-segmentation, mostly because our approach detains geometrically connected 3D components, in contrast with the ground truth which defines segments as semantically connected components.

Moreover, we show a quantitative result in terms of reconstruction accuracy on a synthetic dataset. Specifically, we used one sequence (*lr kt1 with noise*) from the public *ICL-NUIM* dataset [30]. The sequence is made by rendering a mesh model with adding a synthetic depth noise. The ground truth model is also provided. Results are reported in Fig. 12, where the left image shows the reconstruction result with texture, the middle image shows the segmentation result and the right image shows the reconstruction error on each point compared to the ground truth. Notably, the mean point-wise reconstruction error on the whole point cloud is 9.73 mm.

Finally, we show some qualitative results of the segmentation reported by our method for reconstructed indoor environments acquired with our own setup based on a PrimeSense Carmine 1.09 RGB-D sensor. Three examples are shown in Figure 13 together with zoomed-in details, showing, from left to right, a kitchen-like scenario, a table-top scenario and a large scale reconstruction across two rooms.

5.2. *Evaluation of loop closure and failure recovery*

We evaluate here the capability of the proposed loop closure and failure recovery approaches to provide large-scale and long-standing reconstructions with globally consistent segmentation. To this aim, we have selected a subset of sequences from three benchmark datasets that include loop closures and independent sequences from the same scene: the *7Scenes* dataset[29], the *3D Scene* dataset[28] and the *ICL-NUIM* dataset [30].

To evaluate the proposed loop closure with global label consistency, we used the sequence named *Copy room* from *3D Scene* dataset[28]. The result is shown in Fig. 14, where the loop closure is highlighted with a red rectangle. The top image

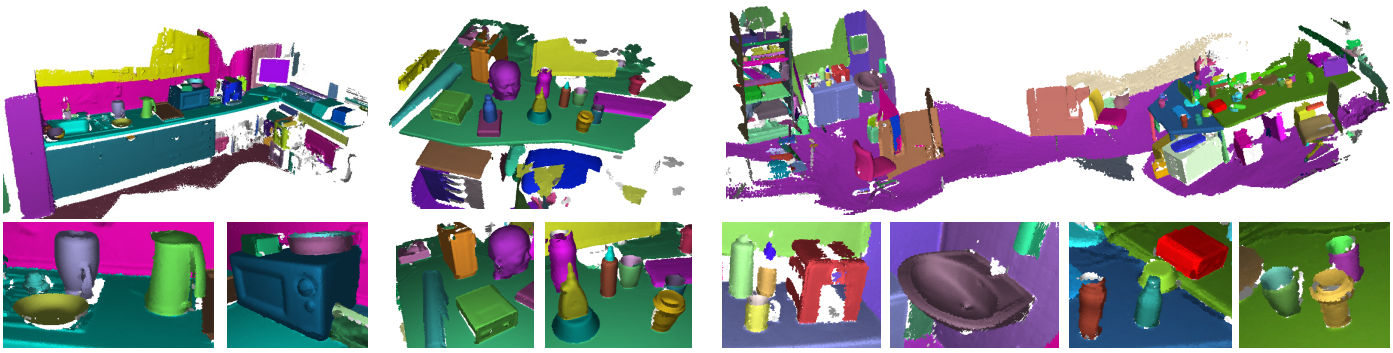

Figure 13: Reconstruction and segmentation yielded by the proposed framework relatively to, from left to right: a kitchen-like scenario, a table-top scenario, a large scale reconstruction across two rooms. Bottom row reports zoomed-in segmentation details of each scene.

