Real-time Dense Mapping for Online Processing and Navigation

Yonggen Ling
Tencent AI Lab
China
ylingaa@connect.ust.hk

Shaojie Shen
The Hong Kong University of Science and Technology
Hong Kong, China.
eeshaojie@ust.hk

Abstract

Autonomous robots require accurate localizations and dense mappings for motion planning. We consider the navigation scenario where the dense representation of the robot surrounding must be immediately available, and require that the system is capable of an instantaneous map correction if a loop closure is detected by the localization module. To satisfy the real-time processing requirement of online robotics applications, our presented system bounds the algorithmic complexity of the localization pipeline by restricting the number of variables to be optimized at each time instant. A dense map representation along with a local dense map reconstruction strategy are also proposed. Despite the limits that are imposed by the real-time requirement and planning safety, the mapping quality of our method is comparable to other competitive methods. For implementations, we additionally introduce a few engineering considerations, such as the system architecture, the variable initialization, the memory management, the image processing, etc., to improve the system performance. Extensive experimental validations of our presented system are performed on the KITTI and NewCollege datasets, and through an online experiment around the HKUST university campus. We release our implementation as open-source ROS packages for the benefit of the community.

1 Introduction

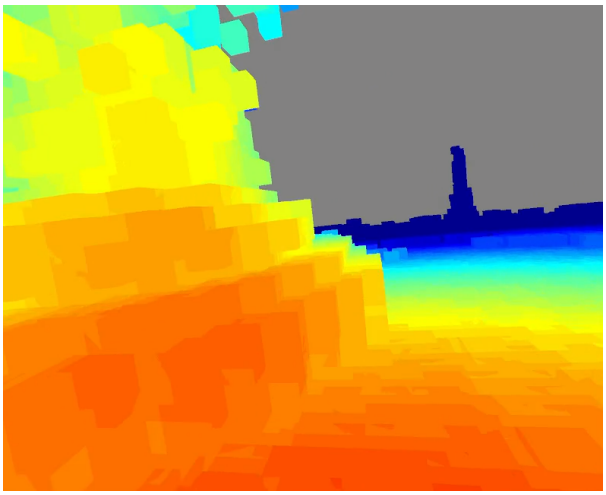
Large-scale long-term autonomous navigation is a growing demand for many robotics applications, such as search and rescue, space and underwater exploration, mobile manipulation, etc. The fundamental component of autonomous robotics systems is accurate robot poses and dense maps that serve as the perception input for obstacle avoidance and

path planning. Traditionally, dense 3D navigational maps are built incrementally in two stages. The first step is to estimate the poses of sensors (such as cameras and laser scanners) together with the positions of sparse features/semi-dense textures using SLAM algorithms (Li and Mourikis, 2013; Shen et al., 2015; Hesch et al., 2014; Huang et al., 2011; Zhang and Singh, 2014). The second step is to create some form of dense maps by projecting 3D points/range scans/disparity maps onto a single global coordinate frame. Popular map representations are occupancy grids (Elfes, 1989), OctoMaps (Meagher, 1982; Hornung et al., 2013), elevation maps (Hebert et al., 1989; Ryde and Hu, 2010), and point clouds (Cole and Newman, 2006; Schoeps et al., 2015). In this way, the dense 3D map is conditioned on the optimized SLAM poses. This mapping pipeline, however, is liable to be problematic if a robot operates in a large area for a long period of time and loop closures are detected. Although the drift of robot poses can be corrected by SLAM algorithms, the built dense map can not be updated in real time due to the huge computational burden involved in map update operations, such as space (re)allocation and ray-casting.

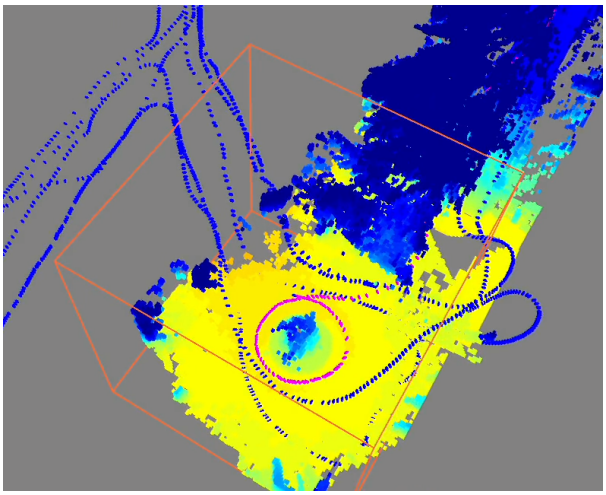
Robotics applications usually require real-time processing. This means that the input data have to be processed at a speed that is faster than or equal to the speed of the input data. The real-time processing requirement of robotics applications is challenging to satisfy because there are only limited computational resources (such as CPUs, GPUs and memory), payloads, and power on robots. In this work, we focus on developing a real-time dense mapping approach using stereo cameras as they are widely equipped on robots (because of their low-cost, small-size and low-power-consumption). Dense maps obtained on the fly at different time instants represent the maximum information about the perceived environment with the limited computational resources on mobile robots. The main contributions are as follows:

- We introduce a dense map representation in which it is easy to integrate depth maps and is flexible to deform in real time after loop closures.
- We propose a subvolume wrapping and blending scheme that operates on truncated signed distance functions. With this scheme, dynamic objects in the environment can be partly handled.
- We present a local map reconstruction strategy to reconstruct the local environment around the instant robot pose for planning based on the temporal and spatial correlations inherent in the proposed map representation.
- We take careful engineering considerations on the image corner feature detection, the pose graph optimization treatment, the bundle adjustment initialization, the memory management, and the multi-thread system architecture to for improving system performance.
- We show extensive experiments on various datasets and real-world data for performance evaluation.

(a) A captured image during the sequence.



(b) The first person view of the reconstructed local dense map.



(c) The third person view of the reconstructed local dense map.

Figure 1: Qualitative mapping results of the real-time campus reconstruction. (a) An instant campus environment with natural and artificial objects. (b) and (c) the first person view (the color varies with respect to the object distance) and the third person view (the color varies with respect to the height to show the environment structure) of the reconstructed local environment on the fly. We only build a local environment around the instant pose for motion planning because of the real-time processing requirement. The local environment consists of local maps within the local window (the orange box in (c)) and local volumes attached to neighboring keyframes (purple cameras in (c)). More results can be found in Sect. 5.2.4.

- We release our implementations as open-source packages available at: `https://github.com/ygling2008/dense_mapping`.

Videos showing the performance of our real-time dense mapping approach are available at: `https://1drv.ms/v/s!ApzRxvwAxXqQmguH6-iBshCx8W_J`. The rest of this paper is structured as follows. Details of the proposed approach are presented in Sect. 3. Implementation details are shown in Sect. 4. Sect. 5 gives an extensive evaluation of the system performance. We discuss the cons and pros of this work in Sect. 6. Finally, Sect. 7 draws the conclusion.

2 Related Work

In practical robotics systems, a relaxation for online dense mapping is made by using a prior map that is built offline beforehand. With prior maps, autonomous navigation can be done by localizing robot poses and planning between places within the built map (Middelberg et al., 2014; Lynen et al., 2015; Amazon Robotics LLC, 2015). (Middelberg et al., 2014) locates the global camera poses from localization servers, and updates the local camera poses using keyframe-based SLAM. The work of (Lynen et al., 2015) does not require localization servers for global localization; instead, it firstly reconstructs a 3D point cloud from a set of database images using SfM, and then compresses this 3D model by removing less significant landmarks and quantizing the 3D point descriptors for storage on mobile phones. Global positions and orientations of cameras are registered by matching 3D-3D matches between the compressed model and the captured images. (Amazon Robotics LLC, 2015) uses predefined markers for localization and navigation. The performance of (Amazon Robotics LLC, 2015) relies heavily on the performance of the marker detection and recognition module. All these systems (Middelberg et al., 2014; Lynen et al., 2015; Amazon Robotics LLC, 2015) have a significant drawback that the prior map can not account for the latest changes in the perceived environment. As a result, mobile robots may fail to or wrongly estimate their online poses and surroundings, making them at risk of collisions with obstacles. In addition, it is not possible for mobile robots to navigate through unknown environments using these systems.

Another relaxation approach is to neglect loop closures and interpret the world as a volumetric map consisting of an "infinite corridor" (Chen et al., 2016; Gao and Shen, 2016a; Ling et al., 2018). The simplest representation of the metric space is an occupancy grid (Elfes, 1989). An occupancy grid employs a tessellation of cubic volumes of equal size, called voxels. Each voxel is associated with an empty or occupied label to indicate the occupancy. To update the occupancy from noisy measurements, signed distance functions are usually used (Elfes, 1989). Depth measurements are averaged in the signed distance function field, and object surfaces are extracted by calculating the zero isosurface in the signed distance functions. Map manipulations can be easily conducted using the occupancy grid representation. However, this representation is not memory-efficient since the grid map should be initialized to be at least as big as the bounding box of the mapping environment, which limits its potential in large-scale situations. To overcome this difficulty, the OctoMap has been proposed for a hierarchical subdivision of the 3D space (Meagher, 1982; Wilhelms and Gelder, 1992; Hornung et al., 2013; Fairfield et al., 2007). The hierarchical property of this representation makes it possible to stop at any level of resolution, which keeps the 3D models compact. It explicitly represents not only the occupied space, but also the free and unknown areas. Another way to improve the compactness of the occupancy grid is to use the technique called voxel hashing (Nießner et al., 2013; Klingensmith et al., 2015) for indexing voxels. Only occupied voxels are stored; therefore, as the major part of the 3D space is usually empty, merely storing occupied

voxels leads to a significant memory reduction. If a single surface assumption is made about the environment, elevation maps (Hebert et al., 1989) that contains the depth profile of the ground surface, are sufficient to represent the 3D space. To relax the strict assumption of a single surface, (Triebel et al., 2006) and (Pfaff et al., 2007) propose to use multiple surfaces per cell, while (Ryde and Hu, 2010) maintains a set of occupied voxels for each cell in the 2D grid. Elevation maps are more memory-efficient than voxel grid methods; however, they are not as memory-efficient as the OctoMap or the voxel hashing if the ground surface consists of a large empty area. The run-time complexity of volumetric fusion scales with respect to the number of voxels; it will be very high if a large number of voxels are given. To alleviate the heavy computational burden, powerful GPUs, instead of CPUs, are adopted (Richard et al., 2011; Nießner et al., 2013). Voxels with signed distance functions have been applied to motion planning algorithms (Ratliff et al., 2009; Oleynikova et al., 2016). In (Ratliff et al., 2009) and (Oleynikova et al., 2016), motions are planned based on the signed distance functions. These algorithms prefer trajectories with larger signed distances for safety, while at the time take the smoothness of trajectories into consideration. The main issue of the occupancy grid representation is that the real topology of the environment is missing due to the lack of loop closures. It is therefore impossible for robots to find shortcuts between locations, which leads to suboptimal path planning results.

Reconstructing dense environments from sparse features is another way to reduce the system complexity. Some light-weight mapping approaches (Lovi et al., 2010; Litvinov and Lhuillier, 2013; Romanoni and Matteucci, 2015b; Romanoni and Matteucci, 2015a; Kutulakos and Seitz, 1999; Labatut et al., 2007) developed in the graphics community have the potential for real-time dense mapping. These algorithms build a 3D model from a sparse point cloud using a 3D Delaunay triangulation (Boissonnat et al., 1988). They then figure out which part of the 3D model is empty as well as which part of it is occupied using visibility constraints. Delaunay triangulation is powerful for its property (Boissonnat et al., 1988) that the circumscribing sphere of a tetrahedron does not contain a vertex in its interior, which avoids degenerate shapes; and the model is finer if points are denser. However, for real applications using the Delaunay triangulation and visibility constraints, there are challenges ahead. Firstly, most of proposed algorithms run offline (Kutulakos and Seitz, 1999; Labatut et al., 2007), or run incrementally but assuming camera poses are fixed or known beforehand (Litvinov and Lhuillier, 2013; Romanoni and Matteucci, 2015b; Romanoni and Matteucci, 2015a). They are not applicable to real-time applications where camera poses are estimated online. Secondly, except the work of (Lovi et al., 2010; Ling and Shen, 2017), the real-time performance at video frame rate is not reported. (Lovi et al., 2010) runs in real time in small, limited indoor environments, while (Ling and Shen, 2017) is able to run in real time in large-scale outdoor environments. Thirdly, loop closures in large-scale environments are usually ignored (Lovi et al., 2010; Litvinov and Lhuillier, 2013; Romanoni and Matteucci, 2015b; Romanoni and Matteucci, 2015a; Kutulakos and Seitz, 1999; Labatut et al., 2007). There is no chance to correct position drifts of the reconstructed 3D environment. Real-time performance in large-scale environments with loop closures are reported in (Ling and Shen,

2017); however, the mapping performance depends heavily on the sparse features detected and optimized. Details in reconstructions are missing, which leads to potential safety issues.

