Real-Time Large-Scale Dense 3D Reconstruction with Loop Closure

Olaf Kähler, Victor A. Prisacariu(✉), and David W. Murray

Department of Engineering Science, University of Oxford, Oxford, UK
{olaf,victor,dwm}@robots.ox.ac.uk

Abstract. In the highly active research field of dense 3D reconstruction and modelling, loop closure is still a largely unsolved problem. While a number of previous works show how to accumulate keyframes, globally optimize their pose on closure, and compute a dense 3D model as a post-processing step, in this paper we propose an online framework which delivers a consistent 3D model to the user in real time. This is achieved by splitting the scene into submaps, and adjusting the poses of the submaps as and when required. We present a novel technique for accumulating relative pose constraints between the submaps at very little computational cost, and demonstrate how to maintain a lightweight, scalable global optimization of submap poses. In contrast to previous works, the number of submaps grows with the observed 3D scene surface, rather than with time. In addition to loop closure, the paper incorporates relocalization and provides a novel way of assessing tracking quality.

1 Introduction

The prompt delivery of highly detailed dense reconstructions of the 3D environment is currently one of the major frontiers in computer vision [1–6]. While off-line dense reconstruction and real-time sparse reconstruction are quite mature (e.g. [7–10]) it is only recently that real-time dense modelling and reconstruction have become feasible. There are two obvious enablers and catalysts. The first is the release of affordable RGB-D sensors such as the Kinect, and the second is the opening up of graphics processing units for general purpose computing.

As the area develops, one can discern the research focus shifting from basic processing to obtain *any* reconstruction to more refined aspects of 3D representation and modelling. Earlier works such as [2,11,12] have firmly established volumetric representations as a powerful approach. Their memory requirements have been reduced by an order of magnitude more recently [13–17], rendering them attractive for large-scale mapping. While the competing surfel based representations (e.g. [18]) never had this issue, two common problems arising in all large scale mapping tasks are those of tracking drift and loop closure.

Electronic supplementary material The online version of this chapter (doi:10.1007/978-3-319-46484-8_30) contains supplementary material, which is available to authorized users.

B. Leibe et al. (Eds.): ECCV 2016, Part VIII, LNCS 9912, pp. 500–516, 2016.
DOI: 10.1007/978-3-319-46484-8_30

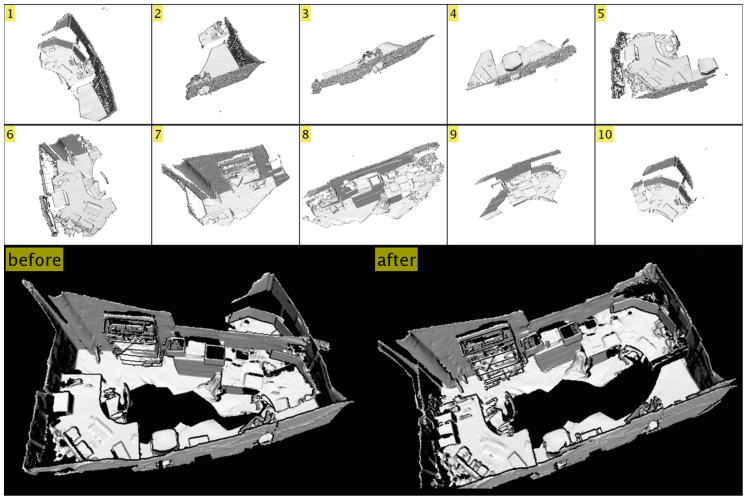


Fig. 1. Example reconstruction. The last row shows the results before and after loop closure, with images 1–10 showing constituent submaps.

Among the current methods addressing loop closure for dense 3D reconstruction, a first group [19–22] selects a subset of keyframes and gathers constraints between them to solve the alignment problem, and only then considers dense reconstruction as a post-processing step. For this group, dense reconstruction typically has to be repeated from scratch whenever there are changes to the graph of keyframes. Rather than using keyframes, a second approach to loop closure is to accumulate constraints continuously and integrate them into the 3D model by deforming the reconstruction [18,23]. However, the relatively high computational complexity of repeatedly deforming the entire dense model leads to poor scalability. A third line of works employs submaps [24–26] to enable loop closure. The idea is to accumulate the information from every single camera image into various overlapping submaps, and to adjust the position and orientation of these submaps as relative constraints between them become available.

In the present work we pick up this idea of submaps, but extend it significantly. First, by gathering the relative constraints in a much more efficient manner, we develop an online system running in real time rather than minutes [25] or hours [26]. Second, we propose a novel way of delineating the submaps from each other, leading to memory growth that scales with the size of the represented 3D scene surface rather than with the sequence length [25,26] or trajectory length [24]. Third, we present a method of fusing the individual submaps into a global model online and on the fly. As enabling tools for the overall pipeline, we also introduce a novel way of assessing the camera tracking quality in dense image alignment and introduce camera relocalisation to the whole pipeline. All of these combine to yield a system which allows interruption of the map building process

and resumption later at an arbitrary position, hence extending and growing the map on demand without requiring a carefully pre-planned trajectory.

A reconstruction obtained by our system is shown in Fig. 1. We focus this paper on the 3D modelling methods and assume that dense depth maps are available as input, be it from monocular sequences [11,12], stereo, or RGB-D sensors. We also rely on volumetric representations, but we believe that many of the ideas easily transfer to a much wider range of applications.

1.1 Related Work

The KinectFusion system [2] has spurred a range of further research into efficient dense 3D mapping. Central to most of these efforts is a 3D volumetric scene representation using a truncated signed distance function (T-SDF) [27]. The same representation has been exploited in systems relying exclusively on RGB cameras without depth sensing [11,12]. However, a fundamental limitation of the original KinectFusion system is its memory complexity, which grows with the size of the represented volume rather than with that of the represented surface, rendering large scale reconstruction and loop closure infeasible. This has been overcome in later works by storing only sparse blocks of the T-SDF around the actual surface and addressing the blocks via tree structures [13–15] and hash functions [16,17]. Our work in particular relies on the highly efficient implementation of voxel block hashing given in [17].