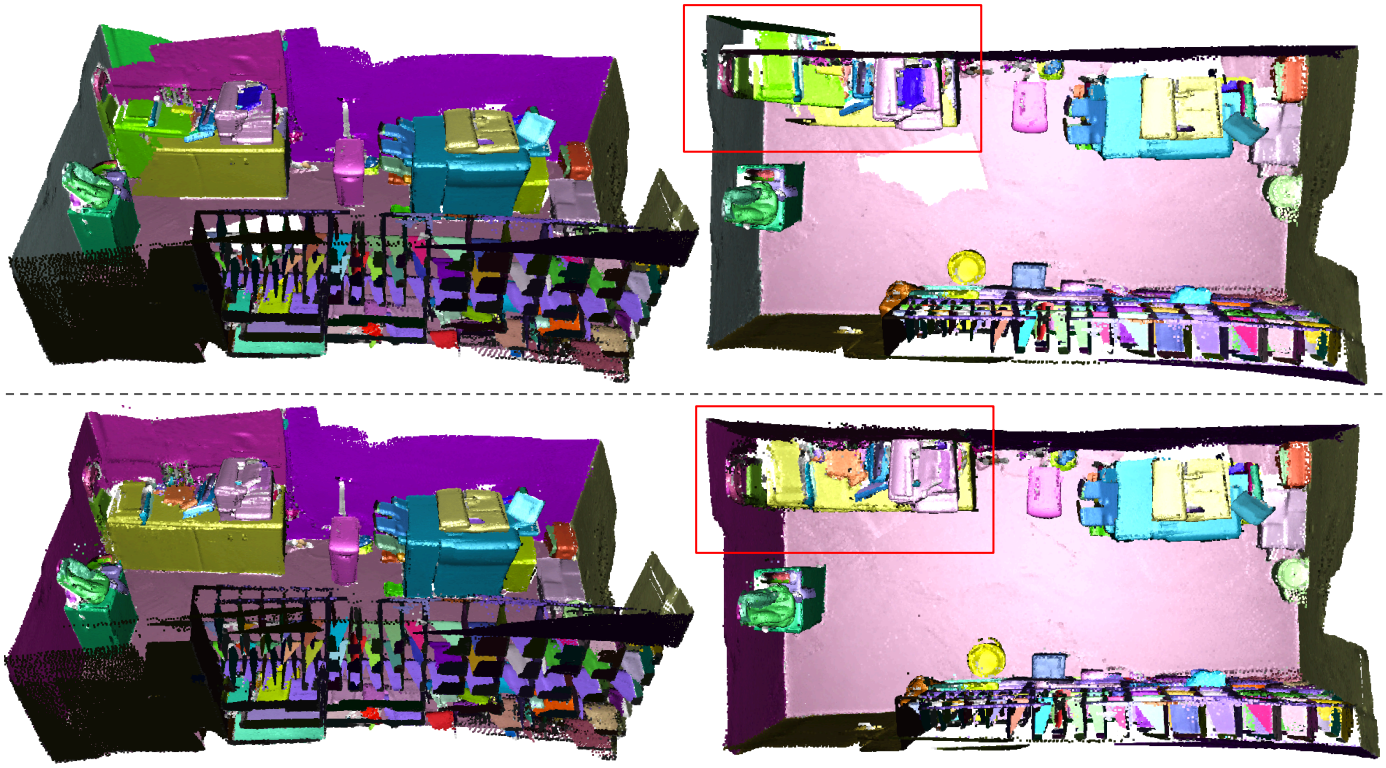

Figure 14: The reconstruction and segmentation result obtained by our method before (top) and after the proposed loop closure with global label consistency (bottom) on the (*Copy room*) sequence from the public *3D Scene* benchmark dataset [28]. The loop closure is highlighted with a red rectangle.

and bottom images show the result, respectively, before and after loop closure. As it can be seen, the proposed approach is able to obtain global consistency among the segment labels, as well as in terms of point cloud geometry.

In addition, to evaluate the proposed failure recovery algorithm, we used 2 scenes (*fire*, *office*), containining four and ten sequences respectively, from the public *7Scenes* benchmark dataset [29]. The dataset on each scene consists of multiple independent sequences acquired within the same environment but including different viewpoints. Also, the sequences include significant viewpoint jumps when the sequence changes, simulating potential tracking failures. The results of failure recovery on the *office* scene is shown in Fig. 1. The top right image shows the reconstructed map relatively to two independent sequences (i.e., before and after tracking is lost), while the bottom right image shows the merged result by the proposed approach. In the figure, gray points denote the key-frames, while blue lines are key-frame edges. Thanks to relocalization and label propagation based on graph optimization, globally consistency among segment labels is enforced before and after tracking failure. The overall results are shown in Fig. 15. The left part in the figure shows the results on the *fire* scene, and the right part shows the results on the *office* scene. On each scene, The center image (bigger one) represents the final map merging all independent sequences, which are respectively shown in the smaller images. The final map demonstrates that the proposed failure recovery algorithm is able to withstand several tracking failures without label contamination.