Fundamentally different from metric maps, topological maps (Gabe et al., 2010; Dudek et al., 1991; Kaelbling and Shatkay, 2002; Richter et al., 2013; Kuwata et al., 2009; Bry and Roy, 2011), whose nodes are places in the world and edges are paths between these places, have been popular in recent years. While edges in pure topological maps (Kuipers and Byun, 1988; Choset and Nagatani, 2001; Gabe et al., 2010) only include the connectivity of places, edges in topological-metric (Duckett et al., 2000; Filliat and Meyer, 2002; Angeli et al., 2009) maps consist of additional metric information (distance, directions, etc.) about how places are related. Therefore, the path planning is a matter of graph search among places. Topological maps have been extensively used for visual navigation in the past (Nister et al., 2004; Fraundorfer et al., 2007; Toon et al., 2007; Steder et al., 2007; Gabe et al., 2010). They are conceptually similar to the biological notion of the cognitive map, which was proposed by Tolman (Tolman, 1948) in studying how rats navigate mazes. They ignore the dense environment information within nodes, thus the amount of data to process is small. The topological relationships between places are often based on visual similarity, and many fast image searching and matching techniques, such as bag-of-words and local sensitive hashing methods, have been developed in the past. The additional metric information of topological-metric maps helps to improve the place recognition performance (Maddern et al., 2012; Pepperell et al., 2014). Works in (Konolige and Agrawal, 2008) and (Davison et al., 2007) show that including positions of sparse landmarks within nodes leads to some gains in the place recognition performance too. However, while topological maps are suitable for analyzing the reachability of given places, they are unsuitable for motion planning. Firstly, associating abstract places and routes with physical places and paths is not easy and there are many ambiguities. Secondly, the topological and metric interpretations of the world have to be well defined. Thirdly, while topological-metric maps support obstacle avoidance and local interactions within a small local environment, most motion planning algorithms require not only the connectivity information between individual places, but also the metric structure of intersections between places for planning smooth trajectories and for low-level control. Lastly, it is infeasible to plan a path that contains the connectivity information for areas that are observed but not physically traveled to by the robot.

While metric maps are more ready for optimal motion planning, topological maps are easier to operate on real-time systems. There have been a few attempts developed in the computer vision community to combine the benefits of both dense metric maps and topological maps (Fioraio et al., 2015; Whelan et al., 2014; Whelan et al., 2016). (Fioraio et al., 2015) subdivides the whole metric space with overlapping subvolumes and registers these subvolumes using an ICP-like method. (Whelan et al., 2014) parameterizes the environment surface with a deformation graph. (Whelan et al., 2016) follows a similar idea to the deformation graph and models the environment surface as a tessellation

of surfels, which are deformed if loops are closed. (Fioraio et al., 2015; Whelan et al., 2014; Whelan et al., 2016) focus on the detailed quality of static indoor reconstructions, but their motivations are not real-time long-term autonomous navigation in large-scale environments. They are all based on KinectFusion (Richard et al., 2011), in which high-quality depth measurements from RGB-D cameras are used. They are not applicable to applications in outdoor environments where depth sensors of RGB-D cameras do not work. Moreover, the algorithmic complexities of the subvolume registration in (Fioraio et al., 2015) and the surface deformation in (Whelan et al., 2016) scale with respect to the space size. Operating an ultimately very large dense map after long-term autonomy will lead to intractable computation time. Signed distance functions are summarized as triangular meshes and dropped in (Whelan et al., 2014), which makes it impossible to integrate the latest changes in the real-world environment and poses a significant threat to safe navigation.

3 Online Dense Mapping

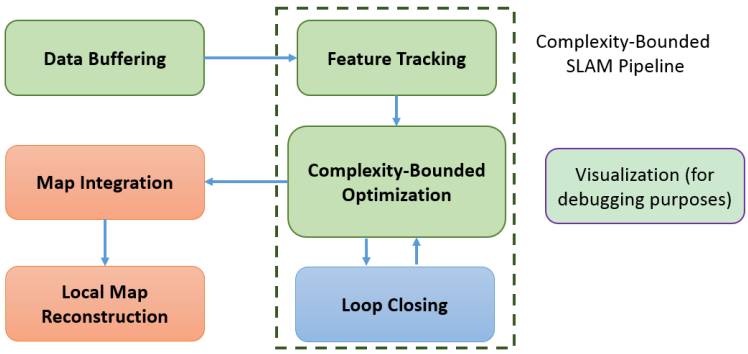


Figure 2: The multi-thread system pipeline. The center of our system is a complexity-bounded SLAM pipeline that estimates the latest camera poses. The map integration thread fuses past built maps and the latest built maps. The local map reconstruction thread reconstructs locally consistent maps for motion planning. The image acquisition and synchronization are done by a data buffering thread. The visualization thread is optional for debugging purposes.

The overall system pipeline is shown in Fig. 2. There are six threads running simultaneously to utilize the multi-core CPU architecture. The center of this system is a complexity-bounded SLAM pipeline. There are three threads in this SLAM pipeline: the feature tracking thread, the complexity-bounded optimization thread and the loop closing thread. The feature tracking thread detects sparse features on images using the ST image corner feature detector (Shi and Tomasi, 1994), and establishes correspondences between them using the sparse optical flow in (Baker and Matthews, 2004). Obtained feature matches are sent to the complexity-bounded optimization thread for the camera pose estimation. We apply a local bundle adjustment (Triggs et al., 1999) on limited camera poses and sparse features to balance the algorithmic complexity against the tracking accuracy. The loop closing thread re-localizes camera poses to reduce pose drift that has inevitably occurred after the long-term pose tracking. It maintains a recognition database

and searches for loops periodically via a bag-of-words place recognizer (DBoW2 (Gálvez-López and Tardós, 2012)). Candidates are verified by the visual similarity and the geometry consistency. The map integration thread projects maps built in the past to the local environment around the latest pose, and then merges them with the latest estimated local map. Another key component of this system is the local map reconstruction thread that builds local dense maps for motion planning. These local maps consists of mapping information that is temporally (built incrementally from stereo images as well as estimated camera poses) and spatially correlated (warped and blended using loop closing results). The size of local maps is bounded for the benefit of real-time computation. Finally, the data buffering thread is responsible for image acquisition and synchronization, while the visualization thread is an optional step for debugging purposes.

3.1 Complexity-Bounded SLAM Pipeline

Motivated by the real-time processing requirement for a robot, we propose a complexity-bounded SLAM pipeline appropriate for real-time processing. Different from the well-known ORB-SLAM2 (Mur-Artal and Tardós, 2017) where the computation time of the local bundle adjustment varies according to the size of the local sparse feature map, our SLAM pipeline is able to provide the latest camera pose within a bounded time. The key idea is to fix the number of keyframes and sparse features to be optimized in the bundle adjustment step and to run the global optimization in an independent lower-priority thread. We adopt the idea of sliding window filtering (Gabe et al., 2010) for jointly optimizing camera poses and sparse features. It will be shown in Sect. 5.1.1 that our complexity-bounded SLAM pipeline is able to provide locally accurate estimates in a bounded time.

3.1.1 Feature Tracking

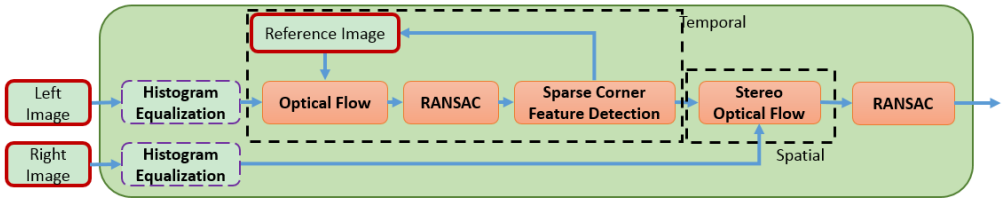


Figure 3: The pipeline of the feature tracking thread. This thread establishes both temporal and spatial correspondences between sparse feature observations in sequential frames, which serves as the input to the sliding window bundle adjustment thread.

The feature tracking thread aims to establish both temporal and spatial correspondences among captured images. It outputs the obtained correspondences to the complexity-bounded optimization thread at 10Hz (the maximum frequency). The whole feature tracking pipeline is shown in Fig. 3. To obtain temporal correspondences, we firstly

search image corner feature correspondences between the current frame (the left-hand image) and the reference frame using the Lucas-Kanade sparse optical flow (Baker and Matthews, 2004). A geometry check is then performed on the obtained matches. It is well known that image corner features in each of the images of the pair are related by fundamental matrices (Hartley and Zisserman, 2003). We use a robust estimator, RANSAC (Fischler and Bolles, 1981), to estimate these fundamental matrices and detect outlier matches that do not conform to the estimated fundamental matrices. Outlier matches are then removed. New image corner features are detected in the current frame via the ST image corner feature detector (Shi and Tomasi, 1994). The reference frame is reset to be the current frame. The Lucas-Kanade sparse optical flow is applied again from the left images to the right images to search spatial correspondences, followed by a geometry check using the robust estimator RANSAC (Fischler and Bolles, 1981) to remove outlier matches. This thread outputs both temporal and spatial correspondences of feature observations in sequential frames for the complexity-bounded optimization thread.

When detecting image corner features using the ST image corner feature detector (Shi and Tomasi, 1994), we enforce an even distribution of these image corner features. This is done by enforcing that the minimum distance between detected image corner features is beyond a predefined distance threshold (for an image resolution of 640×480, we empirically set it to 30.). Additionally, since the brightness consistency assumption in the Kucas-Kanade sparse optical flow as well as the local contrast assumption in image corner feature detection may be violated in some extreme cases, such as changing lighting conditions, low-contrast surfaces, featureless environments, etc., we include an optional step, contrast limited adaptive histogram equalization (Zuiderveld, 1994), to normalize captured images. We empirically find that enabling histogram equalization leads to increased tracking robustness. The effects of feature tracking on testing sequences with and without histogram equalization are presented in Sect. 5.2.2.

3.1.2 Complexity-Bounded Optimization

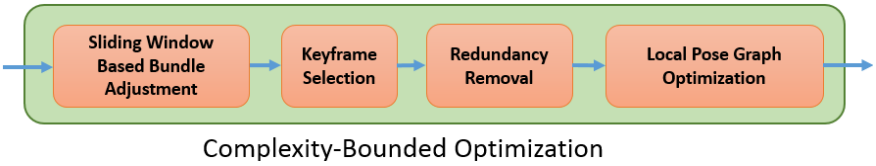


Figure 4: The pipeline of the complexity-bounded optimization thread. This estimates the latest camera pose in a bounded time, decides whether to insert a new keyframe, removes redundant information, and maintains locally consistent poses and sparse features.

The complexity-bounded optimization thread outputs the latest camera pose in a bounded time (Fig. 4). The key idea to bound the computation time is to restrict the number of camera poses and sparse features in the optimization (Fig. 5). Since there is redundancy among consecutive frames, we select keyframes and drop the other frames according to the parallex sufficiency. For further reducing the redundancy among consecutive frames, we cull keyframes and sparse

features if they are in the optimization sliding window and do not contribute to the latest pose estimation. We finally maintain a locally consistent pose graph for the local map reconstruction required by the motion planning module (Sect. 3.3). The reconstructed local pose graph consists of local poses and sparse features around the instant robot pose. Poses and sparse features around loop poses are also included if loops are closed in the loop closing thread (Sect. 3.1.3). Details about steps in this thread are given in the following.

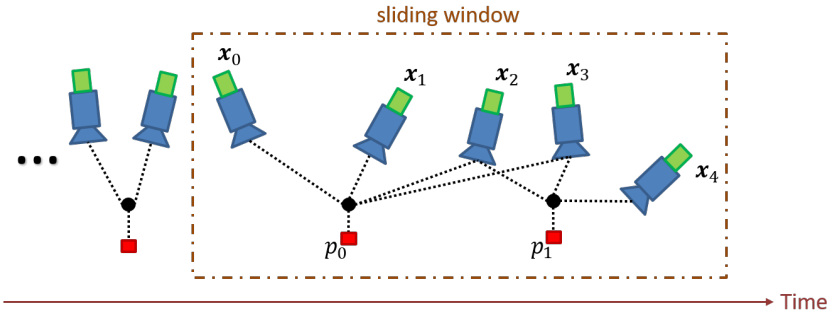


Figure 5: The illustration of the sliding window-based bundle adjustment. We optimize camera poses (shown in green) and feature positions (shown in red) within a fixed-size sliding window (the dashed box).

Sliding Window Based Bundle Adjustment We assume that cameras are calibrated. 3D positions of sparse features in the world coordinate are denoted as $\mathbf{p}_i^w = [x_i\ y_i\ z_i]$. Camera poses are denoted as $\mathbf{R}_i^w \in \text{SO(3)}$ for rotations and $\mathbf{t}_i^w \in \mathbb{R}^3$ for translations. Both sparse feature positions and poses are optimized by minimizing the reprojection error with respect to the tracked features obtained in the feature tracking thread. The error term for the observation of a sparse feature j in the camera i is

$$\mathbf{e}_{ij} = \mathbf{u}_{ij} - \pi(\mathbf{R}_w^i(\mathbf{p}_i^w - \mathbf{t}_i^w)), \tag{1}$$

where $\mathbf{u}_{ij}$ is the tracked feature location and π is the projection function that projects a sparse feature 3D position $\mathbf{p}_i$ into a unit camera plane:

$$\pi(p_i) = \begin{bmatrix} f_x \frac{x_i}{z_i} + c_x \\ f_y \frac{y_i}{z_i} + c_y \end{bmatrix}, \tag{2}$$

where f_x, f_y, c_x, and c_y are the focus length and principle point of the camera calibrated beforehand. We minimize the total error cost within a sliding window:

$$\min_{\mathbf{p}_j, \mathbf{R}_i, \mathbf{t}_i} \Sigma_{i=0,\ldots,N_w-1, j=0,1,\ldots,M_w-1} ||\mathbf{e}_{ij}||_2^2 \tag{3}$$

$$\text{s.t. } \mathbf{t}_0^w = \mathbf{0},$$

where N_w is the length of the sliding window, and M_w is the number of features to be optimized.