However, such large scale reconstruction quickly shows the drawbacks of the separate handling of tracking and reconstruction in dense methods: tracking drift becomes significant before loop closure. Drift can be reduced by additional sensing such as IMU measurements [28,29]. Nevertheless, drift can not be completely avoided and a more substantial change to the representation is required to ensure earlier tracking errors can be tolerated or even corrected by later measurements.

One approach to this issue is to reduce the recorded image sequence to a number of carefully selected keyframes, which are stored along with their 3D poses. Upon loop closure, the accumulated error is distributed by applying pose graph optimisation [30]. In some implementations (e.g. [20]), information from those frames not selected as keyframes is lost to the 3D modelling process, but other methods try to accumulate the non-keyframe information by improving or filtering the keyframe measurements [19,21,22]. In both cases, the keyframes only store 2.5D information, which is precisely what makes these approaches memory efficient and feasible at large scale. To provide the user with feedback, or potentially a robot with a global map for path planning, the local 2.5D maps have to be merged to a globally consistent 3D map. In the above works this step is not detailed, but the underlying assumption is that this can be achieved by fusing the individual keyframes after their final poses have been computed.

Quite the opposite is done in the map-centric representation used in Kintinuous 2.0 [31], ElasticFusion [18] and an earlier work along the same lines [23]. In these systems, the 3D world is deformed globally whenever loop closure requires such changes. However, another problem of dense mapping becomes obvious with this strategy: the deformation has to be applied to the whole 3D model over and

over again and, given the sheer amount of data provided by large scale dense reconstructions, the computational cost becomes significant.

In the works of [24–26] another alternative is picked up, that of local submaps. Submaps date back to much earlier work in SLAM (e.g. [32–34]), but have only recently been considered in the context of dense 3D modelling. In contrast to a single global map, such as that used in KinectFusion, the local submaps can be shuffled around to accommodate loop closure errors. Unlike in the keyframe based methods mentioned earlier, submaps provide a convenient way of accumulating measurements from all intermediate frames, and furthermore they allow "banking" of the computational effort required to fuse this information. In contrast to the global deformations performed by ElasticFusion, the piecewise rigid transformations scale much more readily to large 3D models.

However, some fundamental questions have to be addressed to realize submapping approaches. First, the map has to be split into submaps, and while in [24] a classical threshold on camera motion is used, both lines of research in [25,26] split the image sequence by time, coincidentally choosing to spawn a new submap every 50 frames. While these subdivision schemes are easy to implement, they will scale poorly in case of real world exploratory trajectories. In our work we therefore propose a novel way of delineating submaps based on the part of the global 3D map that is observed in the images, and spawn new submaps only when the camera *view* moves on to explore new parts of the 3D scene.

Second, relative constraints between the submaps have to be evaluated, and all of [24–26] achieve this using a relatively costly alignment of the 3D content. For example, [25] uses a global optimisation operating on full 3D volumes. As such, when a new subvolume is added to the optimisation, the zero-level set and 3D normals need to be extracted, correspondences between volumes need to be found, and a large ICP optimisation has to be run. In contrast, by tracking the camera relative to multiple submaps simultaneously and by exploiting that the resulting trajectory should be unique, we gain the relative constraints "for free" as a side product of normal pose estimation and tracking operations. The computational overhead is therefore reduced to (i) one extra per-frame alignment between a depth map and a local scene, taking a few extra ms each frame, and (ii) a much simplified global optimisation to estimate submap locations working only on 3D poses, without needing any further surface alignments. Overall, the previous works based on submaps aim at offline operation, whereas we very much aim at a real-time system that provides the user with immediate feedback, similar to that which ElasticFusion [18] achieves at smaller scale.

1.2 Outline

The rest of the paper is ordered as follows. We revise the background and basic technologies underlying our system in Sect. 2. The detail of our core contributions is provided in Sect. 3, and an evaluation follows in Sect. 4. We summarise and draw conclusions in Sect. 5.

2 Basic System

Our system is built on the principles behind KinectFusion [2] and voxel block hashing [16]. More specifically, we use the highly efficient implementation of both provided by InfiniTAM [17], the source code of which is provided online. Our contribution as presented in Sect. 3 can readily transfer to alternative systems.

In Sect. 2.1 we revise the camera tracking and integration of depth images from [17]. As a first contribution, we present a method of evaluating tracking quality in Sect. 2.2, which is key for our further work. We also discuss a novel method for splitting a large scene into submaps. Since this is closely related to the representation in our base system, we present this in Sect. 2.3.

2.1 Integration of Depth Maps

We represent the scene in our map as a truncated signed distance function (T-SDF) denoted as $F(\mathbf{X})$. For each 3D point $\mathbf{X}$ within a truncation band $[-\mu \dots \mu]$, the function F returns a tuple $(d, w, \mathbf{c})$, where d denotes the distance of the point $\mathbf{X}$ to the nearest surface, w the number of accumulated observations and (optionally) $\mathbf{c}$ an estimate of the colour. Outside the truncation band, $w = 0$ and d and $\mathbf{c}$ are undefined. The function F is stored volumetrically, sampled on a voxel grid with fixed resolution s, and typically $s = 5\,\text{mm}$ or $10\,\text{mm}$. For voxel block hashing [16,17], the volume is divided into small blocks of $8 \times 8 \times 8$ voxels each, which are allocated sparsely to cover the truncation band $\pm\mu$ around the surface. To find the corresponding block for a point $\mathbf{X}$, a hash function is applied, providing an almost constant time lookup of the tuples $(d, w, \mathbf{c})$.

Each incoming image is processed in three steps: tracking, integration and raycasting. In the tracking step, we estimate the camera pose relative to the model, in the integration step we fuse the depth image into the T-SDF, and lastly we extract the surface from the implicit representation using raycasting.