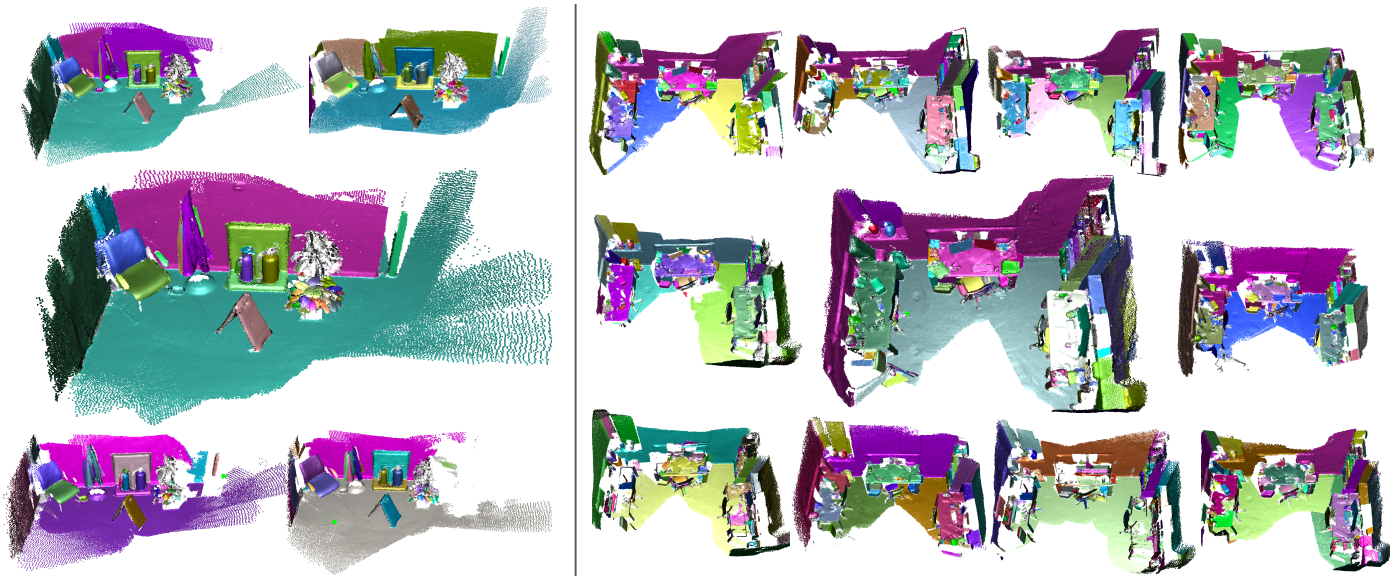

Figure 15: Reconstruction and segmentation results on the *fire* (left) and *office* (right) sequences, both from the *7Scenes* benchmark dataset [29]. In both figures, the central image is the final map obtained by merging all independent sequences via the proposed failure recovery approach, while smaller images show the result from each independent sequence from the same scene.

Additional material concerning the reconstructions obtained on the presented benchmark sequences, as well as further results on real data acquired by our own is available in the supplementary material attached with this submission.

6. Conclusion

We have proposed a framework for simultaneous reconstruction and segmentation, where segments obtained from each depth frame are incrementally merged within a unified segmentation model of the scene built on top of the SLAM reconstruction. Conversely to available methods in literature, our method demonstrated the capability of processing frames in real-time, and to scale advantageously with the size of the reconstructed point cloud. Moreover, we proposed an approach based on loop closure and failure recovery that makes our framework able to cope with large scale and long standing reconstructions while maintaining the segmentation globally consistent.

We believe that large scale and long standing segmentation of reconstructed environments can pave the way to new directions in the field of robotic perception, computer vision and scene understanding. A few examples are represented by real-time object discovery in unknown environments (currently done offline [4]), real-time object recognition from fully-3D point clouds (currently done on single view only [1]), real-time 3D tracking and augumented reality on fully-3D point clouds.

References

[1] A. Aldoma, F. Tombari, J. Prankl, A. Richtsfeld, L. D. Stefano, M. Vincze, Multimodal cue integration through hypotheses verification for rgb-d object recognition and 6dof pose estimation, in: Proc. Int. Conf. on Robotics and Automation (ICRA), 2013.

[2] J. Kenney, T. Buckley, O. Brock, Interactive segmentation for manipulation in unstructured environments, in: Proc. Int. Conf. on Robotics and Automation (ICRA), 2009.

[3] T. Mörwald, M. Zillich, M. Vincze, Edge tracking of textured objects with a recursive particle filter, in: Proc. of the Graphicon, 2009.

[4] A. Karpathy, S. Miller, L. Fei-Fei, Object discovery in 3D scenes via shape analysis, in: Proc. Int. Conf. on Robotics and Automation (ICRA), 2013.

[5] A. Uckermann, R. Haschke, H. Ritter, Realtime 3D segmentation for human-robot interaction, in: Int. Conf. on Intelligent Robots and Systems, 2013.

[6] A. Uckermann, C. Elbrechter, R. Haschke, H. Ritter, 3D scene segmentation for autonomous robot grasping, in: Int. Conf. on Intelligent Robots and Systems, 2012.

[7] A. Pieropan, H. Kjellstrom, Unsupervised object exploration using context, in: Int. Symp. on Robot and Human Interactive Communication, 2014.

[8] A. Abramov, K. Pauwels, J. Papon, F. Worgotter, B. Dellen, Depth-supported real-time video segmentation with the kinect, in: Applications of Computer Vision (WACV), 2012.

[9] R. a. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, A. Fitzgibbon, KinectFusion: Real-time dense surface mapping and tracking, in: Int. Symp. on Mixed and Augmented Reality (ISMAR), 2011.