Additional error terms relating to the IMU measurements are also included in (3) if IMU measurements are available (such as the experiment in Sect. 5.2.4). For convenience of notation, we use $\mathbf{R}^w_{b_{k+1}}$ and $\mathbf{p}^w_{b_{k+1}}$ as the alias to $\mathbf{R}^w_{k+1}$ and $\mathbf{p}^w_{k+1}$. That is, we adopt an IMU-oriented representation. b_k denotes the IMU coordinate at time instant k while c_k is the corresponding camera coordinate. b_k and c_k are related by camera-IMU extrinsic $\mathbf{R}^b_c$ and $\mathbf{t}^b_c$. We use the IMU preintegration method in (Yang and Shen, 2016). Given two time instants t_{b_k} and $t_{b_{k+1}}$, we can write the IMU propagation model for position, velocity and rotation with respect to the earth's inertial frame:

$$\mathbf{p}^w_{b_{k+1}} = \mathbf{p}^w_{b_k} + \mathbf{v}^w_{b_k}\Delta t_k - \frac{1}{2}\mathbf{g}^w\Delta t_k^2 + \mathbf{R}^w_{b_k}\boldsymbol{\alpha}^k_{k+1} \tag{4}$$

$$\mathbf{v}^w_{b_{k+1}} = \mathbf{v}^w_{b_k} + \mathbf{R}^w_{b_k}\boldsymbol{\beta}^k_{k+1} - \mathbf{g}^w\Delta t_k \tag{5}$$

$$\mathbf{q}^w_{k+1} = \mathbf{q}^w_k \otimes \mathbf{q}^k_{k+1} \tag{6}$$

where $\Delta t_k = t_{b_{k+1}} - t_{b_k}$, and $\mathbf{g}^w = [0\ 0\ 9.8]$ is the gravity vector in the earth's inertial frame. $\boldsymbol{\alpha}^k_{k+1}$, $\boldsymbol{\beta}^k_{k+1}$ and $\mathbf{q}^k_{k+1}$ are calculated by integrating the linear accelerations $\mathbf{a}^{b_t}$ and angular velocities $\boldsymbol{\omega}^{b_t}$ between time instants t_{b_k} and $t_{b_{k+1}}$:

$$\mathbf{z}^k_{k+1} = \begin{bmatrix}\boldsymbol{\alpha}^k_{k+1}\\ \boldsymbol{\beta}^k_{k+1}\\ \mathbf{q}^k_{k+1}\end{bmatrix} = \begin{bmatrix}\iint_{t\in[t_{b_k},t_{b_{k+1}}]}\mathbf{R}^k_t(\mathbf{a}^{b_t}-\mathbf{b}^a_t)dt^2\\ \int_{t\in[t_{b_k},t_{b_{k+1}}]}\mathbf{R}^k_t(\mathbf{a}^{b_t}-\mathbf{b}^a_t)dt\\ \int_{t\in[t_{b_k},t_{b_{k+1}}]}\frac{1}{2}\begin{bmatrix}-\lfloor\boldsymbol{\omega}^{b_t}-\mathbf{b}^{\boldsymbol{\omega}}_t\times\rfloor & \boldsymbol{\omega}^{b_t}-\mathbf{b}^{\boldsymbol{\omega}}_t\\ -(\boldsymbol{\omega}^{b_t}-\mathbf{b}^{\boldsymbol{\omega}}_t)^T & \mathbf{0}\end{bmatrix}\mathbf{q}^k_t dt\end{bmatrix}, \tag{7}$$

where biases $\mathbf{b}^a_t$ and $\mathbf{b}^{\boldsymbol{\omega}}_t$ are modeled as Gaussian random walks. In the implementation, since only discrete measurements are available from IMUs, the numerical integration (Forster et al., 2015) is commonly employed to calculate the approximated integrations $\hat{\mathbf{z}}^k_{k+1} = \{\hat{\boldsymbol{\alpha}}^k_{k+1}, \hat{\boldsymbol{\beta}}^k_{k+1}, \hat{\mathbf{q}}^k_{k+1}\}$. The corresponding integration covariance $\boldsymbol{\Sigma}^k_{k+1}$ can also be obtained during the integration (Forster et al., 2015). The inertial error term is defined as

$$e_{S_i}(\hat{\mathbf{z}}^k_{k+1}, \mathcal{X}) = \begin{bmatrix}\delta\boldsymbol{\alpha}^k_{k+1}\\ \delta\boldsymbol{\beta}^k_{k+1}\\ \delta\boldsymbol{\theta}^k_{k+1}\end{bmatrix} = \begin{bmatrix}\mathbf{R}^{b_k}_w(\mathbf{p}^w_{b_{k+1}} - \mathbf{p}^w_{b_k} - \mathbf{v}^w_{b_k}\Delta t_k + \frac{1}{2}\mathbf{g}^w\Delta t_k^2) - \hat{\boldsymbol{\alpha}}^k_{k+1}\\ \mathbf{R}^{b_k}_w(\mathbf{v}^w_{b_{k+1}} - \mathbf{v}^w_{b_k} + \mathbf{g}^w\Delta t_k) - \hat{\boldsymbol{\beta}}^k_{k+1}\\ (\hat{\mathbf{q}}^k_{k+1})^{-1}(\mathbf{q}^w_{b_k})^{-1}\mathbf{q}^w_{b_{k+1}}\end{bmatrix}. \tag{8}$$

where $\mathcal{X} = \{\mathbf{R}_i, \mathbf{t}_i\}_{i=0,\ldots,N_w-1}$. The camera projection is then modified to account for the transformation between

cameras and the IMU. For a sparse feature $\mathbf{p}_j^w$, its projection function on camera i now becomes:

$$\pi(\mathbf{R}_b^c(\mathbf{R}_w^{c_i}(\mathbf{p}_j^w - \mathbf{t}_{c_i}^w) - \mathbf{t}_b^c)), \tag{9}$$

Since Eq. (3) is nonlinear with respect to $\mathcal{X}$, we use the Gauss-Newton method to perform the minimization. The key is that we have to pay attention to the initialization of $\mathcal{X}$. If it is not near to the ground truth value, the Gauss-Newton iterations will not converged. To alleviate optimization failures, we initialize $\mathcal{X}$ as follows:

- If IMUs are not available, we use the constant linear/angular velocity model to predict the latest camera pose. Other variables are initialized as the values obtained from the last time instant after the Gauss-Newton iterations are converged.
- If IMUs are available, we use the IMU propagation to predict the latest camera pose. Other variables are initialized as the values obtained from the last time instant after the Gauss-Newton iterations are converged.

Keyframe Selection After the sliding window-based bundle adjustment, this thread decides whether or not to insert a new keyframe according to the average feature parallax. The average feature parallax of two frames is computed as the average localization differences (in pixels) of matched image corner features between these two frames. If the average feature parallax between the latest keyframe and the second latest frame is greater than t_p ($t_p = 15$ in this work) pixels, a new keyframe is inserted and the sliding window shifts forward. Otherwise, the second latest frame is dropped and the sliding window is not moved.

Redundancy Removal Redundancy removal is an important step towards lifelong operations in large-scale environments. We remove redundant features and keyframes in the system:

1) Keyframe Culling: While keyframes in the local map are checked and culled in ORB-SLAM (Mur-Artal et al., 2015), we do not cull keyframes in the sliding window. We empirically find that early keyframe culling will lead to less localization accuracy and robustness (especially when dealing with large rotations). Keyframes are checked if and only if they are neither in the sliding window nor in the local dense maps (Sect. 3.3). We discard keyframes where 70% of the features have been seen in at least another three keyframes. We summarize the keyframe culling procedure in Algorithm 1.

2) Feature Culling: A sparse point is removed if it is neither seen in the latest frame nor observed in more than two keyframes. We check and cull features observed in removed keyframes in the keyframe culling step.

Algorithm 1 Keyframe Removal: the set of keyframes to be checked s_f, the current frame f.

if f is a new keyframe, **then**
  Push f in the back of s_f;
end if

$f_c \leftarrow$ null pointer;
if s_f is not empty, **then**
  $f_k \leftarrow$ the front of s_f;
  if f_k is not in the sliding window or not in the dense local reconstuction, **then**
    $f_c \leftarrow f_k$;
  end if
end if

if f_c is not a null pointer, **then**
  erase the front of s_f;
  if 70% of features in f_c have been seen in at least other three keyframes, **then**
    cull f_c.
  end if
end if

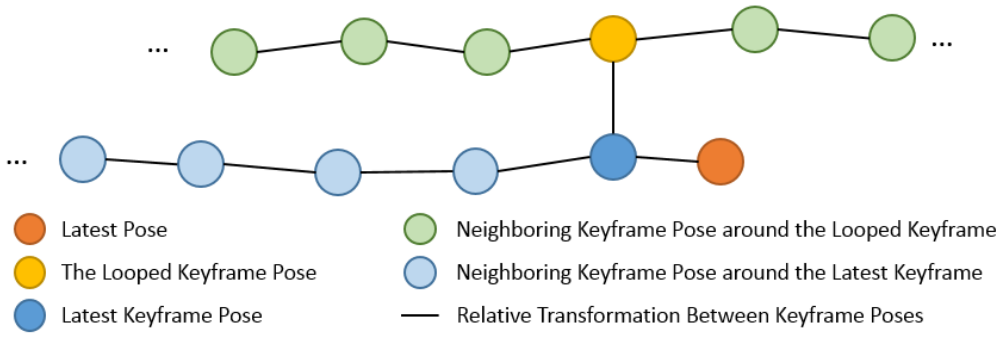


Figure 6: The local pose graph optimization. Neighboring keyframe poses around the loop keyframe (if found) and the latest keyframe are optimized for the local consistency.

Local Pose Graph Optimization If a loop is closed in the loop closing thread (Sect. 3.1.3), a pose graph optimization (Grisetti et al., 2010) is performed in a local window for the local consistency of poses (Fig. 6). The local map reconstruction in Sect. 3.3 requires locally consistent maps for motion planning. We set the size of this local pose graph optimization window to be the same size as the local map reconstruction window. Estimated variables are the latest keyframe pose, the looped keyframe pose, and the neighboring keyframe poses around the looped/latest keyframe. Constraints are 6-DOF relative transformations between these poses, which are obtained in the sliding-window-based bundle adjustment step and the loop closing step (Sect. 3.1.3). The latest pose is fixed during optimization. After the optimization is done, these estimated poses are sent to the local map reconstruction thread for reconstructing local dense maps.

Note that, the local pose graph optimization here is for the local consistency, while the global pose graph optimization

(Sect. 3.1.3) is for the global consistency. The window size of the local pose graph optimization is much smaller than that of the global pose graph optimization. Therefore, the time delay caused by the global pose graph optimization will not affect the local consistency of local poses around the latest robot pose, which is important for the real-time local map reconstruction for motion planning (Sect. 3.3).

3.1.3 Loop Closing

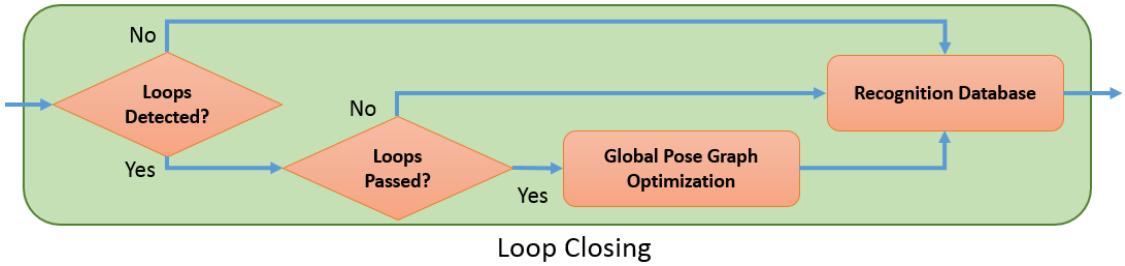


Figure 7: The pipeline of the loop closing thread. This thread searches possible loops for every new keyframe using a bag-of-words place recognizer (DBoW2 (Gálvez-López and Tardós, 2012)). Keyframe poses and feature positions are refined by a global pose graph optimization if loops are closed. To satisfy the real-time processing requirement of the whole system, we lower the priority of this thread.

The loop closing thread (Fig. 7) closes possible loops for every new keyframe using a bag-of-words place recognizer (DBoW2 (Gálvez-López and Tardós, 2012)). If a new keyframe is inserted, we extract FAST image corner features (typically 500 image corner features for every keyframe) at 8 levels of the scale with a scale factor of 1.2 on this keyframe. ORB descriptors are then computed on the retained FAST image corner features. These ORB descriptors are then quantized to be integer numbers by a pre-trained vocabulary tree for fast indexing. We use these numbers as the abstraction of keyframes. If two keyframes are similar, the distributions of the quantized numbers of these two keyframes are similar. We select 10 loop candidates whose distributions of quantized numbers are the most similar to that of the new inserted keyframe. We use the Lucas-Kanade sparse optical flow from the loop candidate keyframe to the new inserted keyframe to establish feature correspondences between them. A geometry check via the robust estimator RANSAC (Fischler and Bolles, 1981) is used for outlier removal. If the number of inlier feature correspondences is greater than 60, we pass it to the temporal consistency check. That is, a loop candidate is regarded as a loop keyframe if and only if its neighboring keyframes are consistently looped with the neighboring keyframes of the inserted new keyframe. We declare loop candidates as accepted if they pass the above tests. Observations as well as 3D positions of matched features are merged. More details about the bag-of-words place recognizer DBoW2 can be found in (Gálvez-López and Tardós, 2012).