Our tracking step is based on the Iterative Closest Points algorithm with projective data association [2]. Using the pre-calibrated intrinsic camera parameters $\mathbf{K}$ we first compute a 2D map of 3D points from each depth image D_t:

$$\mathcal{P}(\mathbf{x}) = D_t(\mathbf{x})\mathbf{K}^{-1} \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix}, \tag{1}$$

At each time t, we now want to estimate the camera pose $\mathbf{T}_t = (\mathbf{R}_t, \mathbf{t}_t)$ for the depth image, consisting of rotation $\mathbf{R}_t$ and translation $\mathbf{t}_t$. As additional input we require a 2D map of 3D points $\mathcal{V}_{t-1}(\mathbf{x})$ and surface normals $\mathcal{N}_{t-1}(\mathbf{x})$ as extracted from the T-SDF for a known pose $\mathbf{T}_{t-1}$. We compute $\mathbf{T}_t$ minimising:

$$\epsilon_{\text{ICP}}(\mathbf{T}_t) = \sum_{\mathbf{x}} \rho \left(\left(\mathbf{T}_t^{-1} \mathcal{P}(\mathbf{x}) - \mathcal{V}_{t-1}(\bar{\mathcal{P}}(\mathbf{x})) \right)^T \mathcal{N}_{t-1}(\bar{\mathcal{P}}(\mathbf{x})) \right), \tag{2}$$

where ρ is a robust error norm and $\bar{\mathcal{P}}(\mathbf{x}) = \pi(\mathbf{K}\mathbf{T}_{t-1}\mathbf{T}_t^{-1}\mathcal{P}(\mathbf{x}))$ is the reprojection of $\mathcal{P}(\mathbf{x})$ into the frame of the provided reference maps $\mathcal{V}$ and $\mathcal{N}$. Using

the well known local frame representation of rotations, this function is linearized and minimised using the Levenberg-Marquardt optimisation algorithm.

To integrate D_t into the T-SDF we follow [17]. For the set of points $\left\{\mathbf{T}_t^{-1}\mathcal{P}(\mathbf{x})\right\}$ and the truncation band $\pm\mu$ around them, we allocate the corresponding voxel blocks in the hash table and declare them part of the visible set $\mathbb{V}$. Then we project all voxels $\mathbf{X}$ in the currently visible voxel blocks into the depth and, optionally, colour images. The tuples $(d, w, \mathbf{c})$ are updated as:

$$d \leftarrow \frac{wd + d^*}{w+1}, \qquad \mathbf{c} \leftarrow \frac{w\mathbf{c} + \mathbf{c}^*}{w+1}, \qquad w \leftarrow w+1, \tag{3}$$

with $d^* = D_t(\pi(\mathbf{K}\mathbf{T}_t\mathbf{X})) - [\mathbf{T}_t\mathbf{X}]_{(z)}$ clamped to $[-\mu \ldots \mu]$ and, if a colour image C_t is available and desired, $\mathbf{c}^* = C_t(\pi(\mathbf{K}\mathbf{T}_t\mathbf{X}))$.

Finally the vertex and normal maps of the surface $\mathcal{V}$ and $\mathcal{N}$ are obtained from the T-SDF by raycasting. After forward projecting bounding boxes for all visible voxel blocks to obtain a valid search range, a ray is cast for each pixel $\mathbf{x}$ in $\mathcal{V}$ to find the zero level set of the T-SDF [17]. Iteratively reading the d values from F, steps are taken until the zero crossing $d = 0$ is found and $\mathcal{V}$ is set to the corresponding 3D point. The field of normals $\mathcal{N}$ is then obtained by computing the cross product between neighbouring pixels in $\mathcal{V}$.

2.2 Evaluation of Tracking Accuracy

Camera tracking can fail, and sometimes does. Many current systems, in particular [17], do not explicitly manage this problem. Detecting tracking failure is essential both for building a robust mapping system, and, as shown later in our paper, in our approach to building constraints between submaps.

The elementary approach for detecting tracking failure is manually to set thresholds on metrics extracted from the tracking process. These could require, for example, that the residual should not be larger than a threshold. In this work we instead train a classifier to separate tracking failure cases and success. For each optimization of ϵ_{ICP}, we measure the percentage of inlier pixels, the determinant of the Hessian and the final residual. We next expand these three values into a 20-dimensional descriptor using a χ^2 kernel map [35]. Lastly, we use an SVM classifier to separate between tracking success and failure. We obtain the required classification parameters by training on the '7 scenes' dataset of [36].

This dataset contains a set of 7 scenes and for each scene, several image sequences are captured and split into a training and a test set. We use the training set to build 3D reconstructions as in Sect. 2.1 and start tracking attempts for the test set using relocalisation [36]. If the pose is correctly estimated with less than 2 mm translational error and less than 2° rotational error from the ground truth, we consider tracking successful. Splitting the resulting 167220 tracking results along with their 3-dimensional feature vectors into half for training and half for test, we obtain a tracking failure detection accuracy of 95 %. Without the kernel expansion accuracy drops to 92.4 %, with virtually no gain in processing speed. In addition to successful and failed tracking, the distance from the support vector becomes an indicator for poor and good tracking accuracy.

2.3 Construction of Submaps

One of the key features of our proposed method is the use of submaps. In the works of [25,26], submaps are created entirely based on a temporal criterion: every k frames a new subvolume is created, and coincidentally the default is $k = 50$ in both works. This means the number of submaps grows linearly with time, not with the explored area, and the submap characteristics change depending on the speed of camera motion. Alternative approaches (e.g. [24]) use thresholds on camera motion to start submaps. Instead we propose a novel criterion: we evaluate the visibility of parts of the scene, and start a new submap once the camera viewport moves away from the central part of the previous submap.

Voxel block hashing lends itself ideally towards such a method. As revised in Sect. 2.1, we allocate new parts of the scene in blocks of $8 \times 8 \times 8$ voxels. Furthermore we maintain a list of visible blocks $\mathbb{V}$. Since blocks allocated early on are more likely to reside near the *core* of a submap, we evaluate the fraction r_{vis} of visible blocks within $\mathbb{V}$ with a smaller creation index than a threshold B. If the fraction drops below a threshold $r_{\text{vis}} < \theta$, the camera viewport has moved on and we can consider starting a new submap. The parameters B and θ therefore jointly determine the typical size of submaps.

3 Large Scale Reconstruction

We have defined the preliminaries of our 3D mapping pipeline, along with criteria for assessing the tracking quality and for determining whether to start a new submap. In the following, we clarify (i) how we relate multiple submaps to each other (Sect. 3.1), (ii) how we incorporate loop closures (Sects. 3.2, 3.3 and 3.4) and (iii) how we visualize our collection of submaps (Sect. 3.5).