[10] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. J. Leonard, J. Mcdonald, Real-time large scale dense RGB-D SLAM with volumetric fusion, in: Intl. J. of Robotics Research, 2015.

[11] P. Henry, M. Krainin, E. Herbst, X. Ren, D. Fox, RGB-D Mapping : Using Depth Cameras for Dense 3D Modeling of Indoor Environments, in: Int. J. of Robotics Research, 2012.

[12] C. Kerl, J. Sturm, D. Cremers, Dense visual SLAM for RGB-D cameras, in: Int. Conf. on Intelligent Robots and Systems, 2013.

[13] P. F. Felzenszwalb, D. P. Huttenlocher, Efficient Graph-Based Image Segmentation, in: Int. J. of Computer Vision, 2004.

[14] A. Golovinskiy, T. Funkhouser, Min-cut based segmentation of point clouds, in: Int. Conf. on Computer Vision Workshops, ICCV Workshops 2009.

[15] T. Rabbani, F. A. van den Heuvel, G. Vosselman, Segmentation of point clouds using smoothness constraint, in: International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences - Commission V Symposium 'Image Engineering and Vision Metrology', 2006.

[16] J. Strom, A. Richardson, E. Olson, Graph-based segmentation for col-

ored 3D laser point clouds, in: Proc. Int. Conf. on Intelligent Robots and Systems (IROS), 2010.
[17] R. Finman, T. Whelan, M. Kaess, J. J. Leonard, Efficient Incremental Map Segmentation in Dense RGB-D Maps, in: Proc. Int. Conf. on Robotics and Automation (ICRA), 2014.
[18] R. Salas-Moreno, B. Glocken, Dense planar SLAM, in: Int. Symp. on Mixed and Augmented Reality (ISMAR), 2014.
[19] R. Finman, P. L, J. J. Leonard, Toward Object-based Place Recognition in Dense RGB-D Maps, in: ICRA workshop on visual place recognition in changing environments, 2015.
[20] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, A. Kolb, Real-Time 3D Reconstruction in Dynamic Scenes Using Point-Based Fusion, in: Int. Conf. on 3D Vision (3DV), 2013.
[21] C. Tomasi, R. Manduchi, Bilateral filtering for gray and color images, in: Int. Conf. on Computer Vision (ICCV), 1998.
[22] C. V. Nguyen, S. Izadi, D. Lovell, Modeling kinect sensor noise for improved 3D reconstruction and tracking, in: Proc. 2nd Joint 3DIM/3DPVT Conference: 3D Imaging, Modeling, Processing, Visualization and Transmission, (3DIMPVT), 2012.
[23] J. Engel, T. Schps, D. Cremers, LSD-SLAM: Large-Scale Direct Monocular SLAM, in: European Conference on Computer Vision (ECCV), 2014.
[24] G. Klein, D. Murray, Improving the agility of keyframe- based SLAM, in: European Conference on Computer Vision (ECCV), 2008.
[25] R. Kuemmerle, G. Grisetti, H. Strasdat, K. Konolige, W. Burgard, g2o: A General Framework for Graph Optimization, in: Int. Conf. on Robotics and Automation (ICRA), 2011.
[26] Z. Marton, R. Rusu, M. Beetz, On fast surface reconstruction methods for large and noisy point clouds, in: Proc. Int. Conf. on Robotics and Automation (ICRA), 2009.
[27] J. Sturm, N. Engelhard, F. Endres, W. Burgard, D. Cremers, A benchmark for the evaluation of RGB-D SLAM systems, in: Int. Conf. on Intelligent Robots and Systems, 2012.
[28] Q. Y. Zhou, V. Koltun, Dense Scene Reconstruction with Points of Interest, in: ACM Transactions on Graphics, 2013.
[29] B. Glocker, J. Shotton, A. Criminisi, S. Izadi, Real-Time RGB-D Camera Relocalization via Randomized Ferns for Keyframe Encoding, in: IEEE Trans. on Visualization and Computer Graphics, 2013.
[30] A. Handa, T. Whelan, J. McDonald, A. Davison, A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM, in: IEEE Intl. Conf. on Robotics and Automation (ICRA), 2014.
[31] S. C. Stein, M. Schoeler, J. Papon and F. Wrgtter, Object Partitioning Using Local Convexity, in: IEEE Conf. on Computer Vision and Pattern Recognition, 2014.
[32] K. Lai, L. Bo, D. Fox, Unsupervised Feature Learning for 3D Scene Labeling, in: Int. Conf. on Robotics and Automation (ICRA), 2014.
[33] D. Hoiem, A. A. Efros, M. Hebert, Recovering Occlusion Boundaries from an Image, in: Int. J. Comput. Vision, 2011.