We do not use the same features for the tracking thread, the complexity-bounded optimization thread, and the loop closing thread as in ORB-SLAM (Mur-Artal et al., 2015). Instead, features in the feature tracking thread and the

complexity-bounded optimization thread are different from features in the loop closing thread. On one hand, this loosely coupled principle makes the whole system more ready for modular programming, such that recent advances in the visual odometry and the loop detection (especially the deep learning-based solution) can be easily integrated. On the other hand, features of the loop detection are used for the visual distinctiveness within the whole captured video sequence, while features of the visual odometry are used for the tracking performance. These two purposes are different.

A global pose graph optimization is applied to refine poses if loops are closed. Variables are all keyframe poses. Constraints are all 6-DOF relative transformations between consecutive keyframe poses as well as all 6-DOF relative transformations between looped keyframe poses. The difference between the local pose graph optimization (Sect. 3.1.2) and the global pose graph optimization here is in the window size. While the former optimizes poses around the latest robot pose, the latter refines all poses in the system. Note that, to satisfy the real-time processing requirement, the loop closing thread is performed in a low-priority thread.

3.2 Map Integration

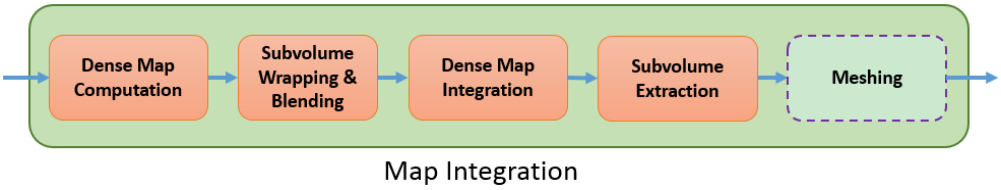


Figure 8: The pipeline of the map integration thread. This thread fuses past maps with latest computed dense maps.

Given estimated keyframe poses and captured stereo image sequences, the map integration thread fuses maps estimated in the past with the latest estimated dense maps by stereo matching. Steps in this thread are shown in Fig. 8. Intuitively, we firstly compute the latest dense depth map using the stereo semi-global matching (**SGM** (Hirschmuller, 2008)). This dense depth map is then merged with existing sub-maps obtained by subvolume wrapping (Sect. 3.2.4) via an improved volumetric fusion proposed in this work to take the uncertainty of stereo matching into account. After maps are merged, we extract well-reconstructed subvolumes that are out of the local window and attach them to nearby poses for the next round subvolume wrapping and blending. Finally, the meshing, an optional step, is conducted for visualization proposes.

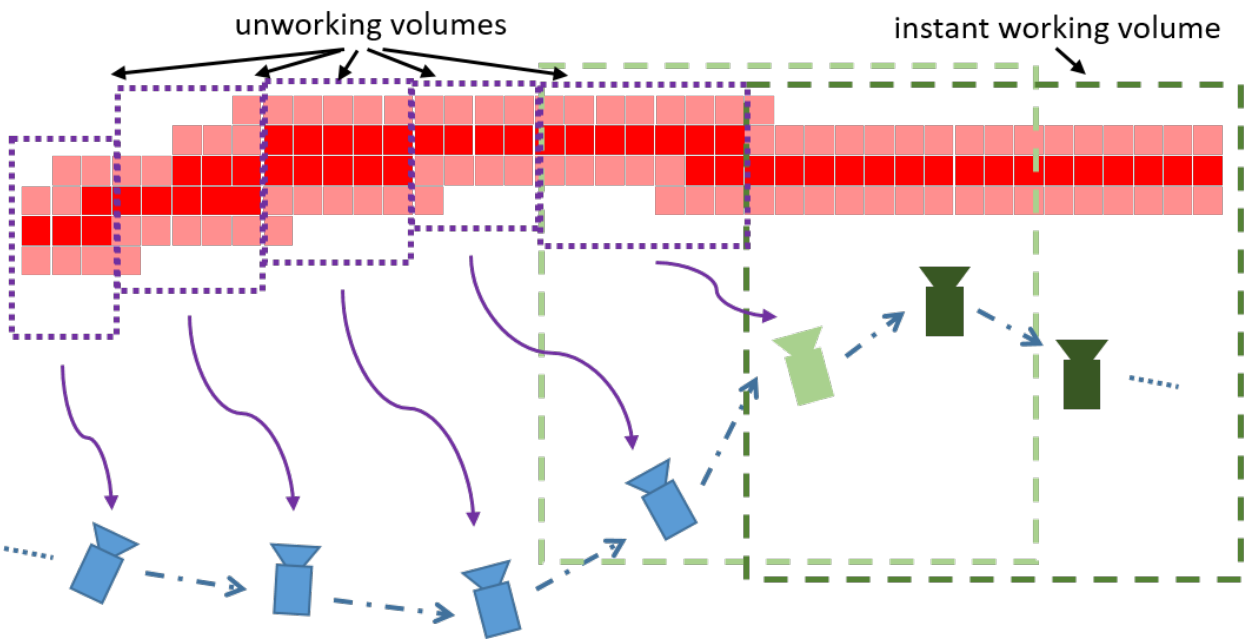


Figure 9: The proposed map representation. This consists of two parts: an instant working volume centered on the latest keyframe pose (the dashed dark green box); and a collection of unworking volumes, which consists of TSDF subvolumes that are out of the instant working volume (the dashed purple boxes). Unworking volumes are attached to the nearest keyframes once extracted (Sect. 3.2.5).

3.2.1 Dense Map Representation

For the flexibility of the map deformation, we subdivide the map into two parts. Fig. 9 gives a detailed illustration of this representation. The first part is an instant working volume. It contains a set of voxels within a fixed cube (parametrized by a constant l_c) centered on the latest keyframe pose (the dashed dark green box). The other part is a collection of unworking volumes, which consists of subvolumes that are out of the instant working volume (the dashed purple boxes). Unworking volumes are attached to the nearest keyframes once extracted (Sect. 3.2.5). The map representation can be seen as a combination of metric maps and topological maps. The instant working volume is responsible for the depth map integration to remove noises as well as compress data. Unworking volumes are responsible for the map deformation to ensure the map consistency if loops are closed.

This map representation is motivated by the observation that keyframe poses estimated from a sparse feature based SLAM pipeline are locally accurate. Built on top of these estimated keyframe poses, the fused dense map using the improved volumetric fusion is also locally accurate. We assume the relative metric distance between keyframe poses and the fused dense map within a local window (i.e., the instant working volume) is preserved no matter how keyframe poses are shifted after loop closures.

3.2.2 Depth Computation

After a new keyframe is inserted, we compute the dense depth map via a GPU-based stereo semi-global matching (**SGM**) (Hernandez-Juarez et al., 2016). Semi-global matching (Hirschmuller, 2008) is a method proposed by Heiko Hirschmueller for stereo depth estimation. Instead of computing each depth estimate independently, this method explicitly enforces spatial correlations of neighboring depth estimates by aggregating evidence from different directions. As a result, depth estimation ambiguities in textureless regions or repetitive-texture regions can be alleviated. A fast method to compute the solution based on the dynamic programming is also proposed for the computation efficiency. Due to the limitation of the paper, readers are directed to reference (Hirschmuller, 2008) for more details about the semi-global matching. The computed dense depth map will then be integrated into a volumetric 3D model (Sect. 3.2.3).

In contrast to the previous works, (Engel et al., 2014) and (Schoeps et al., 2015), which ignore potentially unreliable depth measurements, we integrate all depth values into the volumetric model. This is mainly for the safety of mobile robots, as potential outliers in the dense map may lead to a smaller feasible space for motion planning, while potential missing obstacles may lead to collisions with objects or unexpected disasters.

3.2.3 Depth Integration

While the SGM utilizes the spatial correlation of images, the volumetric fusion further improves the mapping quality by leveraging the temporal information of captured images. We model the world as a 3D array of cubic voxels. Each voxel is associated with a signed distance function (**SDF**) $\phi(\mathbf{x}) : \mathbb{R}^3 \rightarrow \mathbb{R}$ and a weight $w(\mathbf{x}) : \mathbb{R}^3 \rightarrow \mathbb{R}$. SDF $\phi(\mathbf{x})$ is defined as the signed distance between $\mathbf{x}$ and the nearest object surface, and it is positive if it is outside an object, and negative otherwise. By this definition, surfaces of objects are the zero crossing of the signed distance function (i.e., $\phi(\mathbf{x}) = 0$). $w(\mathbf{x})$ represents the confidence of the input distance measurement. As shown in (Curless and Levoy, 1996), averaging the input distance measurements with respective variances over time results in minimizing the weighted sum of the square distances to all the ray endpoints for the zero isosurface of the SDF.

Since the major part of the 3D world is usually empty, we use a hash table to index voxels (Klingensmith et al., 2015; Nießner et al., 2013). Only SDFs as well as weights that are near the object surfaces are stored. These SDFs are called truncated signed distance functions (**TSDF**s):

$$\phi_r(\mathbf{x}) = \begin{cases} \phi(\mathbf{x}), & \text{if } ||\phi(\mathbf{x})|| \leq r \\ \text{undefined}, & \text{otherwise} \end{cases}, \tag{10}$$

where r is the truncated distance threshold. For a given depth measurement d, obtained by projecting depth estimates to the world coordinate using estimated poses, with corresponding ray vector direction $\mathbf{f}$, we classify this ray into three regions(Klingensmith et al., 2015):

$$u \cdot \mathbf{f} \in \begin{cases} \text{hit region}, & \text{if } ||u - d|| \leq r \\ \text{space carving region}, & \text{if } u \leq d - r \\ \text{undefined}, & \text{otherwise} \end{cases} . \tag{11}$$

Voxels within the hit region are updated with the standard volumetric fusion equation (Curless and Levoy, 1996) to eliminate noises:

$$\phi_r(\mathbf{x})' = \frac{\phi_r(\mathbf{x}) \cdot w(\mathbf{x}) + \delta d \cdot \alpha(\delta d)}{w(\mathbf{x}) + \alpha(\delta d)} \tag{12}$$

$$w(\mathbf{x})' = w(\mathbf{x}) + \alpha(\delta d), \tag{13}$$

where $\delta d = \mathbf{x} - d \cdot \mathbf{f}$ and $\alpha(\delta d)$ is the corresponding variance. $\alpha(\delta d)$ is a constant in (Curless and Levoy, 1996). However, in this work, we take into account the fact that errors of depth are sourced from the disparity computation via the stereo SGM. We thus model $\alpha(\delta d)$ as follows:

$$\alpha(\delta d) = (\frac{\partial \delta d}{\partial id})^2 \cdot \sigma_{id}^2 = (\frac{1}{id})^2 \cdot \sigma_{id}^2, \tag{14}$$

where id is its inverse depth value with variance σ_{id}^2 ($\sigma_{id}^2 = 1$ in this work). Here $\alpha(\delta d)$ is a first-order approximation of its inverse depth variance. The initial condition of the SDF is $\phi_r(\mathbf{x}) = constant$ and $w(\mathbf{w}) = 0$.

Voxels within the space carving region are chiseled away. This operation can be viewed as removing potential depth outliers by visibility constraints (i.e., segments between two endpoints of a ray are empty). Free-space carving makes sense for the reason that we care more about which part of the scene does not contain surfaces (for motion planning) than what is inside the objects.

For efficiency of the volumetric fusion, we divide the world into a two-level tree (Klingensmith et al., 2015; Nießner et al., 2013). There are chunks in the first level, and each chunk consists of a fixed grid of N_v^3 ($N_v = 4$ in this work) voxels, which are stored in a monolithic memory block. We index chunks in a spatial 3D hash map on integer coordinates and allocate them from a growing pool of heap memory when they are firstly visited. The complexity of queries and updates on this two-level tree structure is $\mathcal{O}(1)$ thanks to the efficient hash-table lookup. In addition, to

improve the cache performance, we store voxels within chunks in a continuous way.

Note that, in the volumetric fusion, we assume that keyframe poses are accurate. This is not true for non-keyframe poses. They are not well-estimated due to the lack of sufficient parallax. We do not integrate depth maps computed on non-keyframes.

3.2.4 Subvolume Wrapping & Blending

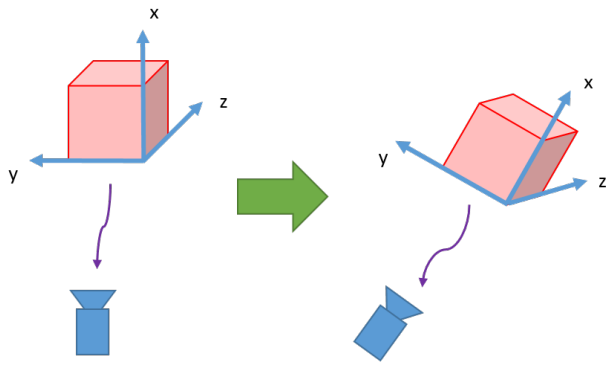


Figure 10: The subvolume wrapping after loop closure. We translate the chunk origin and rotate their axes according to the shifted pose corrections of the attached poses after loop closure.