3.1 Constraints Between Submaps

As in previous works [24–26], a fundamental idea is that local submaps can overlap each other. However, unlike these previous works, we process multiple submaps concurrently, which is still easily possible in real time thanks to the extremely efficient implementation of voxel block hashing in [17].

Whenever we hypothesize a new overlap of submaps, we run an additional independent instance of the map building process. Tracking continues in as many submaps as possible, noting that the camera trajectories should be identical in a global reference frame. This gives us constraints between the submaps. The transformation $\mathbf{T}_{t,i,j}$ from submap i to submap j, as observed at time t is:

$$\mathbf{T}_{t,i,j} = \mathbf{T}_{t,j}^{-1} \mathbf{T}_{t,i}, \tag{4}$$

where $\mathbf{T}_{t,i}$ and $\mathbf{T}_{t,j}$ are the poses tracked in submaps i and j respectively.

Of course, we get the highest quality valid constraints $\mathbf{T}_{t,i,j}$ when tracking is successful. To enforce consistency, multiple such constraints are furthermore

aggregated to an robust overall estimate $\mathbf{T}_{i,j}$. We consider $\mathbf{v}(\mathbf{T})$ to be a compact 6-vector representing the transformation $\mathbf{T}$ and compute

$$\mathbf{v}(\hat{\mathbf{T}}_{i,j}) = \sum_t w_t \mathbf{v}(\mathbf{T}_{t,i,j}) + w\mathbf{v}(\mathbf{T}_{i,j}) \sum_t w_t + w, \tag{5}$$

with $w_t = \frac{\sqrt{2b\hat{r} - b^2}}{\hat{r}}$ and $\hat{r} = \max(\|\mathbf{v}(\mathbf{T}_{t,i,j}) - \mathbf{v}(\hat{\mathbf{T}}_{i,j})\|, b)$ resulting from the robust Huber error norm with outlier threshold b, and w the number of previous observations that went into the estimation of $\mathbf{T}_{i,j}$, if any. Using iteratively reweighted least squares, we compute not only an estimate $\hat{\mathbf{T}}_{i,j}$, but also the inliers amongst the set of new constraints $\mathbf{T}_{t,i,j}$. As we will detail in the following sections, we can stop tracking a submap and discard the newly computed $\mathbf{T}_{t,i,j}$ if they are not self-consistent or if they are not consistent with our previous estimate $\mathbf{T}_{i,j}$.

In contrast to the ideas presented in previous works [24–26], the constraints between submaps in our system are essentially computed for free, as the tracking has to be performed anyway and the aggregation is lightweight. This greatly reduces the overall computational cost.

3.2 Loop Closure Detection and Relocalisation

In order to manage loop closures, we have to detect whenever the camera is revisiting a previously observed part of the scene. We use the keyframe based relocalisation system of Glocker et al. [36] for this, due to its simplicity and efficiency. While the tracking and mapping of the volumetric base system is running almost entirely on the GPU, this relocalisation module can easily run in parallel on a single CPU core without a noticeable impact on performance.

Each incoming depth image D_t is subsampled to a resolution of 40×30 pixels and filtered with a Gaussian with $\sigma = 2.5$, resulting in $\tilde{D}_t$. We then use a fern conservatory to compute a simplified encoding for the image. In our implementation, the conservatory consists of $n_F = 500$ ferns and each computes $n_B = 4$ binary decisions. Each binary decision is thresholding an individual pixel in $\tilde{D}_t$ according to $\tilde{D}_t(\mathbf{x}_i) < \vartheta_i$, where $\mathbf{x}_i$ and ϑ_i are chosen randomly at initialisation. The concatenation of binary codes $f_i = \{0, 1\}$ of one fern results in a code block $b_j = (f_1, \ldots, f_{n_B})$. Using the operator $\neq$ to return a binary value $\{0, 1\}$, [36] describes an efficient way of computing the dissimilarity

$$\text{BlockHD}(\tilde{D}_{t_1}, \tilde{D}_{t_2}) = \frac{1}{n_F} \sum_j \left(b_j(\tilde{D}_{t_1}) \neq b_j(\tilde{D}_{t_2}) \right) \tag{6}$$

between multiple codes simultaneously. A list of codes for the keyframes is maintained, and if the minimum dissimilarity score of $\tilde{D}_t$ to all existing keyframes is above a threshold, the image is considered for addition as a new keyframe.

As explained in the following, we use this process both for detecting loop closures and for relocalisation.

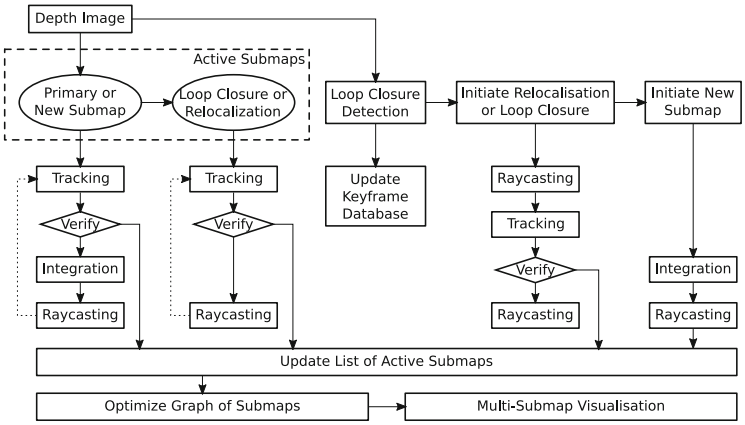


Fig. 2. Overview of the control logic. See Sect. 3.3 for detailed explanation.

3.3 Control Logic and Validation of Loop Closure

The loop closure detection process returns only keyframe IDs. While these can readily be linked to tuples of poses and submap indices, a more sophisticated treatment is clearly required to verify the proposals. This is done by our control logic, an outline of which is given in Fig. 2.

We maintain a list of *active submaps* at all times. While tracking and raycasting are usually performed in all active submaps, new depth information usually only gets integrated into one selected *primary submap*. At the beginning of the processing, this is just the first and only submap in our list. We evaluate each tracking result as outlined in Sect. 2.2 and if tracking is poor, we do not integrate depth information at all but still continue tracking. If tracking is lost in any submap, it becomes inactive.

Once the camera has moved away from the core of the primary submap according to the criterion defined in Sect. 2.3, we initiate a *new submap*. Since this new submap is initially empty, we also have to integrate depth information there in order to start the raycasting and tracking process. We accumulate constraints between the primary and the new submap as outlined in Sect. 3.1 and, once we have a stable estimate of the relative pose with a number N_{stable} inlier frames, we declare the new submap as readily initialised.

In parallel, we run the Loop Closure Detection system outlined in Sect. 3.2. If we have a stable tracking result from the primary scene, and if the internal criteria of the loop closure detection system suggest adding a new keyframe, we update the database of keyframes. Furthermore, if the Loop Closure Detection system suggests that we have seen a similar depth image before while tracking in a submap, that is not currently active, we initiate a new *loop closure attempt*. This means we declare the newly detected other submap as an active submap and start raycasting and tracking in it. Since we attempt to track subsequent frames in both (i) the current primary submap and (ii) the submap where we

attempt to close the loop, we can again establish constraints between the two. We still have to verify the loop closure, and for this we attempt to robustly estimate a relative pose between the two submaps as before, but this time honouring the aggregagte of the previously acquired pose constraints. Once we get N_{stable} inlier frames, we declare the loop closure successful. If this is not the case after N_{attempts} frames, we dismiss the attempted loop closure as erroneous.

Obviously, if tracking of all active scenes is lost, we can not establish relative pose constraints, but instead attempt relocalization. In this case, we declare the relocalisation successful if N_{stable} tracking attempts have been declared successful by our tracking quality criterion from Sect. 2.2, and we declare it failed if it is not successful for N_{attempts} frames.

At the end of the processing pipeline, we maintain the list of *active submaps*, removing submaps that can no longer be tracked and loop closure or relocalisation attempts that have failed. If there are multiple candidates, we also pick a new *primary scene* by checking our submap visibility constraint from Sect. 2.3 and selecting the submap with the largest portion currently visible. If any new scenes or loop closure attempts have been successfully validated, we also trigger a new process for optimising the graph of submaps, which we explain in the following Sect. 3.4. The results from this optimization are not used by the core pipeline outlined above in any way, but are crucially important for visualizing or evaluating the global map, as we shall discuss in Sect. 3.5.

3.4 Submap Graph Optimisation

From the above pipeline we obtain a number of constraints between pairs of submaps which can be used to estimate the pose of each submap in global coordinates. If there are no loop closures it is trivial to compute the exact submap poses, but more generally a pose graph optimization problem [30] arises.

Denote by $\hat{\mathbf{q}}(\mathbf{T})$ the three imaginary components of the quaternion representing the rotational part of an Euclidean transformation $\mathbf{T}$. Further let $\mathbf{t}(\mathbf{T})$ be the vector of the translational components and $\mathbf{v}(\mathbf{T}) = (\hat{\mathbf{q}}(\mathbf{T}), \mathbf{t}(\mathbf{T}))^\top$ the concatenation of the two. If $\mathbf{P}_i$ denotes the pose of a submap i and $\mathbf{T}_{i,j}$ the relative constraint between them as described in Sect. 3.1, then the error function we want to minimize is:

$$\epsilon_{\text{graph}} = \sum_{\{i,j\} \in \mathbb{T}} \|\mathbf{v}(\mathbf{P}_i \mathbf{P}_j^{-1} \mathbf{T}_{i,j})\|_2, \tag{7}$$

where $\mathbb{T}$ denotes the set of all pairwise constraints $\mathbf{T}_{i,j}$ that have been gathered so far. This function can be optimised with standard non-linear methods (in our case Levenberg-Marquardt), and as in [30] we use the quaternion representation for the ease of computing derivatives $\nabla \hat{\mathbf{q}}$. It is also noteworthy that the Hessian $\nabla^2 \epsilon_{\text{graph}}$ or its approximation is sparse. While for a small number of up to 100 submaps there is no noticeable difference, the use of sparse and super-nodal matrix factorisation methods [37] is advisable. In practice this optimisation is only triggered occasionally and its low computational complexity allows it to easily run in a background thread on the CPU.

3.5 Global Model Visualisation

The results of the Submap Graph Optimisation and of the online updates of the local submaps are crucially important for the live, real time visualisation of the global map. This provides users with immediate feedback and allows them to perform future path planning according to their desired exploration goals.

The representation of local submaps as T-SDFs has the major advantage, that the individual submaps can be fused on the fly while rendering the global map. To this effect, we define a new, combined T-SDF $\hat{F}$ as

$$\hat{F}(\mathbf{X}) = \sum_i F_w(\mathbf{P}_i \boldsymbol{X}) F(\mathbf{P}_i \boldsymbol{X}), \tag{8}$$

where $\mathbf{P}_i$ denotes the pose of submap i as estimated using Eq. (7) and where we use the notation F_w to denote taking the w entry out of a T-SDF F. We can simply substitute $\hat{F}$ in the raycasting step as outlined in Sect. 2.1 and extract the 3D surface, normals and (optionally) colours as before. Note that, while the computational complexity naturally increases when many submaps are visible at the same time, we can still easily display a number of them simultaneously while still estimating the global map in the background using standard, consumer grade hardware. This can be further optimised by limiting the number of submaps displayed simultaneously and pre-selecting a list of suitable submaps.

4 Evaluation

We illustrate the results achieved with our system in Figs. 1 and 3. Along with the video attachment in the supplementary material, these provide a qualitative impression of our method. In the following we will also evaluate our system quantitatively (Sect. 4.1) and investigate its memory requirements (Sect. 4.2) and runtime (Sect. 4.3).

4.1 Quantitative Results

While many previous works evaluate the accuracy of the camera trajectory as a proxy for the overall reconstruction and mapping process, this is not an option for us. Our method is map-centric, and we do not estimate a unique and globally consistent camera trajectory. In fact, we explicitly track the camera multiple times simultaneously relative to different local submaps. Our evaluation therefore focusses on the accuracy of the 3D maps that we generate.