If revisited places are detected and loops are closed, we optimize all robot poses via a local pose graph optimization for a fast response of the pose correction (Sect. 3.1.2). Positions of subvolumes are updated according to the pose shift of attached keyframes. Fig. 10 shows how we update chunks in subvolumes after loop closure. We translate chunk origins and rotate their axes according to the shifted pose correction of the attached keyframe. Some refined subvolumes may overlap with the working volume. We blend them with the working volume:

$$\phi_r(\mathbf{x})' = \frac{\phi_r(\mathbf{x}) \cdot w(\mathbf{x}) + \phi_p(\mathbf{x}) \cdot w_p(\mathbf{x})}{w(\mathbf{x}) + w_p(\mathbf{x})} \tag{15}$$

$$w(\mathbf{x})' = w(\mathbf{x}) + w_p(\mathbf{x}), \tag{16}$$

where $\phi_r(\mathbf{x})$, $w(\mathbf{x})$, $\phi_p(\mathbf{x})$ and $w_p(\mathbf{x})$ are the TSDF in the working volume, the weight in the working volume, the projected TSDF, and the projected weight, respectively.

Fig. 11 illustrates how the subvolume blending works before and after loop closure. Poses in orange are closed with loops. Subvolumes in dashed purple boxes overlap with the instant working volume (i.e., dashed dark green box) after the pose correction. These are blended with the instant working volume. The subvolume blending not only removes redundancy on overlapping places to make the overall system more memory-efficient, but also provides the

maximum information that a robot has 'experienced' around the latest keyframe pose by the map reuse. Changes in the environment can also be taken into account in this subvolume blending.

Since the subvolume blending only involves subvolumes that are overlapped with the working volume, its complexity is bounded and does not scale with respect to the whole map size. Note that, as shown in Fig. 8, this step should be performed before the depth map integration step (Sect. 3.2.3). Projected volumes may contain outliers caused by environment changes or sensor measurements. The depth map integration step (Sect. 3.2.3) may remove these outliers while integrating the latest depth maps.

3.2.5 Subvolume Extraction

As the window of the instant working volume moves along with the robot movement (from the dashed light green box to the dashed dark green box in Fig. 9), some parts of the instant working volume may be out of the working window. We extract out-of-window subvolumes and attach them to the nearest keyframe poses (see the dashed purple boxes and solid purple arrows in Fig. 9). The extracted information includes origins of chunks represented by a relative transform to the attached keyframe poses, voxels in chunks, and three orthogonal axes of chunks that indicate the chunk directions. This extracted information is needed in the subvolume blending (Sect. 3.2.4).

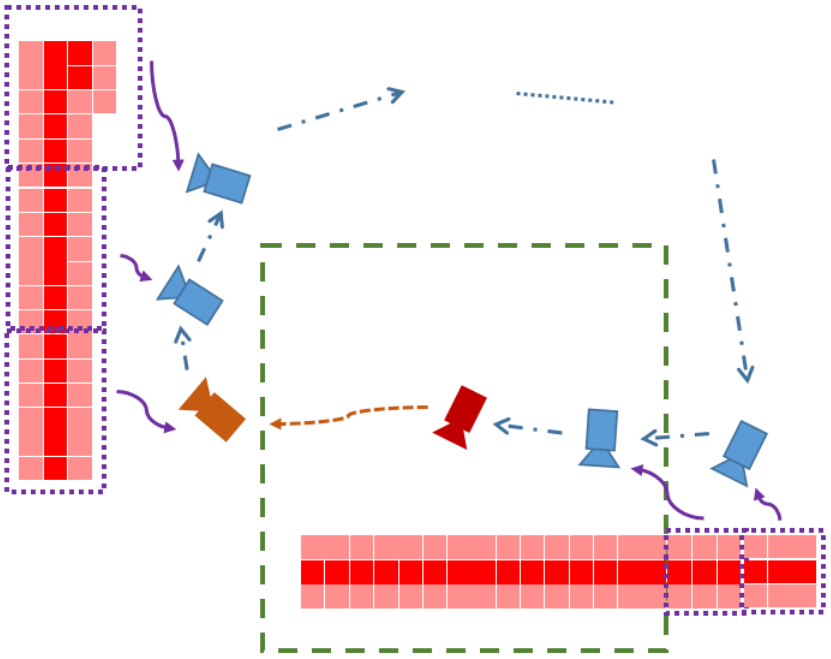
In contrast to previous works (Whelan et al., 2016; Whelan et al., 2014) that perform surface meshing and drop all voxels, we save all the raw data so as to handle environment changes when we revisit past places.

3.2.6 Meshing

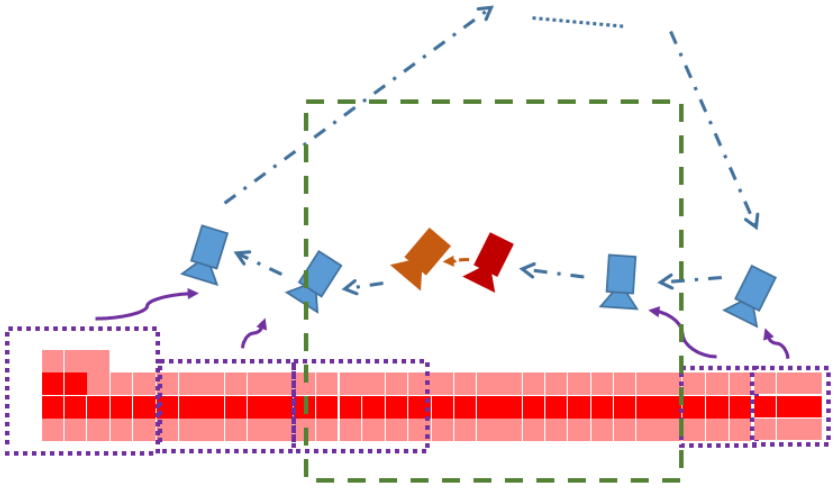
While voxels are sufficient for motion planning (Ratliff et al., 2009; Oleynikova et al., 2016), colors and textures are more suitable for human interactions and visual debugging. We include an optimal step, the marching cubes (Lorensen and Cline, 1987), to extract polygonal meshes of an isosurface from the SDFs.

3.3 Local Map Reconstruction

The local map reconstruction thread reconstructs local dense maps around the latest keyframe pose for motion planning. While the number of keyframes and the size of the built map in the system will scale with respect to the travel time/distance, the computation time of map reconstruction has to be bounded for real-time applications. This requires us to limit the overall complexity of this thread. The good news for robotics applications is that motion planning is done locally and sequentially. Full dense reconstructions are not needed at each time instant. Instead, sub-maps



(a) Before closing a loop.



(b) After closing a loop.

Figure 11: The subvolume wrapping and blending (a) before and (b) after closing a loop. The latest keyframe is shown in red, while the loop keyframe is shown in orange. After closing a loop, subvolumes attached to the loop keyframe are wrapped (Fig. 10) and blended with the instant working volume (dashed green box).

around the latest robot pose are enough. We do not reconstruct full dense maps as in (Fioraio et al., 2015; Whelan et al., 2014; Whelan et al., 2016) because this is infeasible for a real-time robotics system with limited computations. In addition, the algorithmic complexity of motion planning algorithms usually scales with respect to the map size. Planing motions in large maps is not practical. Local dense maps are the maximum information about the perceived environment around the instant robot pose regarding what a real-time system is able to handle.

We construct local dense maps for motion planning. Each local dense map consists of the instant working volume and subvolumes attached to local keyframes (Fig. 9). The set of local keyframes contains not only temporally correlated keyframes but also spatially correlated keyframes linked by loop closures. The number of local keyframes n_l in the local dense map reconstruction is determined according to the computation time of this thread and the used motion planning algorithm. Fig. 11 illustrates what a local dense map looks like before and after loop closure. Local dense maps in various datasets and real-world experiments are shown in the experiment section of this paper (Fig. 18, Fig. 19, Fig. 28, and Fig. 31).

If planning destinations are within the reconstructed local maps, traditional planning algorithms, such as the point cloud based planning method in (Gao and Shen, 2016b), can be used. Conversely, if planning destinations are not in local maps, we apply a heuristic strategy:

- 1) We use motion planning algorithms to plan paths from the instant robot pose to the expected destination, by assuming space that is out of the reconstructed local map is empty. We then calculate the total length of the planned path, which is denoted as l_p.
- 2) Starting from the instant robot pose, we use a breath-first search algorithm to traverse nodes that are not in the reconstructed local maps and whose distances are smaller than l_p (to bound the algorithmic complexity), in the pose graph. We denote the searched set of nodes as Φ_p. We compute the distances between nodes in Φ_p and the expected destination. If the minimum of the computed distances is smaller than l_p, we choose the path from the current node to the node with the minimum computed distance on the pose graph. Otherwise, we choose the path planned in step 1.

This heuristic strategy can be viewed as a mix of topological planning on the global pose graph and the local motion planning using the reconstructed local map.

4 Implementations

4.1 Memory Management

Though we have the redundancy removal step in the complexity-bounded optimization thread, the used memory after long-term autonomy will ultimately be very large such that onboard systems are not able to cope with it. Thus, we propose an algorithm to drop "old" keyframes. The assumption is that the "newer" the keyframes are the more likely they will be reused. The algorithm is summarized in Algorithm 2. We push a new inserted keyframe f_k into a keyframe

list l_k (Line 1). For temporal and spatial neighboring keyframes of f_k, we update their timestamp as the timestamp of f_k (Line 3–6). We then check whether or not the size of l_k is greater than the maximum number of keyframe thresholds m_f. If yes, we erase the oldest keyframe. Their attached subvolumes and related visual SLAM features are also removed (Line 8–15). The goal of memory management is to save as much information that is likely to be reused as possible. Sect. 5.2.3 shows how memory management affects the system performance.

Algorithm 2 Memory Management: a new inserted keyframe f_k, the keyframe list l_k, the maximum number of keyframes m_f.

```
1:  push f_k into l_k.
2:
3:  F_k ← search the neighborhood keyframes of f_k
4:  for each keyframe f_n in F_k do
5:      set the timestamp of f_n to be the timestamp of f_k
6:  end for
7:
8:  while the size of l_k is greater than m_f do
9:      f ← the oldest keyframe in m_f;
10:     delete subvolumes attached in f;
11:     erase f;
12:     for each feature p observed in f do
13:         delete p if it is not observed in at least three keyframes.
14:     end for
15: end while
```

4.2 Asynchrony & Multithreading

We fix the output frequency of the feature tracking thread to be 10 Hz (maximum frequency). Other threads are varied according to the CPU and GPU payload. In particular, if threads are free, they will process the input data if notified signals are given by the data-dependent threads. Otherwise, if threads are busy, they will put the input data into data buffers. However, if the buffer size is more than 2, the new arrived data will be dropped. This mechanism is to satisfy the real-time processing requirement of online systems. The data processing frequency has to be at least equal to the input data frequency such that data buffers will not grow unboundedly as the system runs.

5 Experiments

The whole system is implemented in C++, with the ROS as the interfacing robotics middleware. We use the ceres solver for all the optimizations involved[1]. All testings are carried out in a commodity Lenovo laptop Y50 with an i7-4720HQ CPU (2.6 GHz), a GTX-960M GPU (3 GB), and a 16 GB RAM. We have performed extensive experimental

[1] http://ceres-solver.org/

validations of our system on the KITTI dataset and the NewCollege dataset, as well as through a real-time online experiment around our university campus.

5.1 KITTI Dataset

The benchmark from the KITTI dataset consists of sequences of a car driven around a residential area, with stereo images and ground truth position from the GPS and ground truth depth maps provided by a Velodyne laser scanner. Our system processes all sequences at the same frame rate at which they are captured (10 Hz). Among the sequences, there are moving objects (cars, bikes, walking people, etc.) in the captured images. Our system is able to track all the testing sequences. The sliding window size N_w of our system is 16. The maximum number of features M_w is 400. If M_w is greater than 400, we prefer features with longer tracked lengths. The window length of the instant working volume l_c is 20, while the number of neighborhood keyframes n_l is 150.

5.1.1 Localization Accuracy

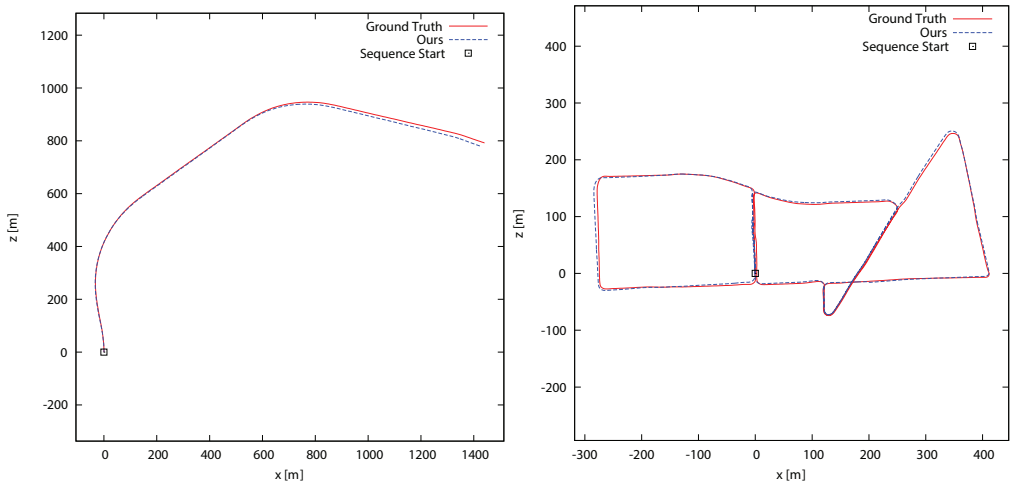
We evaluate our SLAM pipeline performance according to the guideline of the official KITTI website. [2] Sequences from 11 to 21 are tested. This evaluation computes translational and rotational error. Part of the whole ranklist available at the time of this paper's submission is shown in Table 1. As expected, ORB-SLAM2 (Mur-Artal and Tardós, 2017) works better than our complexity-bounded SLAM (CBSLAM), because it has a much larger bundle adjustment window size. Generally, the size of the bundle adjustment window is 2–3 times bigger than that of ours. It takes about 0.3 to 0.4 seconds for ORB-SLAM2 to do the local bundle adjustment, while our complexity-bounded SLAM finishes the local bundle adjustment within 0.05 seconds. The average translation error of our complexity-bounded SLAM is 1.24%. The average rotation error is 0.0029 degrees per meter. Our SLAM performs better than FRVO (Wu et al., 2017), SSLAM (Bellavia et al., 2015), and eVO (Sanfourche et al., 2013), both for the translation and rotation accuracy.