We use the dataset from [39], and in particular the synthetically rendered `living-room` sequences that come with a ground truth 3D model. We run our proposed system on the image sequences provided, then extract a surface mesh from the combined SDF $\hat{\mathbf{F}}$ using Marching Cubes, and register the resulting reconstruction with the original 3D model. The observed average errors are reported in Table 1. Note in particular that the sequence `lr-kt3` does trigger a loop closure event and therefore tests our whole pipeline.

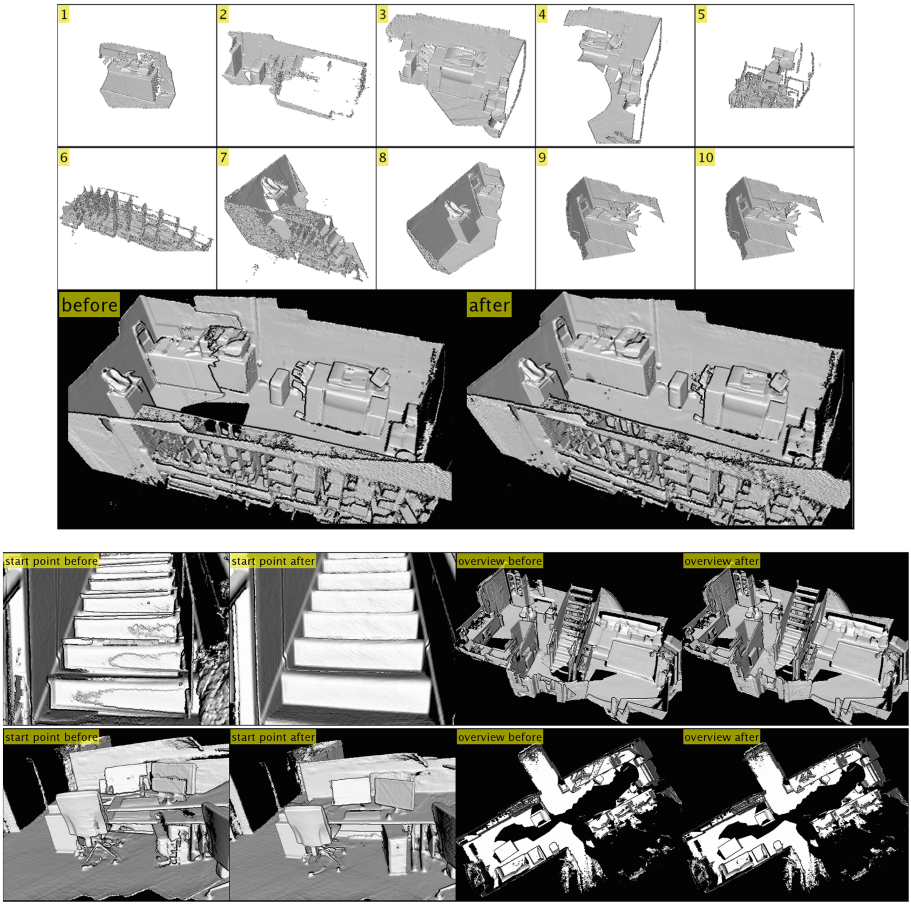
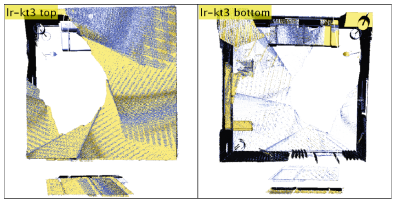
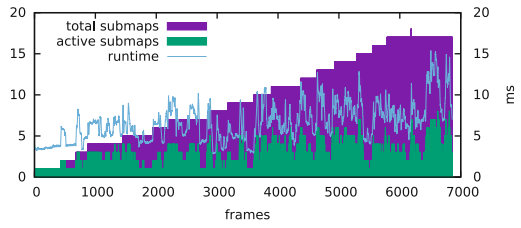


Fig. 3. Example reconstructions. We show the reconstruction result before and after loop closure, along with (top) images 1–10 showing constituent submaps and (bottom) the point of closure. The data for the top sequence is from [38].

Table 1. Left: Surface reconstruction error as evaluated on the ICL-NUIM dataset [39]. Right: Top and bottom views of the reconstruction obtained for sequence `lr-kt3`.

sequence	`lr-kt0`	`lr-kt1`	`lr-kt2`	`lr-kt3`
DVO SLAM	0.032	0.061	0.119	0.053
RGB-D SLAM	0.044	0.032	0.031	0.167
MRSMap	0.061	0.140	0.098	0.248
Kintinouous	0.011	0.008	0.009	0.150
ElasticFusion	**0.007**	**0.007**	0.008	0.028
Our system	0.013	0.011	**0.001**	**0.014**





		Max Reloc Seeds		
B	Θ	1	2	3
500	0.1	7.18	7.51	7.61
500	0.2	7.33	7.70	7.99
500	0.4	8.06	8.39	8.40
1000	0.1	7.16	7.46	7.62
1000	0.2	7.57	7.63	7.85
1000	0.4	8.10	8.37	8.40

Fig. 4. Left: Number of total and active submaps as well as processing time over the course of a long sequence. A running average over 10 frames was used to display processing times. Right: Overall average processing time (in milliseconds) for the sequence used to generate the bottom-most result in Fig. 3, as a function of scene size (parameters B and Θ) and the maximum allowed number of relocalisation seeds.

4.2 Memory Requirements

With large-scale 3D reconstruction in mind, scalability is obviously a major concern. In our system, each submap currently has a defined maximum size and memory footprint, and the overall memory requirements scale with the number of allocated submaps. In Fig. 4 we illustrate the growth of these memory requirements over the course of a long sequence. Note that the plateau towards the end indicates that the system is tracking within existing submaps because of a loop closure. The steps in between indicate rapid or not so rapid exploration of new parts of the scene.

We also note that not all submaps have to be kept in active memory at all times. Only a subset is used for tracking and updating the map, and the others can be swapped out, to host memory or even to disk. Figure 4(left) also shows the number of active submaps at any given time, and, while this number is varying quite significantly over the course of the sequence, it usually stays well below 5 active submaps, with only occasional spikes up to 8 and an average of 2.7. With well implemented swapping methods, this behaviour allows almost constant active memory requirements regardless of the sequence length.

Using the block based representation of our base system [17], our submaps consist on average of about 14636 allocated voxel blocks, the largest one of 29428 blocks for this sequence. This is at a voxel grid resolution of $s = 5\,\text{mm}$: at coarser resolutions the memory requirements of submaps shrink accordingly. In our current system we therefore limit the maximum number of blocks per submap to 65536. With each block representing $8 \times 8 \times 8$ voxels, and each voxel requiring 3 bytes of memory, along with an overhead of about 1 MB for the hash table data, each submap therefore requires about 100 MB of memory. Colour information, if desired, requires another 3 bytes per voxel, hence doubling the memory requirements of each submap. Still, even lower grade consumer graphics hardware will easily hold a few such submaps in memory.

4.3 Runtime

One of the key contributions of this work is an online system running in real time. To verify this claim, we measure the overall processing time for each frame on a test system with a Intel Core i7-5960X CPU and a Nvidia Titan X GPU. Our measurements take into account the tracking, integration and raycasting in each submap (Sect. 2.1), as well as all loop closure detection (Sect. 3.2), verification and other maintainance (Sect. 3.3), and they are the joint CPU and GPU times.

The resulting numbers are included in Fig. 4(left) for the sequence shown in Fig. 1. Obviously there is a high correlation between the number of active submaps and the processing time, and as [17] explains, the majority of the runtime is spent in raycasting and tracking, which are the key steps to our approach. Since the number of active submaps is independent of sequence length and overall scene size, the processing time also does not grow, but remains fairly constant. Even at peak times the processing requires just about 15.4 ms, and the average over the whole sequence is 6.6 ms, corresponding to about 150 frames processed per second. Figure 4(right) shows the average processing time for the sequence used to generate the bottom-most result in Fig. 3, w.r.t. scene size (parameters B and Θ) and the maximum allowed number of relocalisation seeds. The processing time required for the same sequence without using the loop closure algorithm was 5.4 ms, meaning that the loop closure operations adds as little as *1.8* ms over the original processing. This overhead is proportional to the number of scenes, as shown also by Fig. 4(left), but the increase is minor.

5 Conclusions

In this paper we have described an end-to-end processing pipeline which delivers large scale, dense, and 3D maps to the user from depth imagery at frame rate and with a average latency per frame of less than 7 ms. Loops are closed on the fly. The underlying representation and processing shares much with earlier works — a volumetric representation utilizing a truncated signed distance function is combined with a very carefully coded voxel block hashing scheme to mitigate memory growth. But key to overall performance, and performance through loop closure, is the use of submapping, a technique common in sparse reconstruction but less so in dense approaches.

We propose a novel criterion for dense submap initiation, involving detection of when the camera moves away from its current quasi-local viewing volume, which suppresses the over-frequent creation of submaps using earlier methods. Conveniently, our method is well-suited to voxel block hashing. We introduce a method of submap alignment which demands concordance of camera trajectories rather than full structural agreement in overlapping regions, a simplification which greatly reduces computation load, indeed to the point where alignment is essentially cost-free. Last, we describe a process for loop closure detection based on an earlier method of sensor relocalization.

The system is fully implemented on a single high-end GPU and host CPU. Further to the swapping described in previous works [16,17], very large scale

global maps can easily be swapped out from active GPU memory to main memory or even to disk. Future experimentation is aimed at characterizing performance during swapping in ever longer runs.

Acknowledgments. This work is partially supported by Huawei Technologies Co. Ltd. any by grant EP/J014990 from the UK's Engineering and Physical Science Research Council.

References

1. Newcombe, R.A., Davison, A.J.: Live dense reconstruction with a single moving camera. In: Proceedings 23rd IEEE Conference on Computer Vision and Pattern Recognition, pp. 1498–1505 (2010)
2. Newcombe, R.A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A.J., Kohli, P., Shotton, J., Hodges, S., Fitzgibbon, A.: KinectFusion: real-time dense surface mapping and tracking. In: International Symposium on Mixed and Augmented Reality, pp. 127–136 (2011)
3. Whelan, T., Johannsson, H., Kaess, M., Leonard, J.J., McDonald, J.: Robust real-time visual odometry for dense RGB-D mapping. In: Proceedings of 2013 IEEE International Conference on Robotics and Automation, pp. 5724–5731 (2013)
4. Bylow, E., Sturm, J., Kerl, C., Kahl, F., Cremers, D.: Real-time camera tracking and 3D reconstruction using signed distance functions. In: Proceedings Robotics: Science and Systems Conference (2013)
5. Engel, J., Schöps, T., Cremers, D.: LSD-SLAM: large-scale direct monocular SLAM. In: Proceedings of 13th European Conference on Computer Vision, pp. 834–849 (2014)
6. Newcombe, R.A., Fox, D., Seitz, S.M.: Dynamic fusion: reconstruction and tracking of non-rigid scenes in real time. In: Proceedings of 28th IEEE Conference on Computer Vision and Pattern Recognition (2015)
7. Pollefeys, M., van Gool, L., Vergauwen, M., Verbiest, F., Cornelis, K., Tops, J., Koch, R.: Visual modeling with a hand-held camera. Int. J. Comput. Vis. **59**(3), 207–232 (2004)
8. Seitz, S.M., Curless, B., Diebel, J., Scharstein, D., Szeliski, R.: A comparison and evaluation of multi-view stereo reconstruction algorithms. In: Proceedings of 19th IEEE Conference on Computer Vision and Pattern Recognition, pp. 519–528 (2006)
9. Davison, A.J., Reid, I.D., Molton, N.D., Stasse, O.: MonoSLAM: real-time single camera SLAM. IEEE Trans. Pattern Anal. Mach. Intell. **26**(6), 1052–1067 (2007)
10. Klein, G., Murray, D.W.: Parallel tracking and mapping for small AR workspaces. In: Proceedings of 6th IEEE/ACM International Symposium on Mixed and Augmented Reality, pp. 225–234 (2007)
11. Newcombe, R.A., Lovegrove, S.J., Davison, A.J.: DTAM: dense tracking and mapping in real-time. In: International Conference on Computer Vision (ICCV), pp. 2320–2327 (2011)
12. Pradeep, V., Rhemann, C., Izadi, S., Zach, C., Bleyer, M., Bathiche, S.: MonoFusion: real-time 3D reconstruction of small scenes with a single web camera. In: International Symposium on Mixed and Augmented Reality, pp. 83–88 (2013)
13. Steinbruecker, F., Sturm, J., Cremers, D.: Volumetric 3D mapping in real-time on a CPU. In: International Conference on Robotics and Automation (ICRA), pp. 2021–2028 (2014)