Table 1: The statistics of localization errors on KITTI testing sequences (11–21).

Method	Translation Error	Rotation Error
ORB-SLAM2 (Mur-Artal and Tardós, 2017)	1.15 %	0.0027 [deg/m]
Ours (CBSLAM)	1.24 %	0.0029 [deg/m]
FRVO (Wu et al., 2017)	1.26 %	0.0038 [deg/m]
SSLAM (Bellavia et al., 2015)	1.57 %	0.0044 [deg/m]
eVO (Sanfourche et al., 2013)	1.76 %	0.0036 [deg/m]

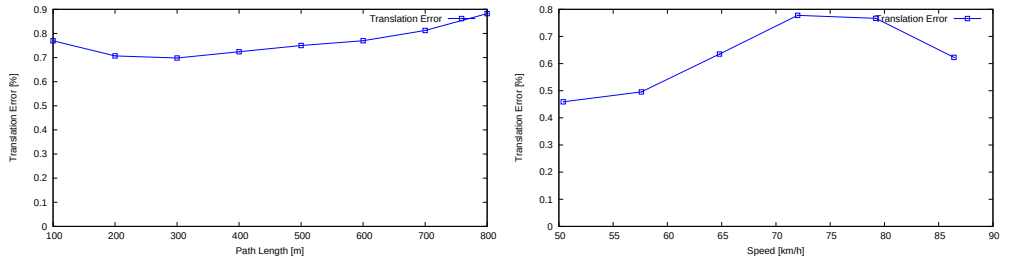
The pose estimation from our CBSLAM is locally accurate. It drifts slowly if no loops are detected (Fig. 12(a)). A

[2] http://www.cvlibs.net/datasets/kitti/eval_odometry.php



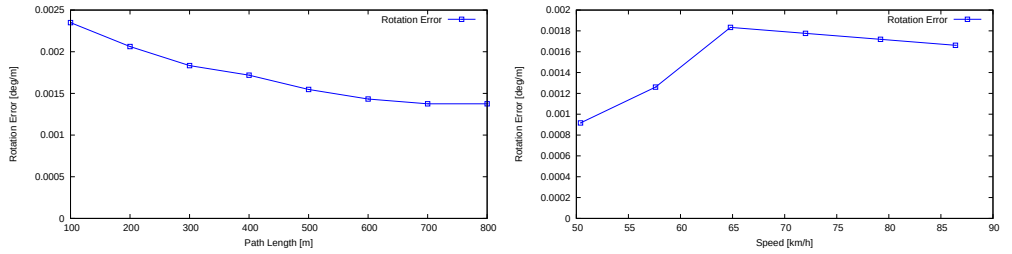
(a) The estimated trajectory on the KITTI 12 sequence (no loops).

(b) The estimated trajectory on the KITTI 13 sequence (multiple loops).



(c) The translation error w.r.t the path length on the KITTI 12 sequence.

(d) The translation error w.r.t the speed on the KITTI 12 sequence.



(e) The rotation error w.r.t the path length on the KITTI 12 sequence.

(f) The rotation error w.r.t the speed on the KITTI 12 sequence.

Figure 12: The localization error of our CBSLAM on the KITTI 12 & 13 sequence. (a) It drifts slowly if no loops are detected. (b) Drift is corrected if loops are closed. (c–f) The localization errors at different path lengths and speeds.

locally accurate pose estimation is vitally important for the local map reconstruction thread to build the dense 3D maps because the depth map integration on the instant working volume is conditioned on the estimated keyframe poses. We use the KITTI 12 sequence for a detailed illustration of how our CBSLAM performs locally with respect to different

path lengths and speeds. As shown in Fig. 12(c)–(f), the pose drift of our CBSLAM is not significant at path lengths less than 1 km and speeds less than 100 km/h. If we revisit past places, our CBSLAM is able to close loops and correct pose drift (Fig. 12(b)).

5.1.2 Quantitative Analysis of Online Mapping Performance

In real-time robotics applications, key factors are the mapping accuracy, mapping density, and outlier rate. While we prefer a higher mapping accuracy, the mapping density and the outlier rate are bases of the motion planning. On one hand, the first concern of the motion planning is safety. To this end, a denser map helps a safer planning. On the other hand, the outlier rate relates to the efficiency of motion planning. Outliers on maps may alter the expected path or even block the expected path. We have to balance all these three key factors for real-time robotics applications. We adopt a conservative strategy in this paper. The priority of these factors is: mapping density $>$ outlier rate $>$ mapping accuracy.

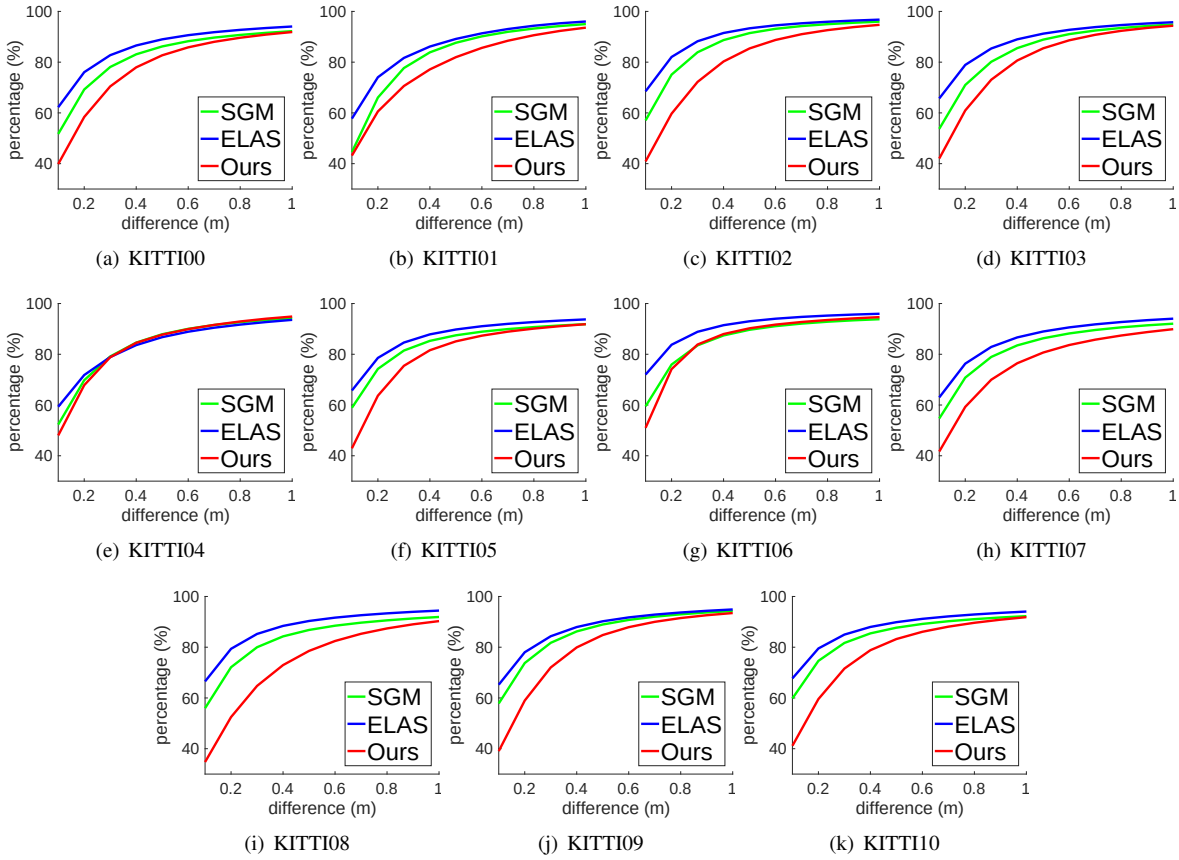
Testing sequences from 00 to 10 are used. The ground truth depth is provided by the Velodyne laser scanner data. The voxel size is set to be 0.3 meters for all experiments. The average scene depth of the KITTI sequences is 15.0 meters. Four measurement metrics are used: the depth difference percentage, the safe margin percentage, the outlier percentage, and the mapping density. We project built dense maps and Velodyne laser scanner measurements into keyframe views to evaluate the mapping performance. Suppose N_p is the number of valid depth values in the depth map, N_k is the total number of depth values in the depth map, d_i^m is the reconstructed depth measurement, and d_i^v is the corresponding Velodyne laser scanner data. We have

1) The depth difference percentage is the percentage of depth differences within a certain range threshold t_d (Perentage-diff(t_d)):

$$\text{Perentage-diff}(t_d) = \frac{\sum_i^{N_p} \{\text{abs}(d_i^m - d_i^v) < t_d\}}{N_p} \times 100. \tag{17}$$

The depth difference percentage measures the exact mapping accuracy. We do not use the disparity comparison adopted by most vision-based algorithms, since for real-world motion planning, depth values instead of disparities are used. We enumerate the depth difference t_d from 0.67% (0.1 meters) to 6.7% (1.0 meters) of the average scene depth, and calculate the percentage of depth difference within the threshold range. We compare our reconstructed maps with depth maps calculated by SGM (Hernandez-Juarez et al., 2016) and ELAS (Geiger et al., 2010). Fig. 13 shows the results. The mapping accuracy of all algorithms are similar at the locations of keyframe views if the depth difference range is larger than 0.5 meters. For a small depth difference range (say smaller than 0.3 meters), our system performs

the worst. This is because the voxel size of our mapping algorithm is 0.3 meters. The size of the voxels makes a big difference in the details of the reconstructed 3D maps.



(a) KITTI00 (b) KITTI01 (c) KITTI02 (d) KITTI03

(e) KITTI04 (f) KITTI05 (g) KITTI06 (h) KITTI07

(i) KITTI08 (j) KITTI09 (k) KITTI10

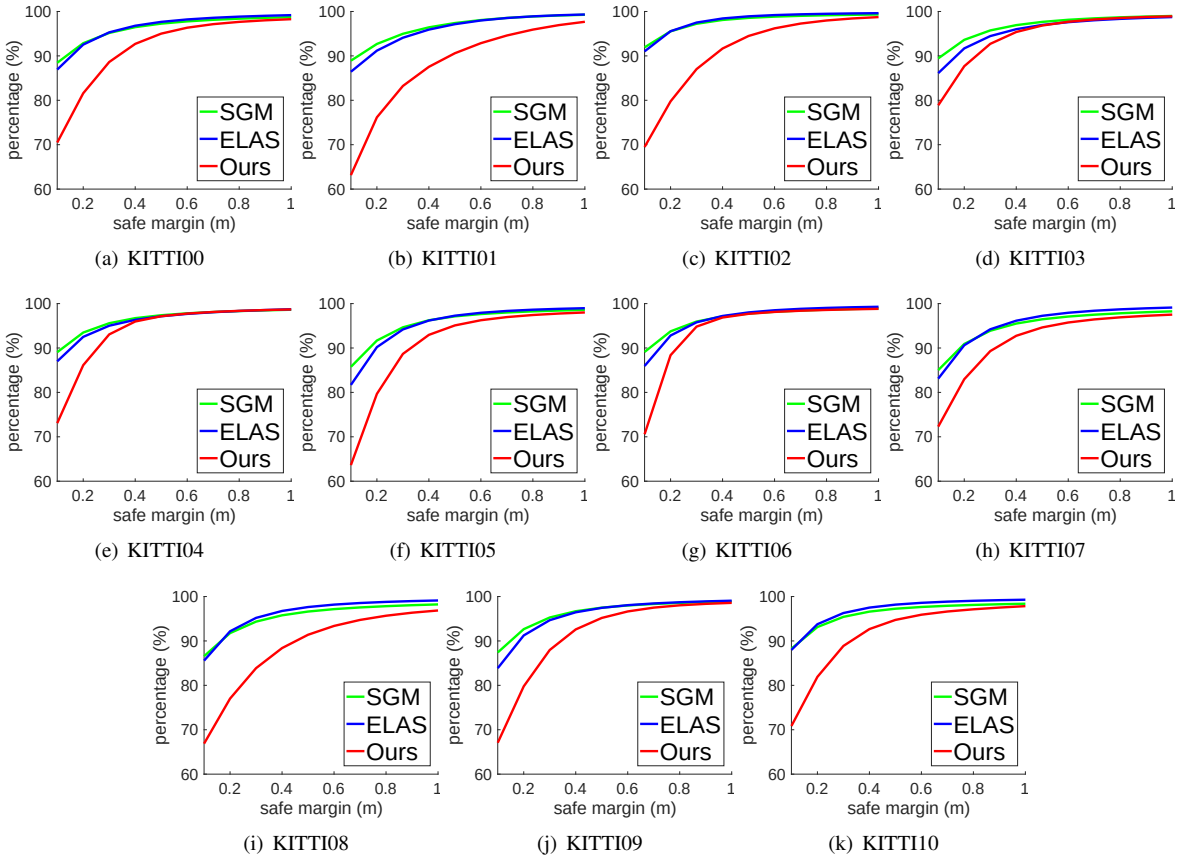
Figure 13: The depth difference percentage on the KITTI dataset. We enumerate depth differences from 0.67% (0.1 meters) to 6.7% (1.0 meters) of the average scene depth, and calculate the percentage of depth differences within a given difference threshold. The horizontal axis is the difference threshold, while the vertical axis is the corresponding percentage. Results from our approach, SGM (Hernandez-Juarez et al., 2016) and ELAS (Geiger et al., 2010) are plotted for comparison.