14. Chen, J., Bautembach, D., Izadi, S.: Scalable real-time volumetric surface reconstruction. ACM Trans. Graph. **32**(4), 113:1–113:16 (2013)
15. Zeng, M., Zhao, F., Zheng, J., Liu, X.: Octree-based fusion for realtime 3D reconstruction. Graph. Models **75**(3), 126–136 (2013)
16. Nießner, M., Zollhöfer, M., Izadi, S., Stamminger, M.: Real-time 3D reconstruction at scale using voxel hashing. ACM Trans. Graph. **32**(6), 169:1–169:11 (2013)
17. Kähler, O., Prisacariu, V.A., Ren, C.Y., Sun, X., Torr, P.H., Murray, D.W.: Very high frame rate volumetric integration of depth images on mobile devices. IEEE Trans. Vis. Comput. Graph. (Proceedings International Symposium on Mixed and Augmented Reality 2015) **21**(11), 1241–1250 (2015)
18. Whelan, T., Leutenegger, S., Moreno, R.S., Glocker, B., Davison, A.: Elasticfusion: dense SLAM without a pose graph. In: Proceedings of Robotics: Science and Systems (2015)
19. Endres, F., Hess, J., Sturm, J., Cremers, D., Burgard, W.: 3-D mapping with an RGB-D camera. IEEE Trans. Robot. **30**(1), 177–187 (2014)
20. Kerl, C., Sturm, J., Cremers, D.: Dense visual SLAM for RGB-D cameras. In: Intelligent Robots and Systems (IROS), pp. 2100–2106 (2013)
21. Meilland, M., Comport, A.I.: On unifying key-frame and voxel-based dense visual SLAM at large scales. In: Intelligent Robots and Systems (IROS), pp. 3677–3683 (2013)
22. Engel, J., Schöps, T., Cremers, D.: LSD-SLAM: large-scale direct monocular SLAM. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014, Part II. LNCS, vol. 8690, pp. 834–849. Springer, Heidelberg (2014)
23. Weise, T., Wismer, T., Leibe, B., Gool, L.V.: Online loop closure for real-time interactive 3D scanning. Comput. Vis. Image Underst. **115**(5), 635–648 (2011). Special issue on 3D Imaging and Modelling
24. Stuckler, J., Behnke, S.: Multi-resolution surfel maps for efficient dense 3D modeling and tracking. J. Vis. Commun. Image Representation **25**(1), 137–147 (2014). Visual Understanding and Applications with RGB-D Cameras
25. Fioraio, N., Taylor, J., Fitzgibbon, A., Stefano, L.D., Izadi, S.: Large-scale and drift-free surface reconstruction using online subvolume registration. In: Computer Vision and Pattern Recognition (CVPR), pp. 4475–4483 (2015)
26. Choi, S., Zhou, Q.Y., Koltun, V.: Robust reconstruction of indoor scenes. In: Computer Vision and Pattern Recognition (CVPR), pp. 5556–5565 (2015)
27. Curless, B., Levoy, M.: A volumetric method for building complex models from range images. In: Conference on Computer Graphics and Interactive Techniques (SIGGRAPH), pp. 303–312 (1996)
28. Klingensmith, M., Dryanovski, I., Srinivasa, S., Xiao, J.: Chisel: Real time large scale 3D reconstruction onboard a mobile device using spatially hashed signed distance fields. In: Proceedings of Robotics: Science and Systems (2015)
29. Schops, T., Sattler, T., Hane, C., Pollefeys, M.: 3D modeling on the go: interactive 3D reconstruction of large-scale scenes on mobile devices. In: International Conference on 3D Vision (3DV), pp. 291–299 (2015)
30. Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., Burgard, W.: g2o: a general framework for graph optimization. In: International Conference on Robotics and Automation (ICRA), pp. 3607–3613 (2011)
31. Whelan, T., Kaess, M., Johannsson, H., Fallon, M., Leonard, J.J., McDonald, J.: Real-time large-scale dense RGB-D SLAM with volumetric fusion. Int. J. Robot. Res. **34**(4–5), 598626 (2015)

32. Eade, E., Drummond, T.: Unified loop closing and recovery for real time monocular SLAM. In: Proceedings of the British Machine Vision Conference, pp. 6.1–6.10 (2008)
33. Williams, B., Cummins, M., Neira, J., Newman, P., Reid, I., Tardos, J.: An image-to-map loop closing method for monocular SLAM. In: Proceedings of International Conference on Intelligent Robots and and Systems, pp. 2053–2059 (2008)
34. Estrada, C., Neira, J., Tardos, J.D.: Hierarchical SLAM: real-time accurate mapping of large environments. IEEE Trans. Robot. **21**(4), 588–596 (2005)
35. Vedaldi, A., Zisserman, A.: Efficient additive kernels via explicit feature maps. Pattern Anal. Mach. Intell. **34**(3), 480–492 (2011)
36. Glocker, B., Izadi, S., Shotton, J., Criminisi, A.: Real-time RGB-D camera relocalization. In: International Symposium on Mixed and Augmented Reality (ISMAR), pp. 173–179 (2013)
37. Davis, T.A.: Direct Methods for Sparse Linear Systems. SIAM Series on Fundamentals of Algorithms. PWS Publishing, New York (2006)
38. Zhou, Q.Y., Koltun, V.: Dense scene reconstruction with points of interest. ACM Trans. Graph. **32**(4), 112:1–112:8 (2013)
39. Handa, A., Whelan, T., McDonald, J., Davison, A.: A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. In: International Conference on Robotics and Automation, pp. 1524–1531 (2014)