2) The safe margin percentage is the percentage of depth values that are less than the ground truth depth values given a certain margin threshold s_d (Perentage-safe(s_d)):

$$\text{Perentage-safe}(s_d) = \frac{\sum_i^{N_p} \{d_i^m < d_i^v + s_d\}}{N_p} \times 100. \tag{18}$$

The safe margin percentage measures how "safe" the mapping is given a certain margin threshold (which is involved in the obstacle expansion step of most obstacle avoidance algorithms). This is crucial for autonomous navigation, which takes safety into account. Safe margins s_d start from 0.67% average scene depth (0.1 meters) and end at 6.7% average scene depth (1.0 meters). The estimated depth, which is less than the ground truth depth plus the safe margin,

is supposed to be the safe depth. Results are shown in Fig. 14. Again, our mapping approach performs the worst at safe margins less than 0.3 meters due to the limitation on the voxel size, while at safe margins larger than 0.5 meters, all algorithms achieve similar performances.



(a) KITTI00 (b) KITTI01 (c) KITTI02 (d) KITTI03

(e) KITTI04 (f) KITTI05 (g) KITTI06 (h) KITTI07

(i) KITTI08 (j) KITTI09 (k) KITTI10

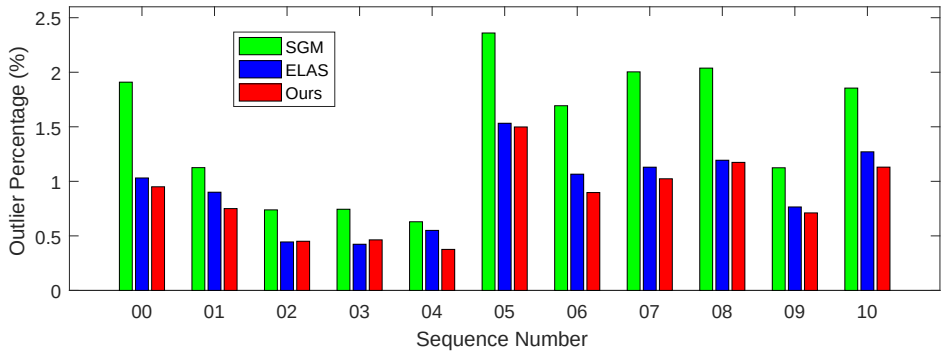
Figure 14: The safe margin percentage on the KITTI dataset. A depth estimate, which is less than the ground truth depth value plus the safe margin, is supposed to be a safe depth estimate. The horizontal axis is the safe margin, while the vertical axis is the corresponding percentage. Results from our approach, SGM (Hernandez-Juarez et al., 2016) and ELAS (Geiger et al., 2010) are plotted for comparison.

3) The outlier percentage is the percentage of depth that is less than the ground truth depth given a certain outlier threshold o_d (Perentage-outlier(o_d)):

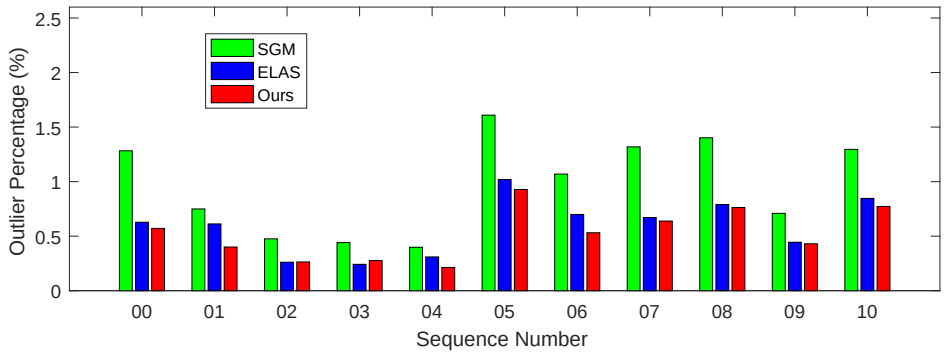
$$\text{Perentage-outlier}(o_d) = \frac{\sum_i^{N_p} \{d_i^m < d_i^v - o_d\}}{N_p} \times 100. \tag{19}$$

The outlier percentage measures how "useful" the instant mapping is given an outlier distance threshold. A smaller outlier percentage indicates a "clearer" space for motion planning. The outlier percentage affects the size of the feasible space for motion planning. We plot results obtained by setting outlier thresholds (o_d) to be = 3.0 or 4.0 in Fig. 15. Similar results are observed for larger outlier thresholds. The outlier percentage of our mapping approach is

similar to the result of ELAS, while SGM has the highest outlier rate.



(a) $o_d = 3$.



(b) $o_d = 4$.

Figure 15: The outlier percentage on the KITTI dataset. The horizontal axis is the sequence number of the KITTI dataset (from 00 to 10), while the vertical axis is the outlier percentage given an outlier threshold (o_d). The outlier percentage is the percentage of depth estimates that are less than the ground truth depth values given a certain outlier threshold (o_d = 3 or 4). Results from our approach, SGM (Hernandez-Juarez et al., 2016) and ELAS (Geiger et al., 2010) are plotted for comparison. Similar results are observed for other outlier thresholds.

4) The average mapping density (%) is the average density of depth estimates:

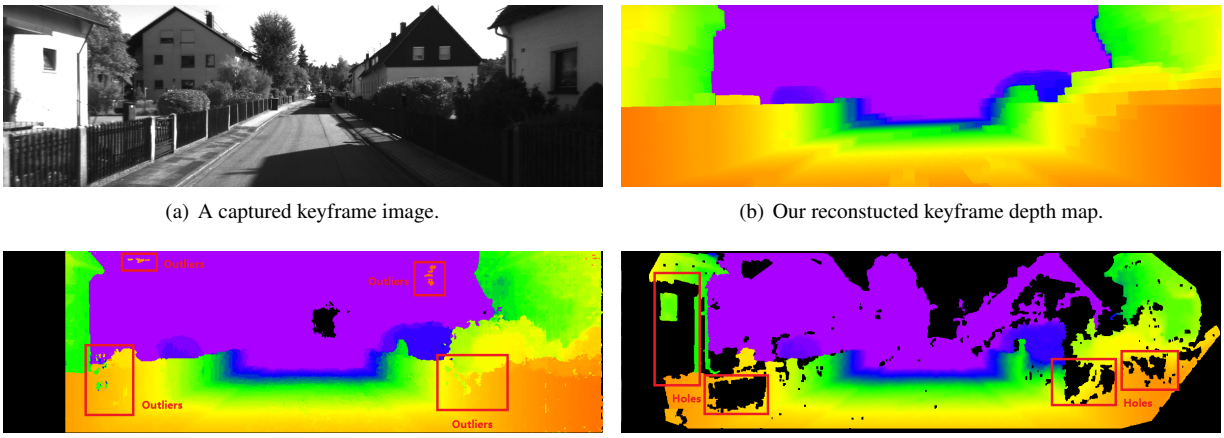
$$\text{density} = \frac{N_p}{N_k}. \tag{20}$$

The average mapping density measures how "complete" the mapping is. A denser mapping density is beneficial for safer motion planning. We denote the average mapping density of our approach as 100%. The average mapping density is evaluated using all testing sequences and is summarized in Table 2.

Table 2: The statistics of the average mapping density on KITTI testing sequences (00-10).

Method	**Ours**	**SGM (Hernandez-Juarez et al., 2016)**	**ELAS (Geiger et al., 2010)**
Density (%)	100%	84.3%	78.7%

Discussion It seems that using depth estimates from ELAS is the best choice in terms of the depth difference percentage (Fig. 13), the safe margin percentage (Fig. 14), and the outlier percentage (Fig. 15), but a detailed look at the reconstructed depth maps leads to a different conclusion. Details can be seen in Fig. 16. Pixels without depth measurements are marked in black. Colors of pixels vary with respect to the distances to the environment surface (from orange to purple). ELAS only generates depth values that are well estimated. This is the reason why it performs well on the depth difference percentage, the safe margin percentage, and the outlier percentage. The completeness of depth maps computed by ELAS is much less than that of our reconstructed maps. Our mapping approach provides the most dense maps on keyframe views, and achieves a similar performance to ELAS and SGM at certain thresholds ($t_d > 0.5$, $o_d > 3.0$) or margins ($s_d > 0.5$). In addition, depth maps from SGM contain notable outliers, while our reconstructed maps are "clearer" for motion planning.



(a) A captured keyframe image.

(b) Our reconstucted keyframe depth map.

(c) The depth map calculated by SGM.

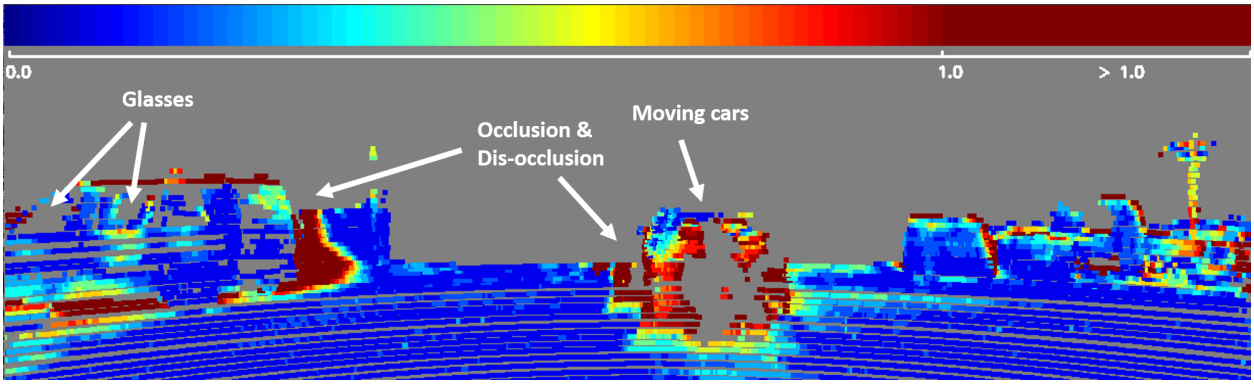
(d) The depth map calculated by ELAS.

Figure 16: The qualitative mapping completeness on the KITTI dataset. Pixels without depth measurements are marked in black. Colors of pixels vary with respect to the distance to the environment surface. (a) a captured keyframe image. (b-d) depths from our approach, SGM and ELAS, respectively.

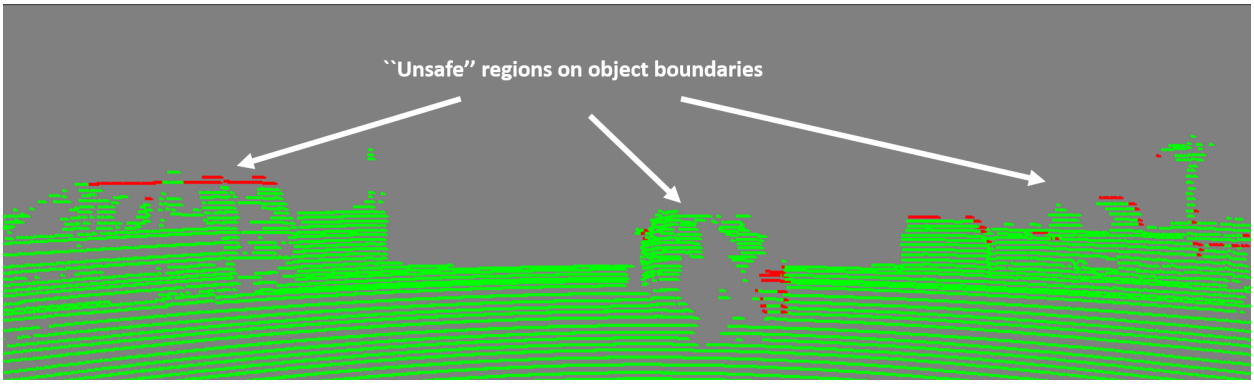
Moreover, for all algorithms, the safe depth percentage is significantly higher than the accuracy percentage. This is good for autonomous navigation, where safety is the first issue we should consider. At a safe margin of around 6.7% average scene depth (1.0 meters), the percentage of safe depth is more than 95%. We have also tested on larger safe margins, but the gain on neither the depth difference percentage nor the safe depth percentage is significant. Neither percentages will be 100%. We show limitations of our mapping approach in Fig. 17. There are several reasons for inducing depth errors: glass windows of cars can not be well estimated, depth estimates on dynamic objects are not consistent, occlusions and disocclusions make it hard for cameras to produce reliable depth estimates, and distant surfaces are not able to be well reconstructed due to the insufficient visual disparity. By setting s_d to 1.0, we get the safe depth image. "Safe" pixels are marked in green, while "unsafe" pixels are marked in red. "Safe" pixels cover most regions. However, there are still a few "unsafe" pixels on the boundary of objects due to the limitation on a large

voxel size. Finer voxels lead to safer reconstructed maps at the cost of heavier computation.

(a) A captured keyframe image.



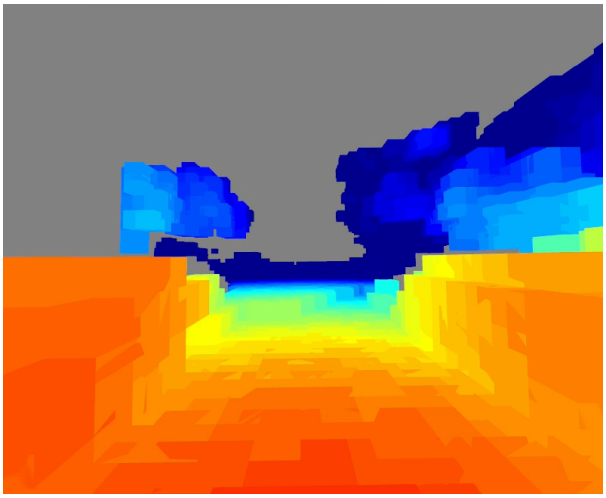
(b) The error depth image of the reconstructed keyframe view.



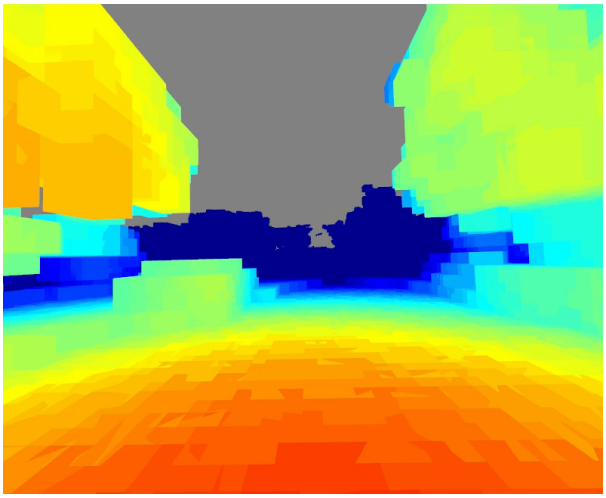
(c) The safe depth image of the reconstructed keyframe view.

Figure 17: The limitation of our reconstructed 3D maps projected on keyframe views. (a) A captured keyframe image. (b) Reconstructed depth errors shown in color. We see large depth errors on glass, moving objects, and occlusion/dis-occlusion regions. (c) The safe depth image obtained by setting s_d to 1.0. "Safe" pixels are marked in green, while "unsafe" pixels are marked in red. While "safe" pixels cover most regions, we notice a few "unsafe" pixels on the boundary of objects.

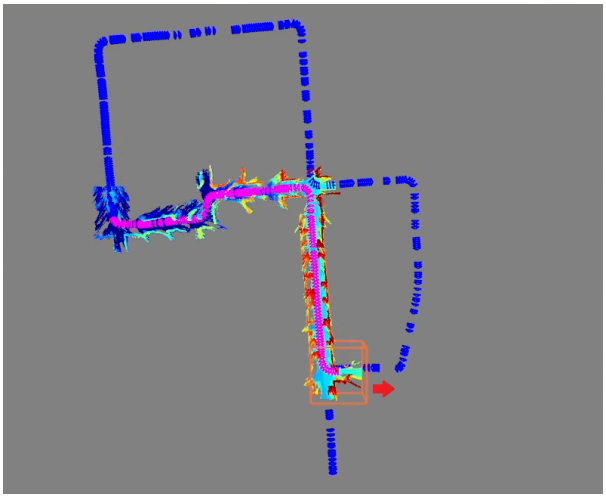
(a) A captured image to show the perceived environment.



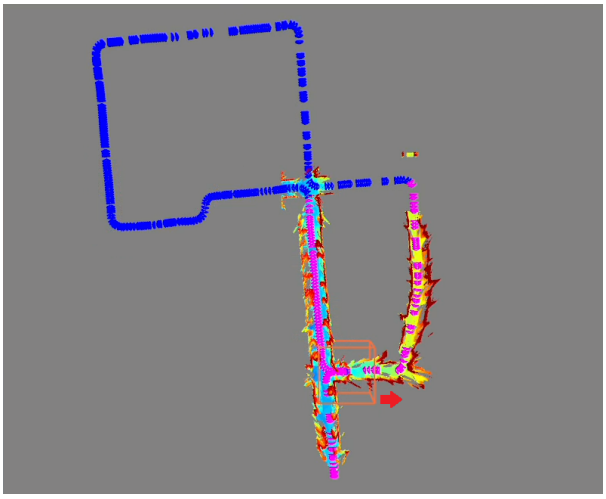
(b) The reconstructed local dense map before loops.



(c) The reconstructed local dense map after loops



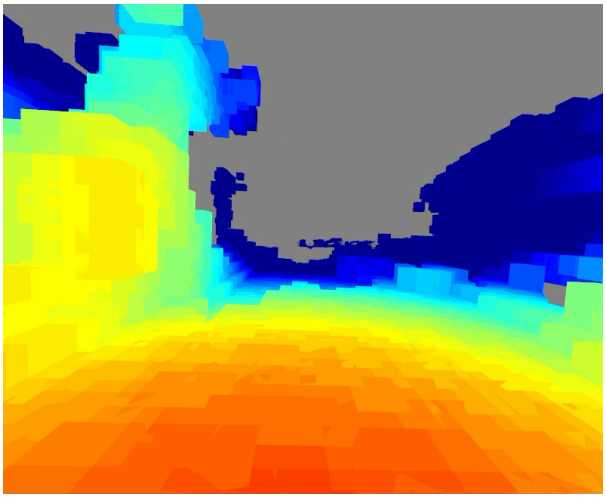
(d) The top-down view of the reconstructed local dense map before loops.



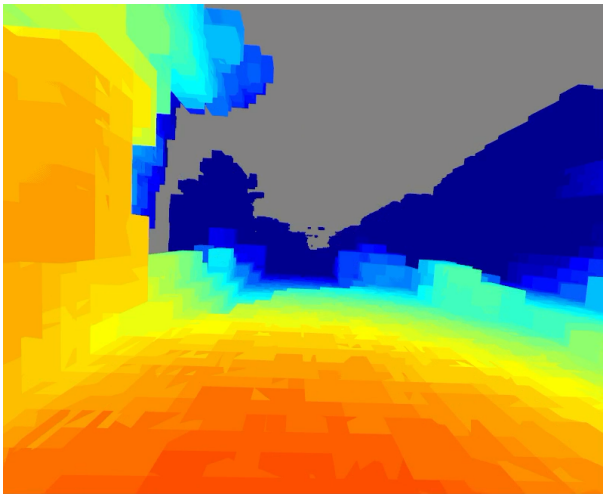
(e) The top-down view of the reconstructed local dense map after loops.

Figure 18: Qualitative mapping results on the testing sequence KITTI00. (a) a captured image to show the perceived environment. (b, c) First person views of the reconstructed local dense maps before and after loop closures. The color varies with respect to the object distance. (d, e) Corresponding top-down views of the reconstructed local dense maps. The color varies with respect to the height to show the environment vertical structure. The orange box is the instant working volume. Driving directions of the cars are shown by red arrows. As loops are closed in an independent low-priority thread, there is a time offset between (b) and (c) / (d) and (e). Our system produces temporally and spatially corrected local maps for motion planning in a bounded time.

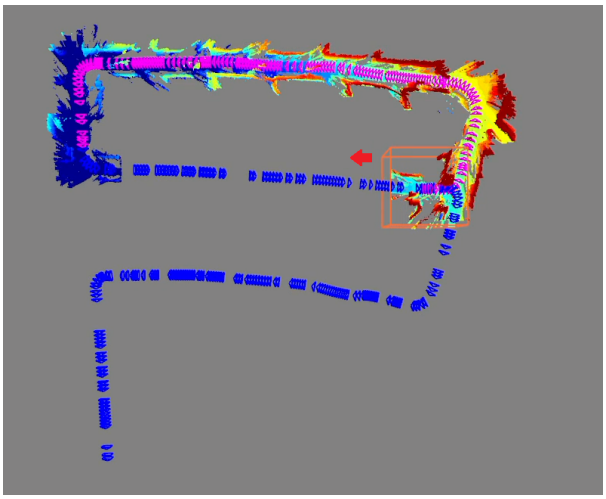
(a) A captured image to show the perceived environment.



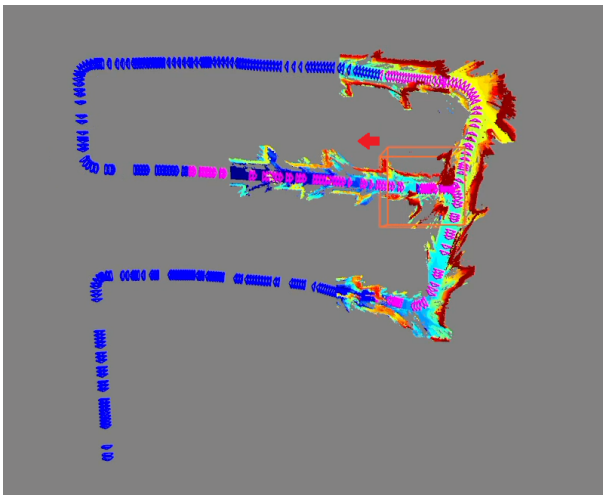
(b) The reconstructed local dense map before loops.



(c) The reconstructed local dense map after loops



(d) The top-down view of the reconstructed local dense map before loops.



(e) The top-down view of the reconstructed local dense map after loops.

Figure 19: Qualitative mapping results on the testing sequence KITTI05. (a) a captured image to show the perceived environment. (b, c) First person views of the reconstructed local dense maps before and after loop closures. The color varies with respect to the object distance. (d, e) Corresponding top-down views of the reconstructed local dense maps. The color varies with respect to the height to show the environment vertical structure. The orange box is the instant working volume. Driving directions of the car are shown by red arrows. As loops are closed in an independent low-priority thread, there is a time offset between (b) and (c) / (d) and (e). Our system produces temporally and spatially corrected local maps for motion planning in a bounded time.

5.1.3 Qualitative Analysis of Online Mapping Performance

We present the qualitative mapping performance in this subsection. Results obtained in testing sequences of KITTIT00 and KITTI05 are shown in Fig. 18 and Fig. 19, respectively. For both figures, sub-figure (a) is a captured image to show the perceived environment. The first person view of the reconstructed local dense maps before and after loop closures are shown in sub-figures (b) and (c). Corresponding top-down views of the reconstructed local dense maps are shown in sub-figures (d) and (e). Local keyframes (Sect. 3.3) are shown in purple, while the remaining keyframes under the memory management (Sect. 4.1) are shown in blue. The instant working window is shown in orange (Sect. 3.2.1). Our system produces dense local maps in a bounded time. The reconstructed maps are not only temporally correlated but also spatially correlated. They represent the maximum information around the instant keyframe pose regarding what a real-time system is able to process with the real-time processing requirement.

5.1.4 Online Performance

Table 3: The statistics of the time spent by each thread on the KITTI00 sequence.

Thread	**Mean (ms)**	**Std (ms)**	**Max (ms)**
Feature Tracking	16.26	6.74	27.07
Complexity-Bounded Optimization	60.94	16.20	92.49
Loop Detection	0.59	0.64	1.89
Map Integration	59.36	11.64	91.58
Dense Map Reconstruction	30.59	9.74	51.23

Table 4: The statistics of the time spent by the sliding window bundle adjustment and the complexity-bounded optimization thread at different parameter settings.

N_w	M_w	**Mean Sliding Window BA Time (ms)**	**Mean Complexity-Bounded Opt. Thread Time (ms)**
12	200	31.42	42.98
12	400	46.27	58.93
16	200	34.11	47.12
16	400	47.57	60.94
20	200	36.98	50.68
20	400	49.67	64.22

Table 5: The computation time of the depth map computation at different resolutions in the KITTI00 sequence. Both CPU and GPU implementations of SGM are tested. Corresponding delays from receiving images to local dense map generations are also summarized.

Resolution	**CPU / Average Delay / Max Delay (ms)**	**GPU / Average Delay / Max Delay (ms)**
1241×376	119.97 / 258.76 / 355.87	28.36 / 167.74 / 264.26
620×183	27.31 / 166.1 / 263.21	8.75 / 147.54 / 244.27
310×91	5.72 / 144.51 / 241.62	3.15 / 141.94 / 239.05

Since real-time processing requirement is the basis of robotics applications, we report the statistics of times spent by different threads on the KITTI00 sequence. Table 3 shows the results. Our system is able to process the whole sequence at 10 Hz (where it is recorded). The average delay from receiving images to reconstructing local dense maps

is 167.74 ms, while the max delay is 314.26 ms. It consists of the processing time of the feature tracking thread, the complexity-bounded optimization thread, the map integration thread, and the local map reconstruction thread. We also test the influence of two parameters, the length of the sliding window N_w and the maximum number of features M_w, on the computation time of the sliding window bundle adjustment. Statistics are shown in Table. 4. We see that, by limiting the length of the sliding window and the maximum number of features, we are able to bound the sliding window bundle adjustment computation time.

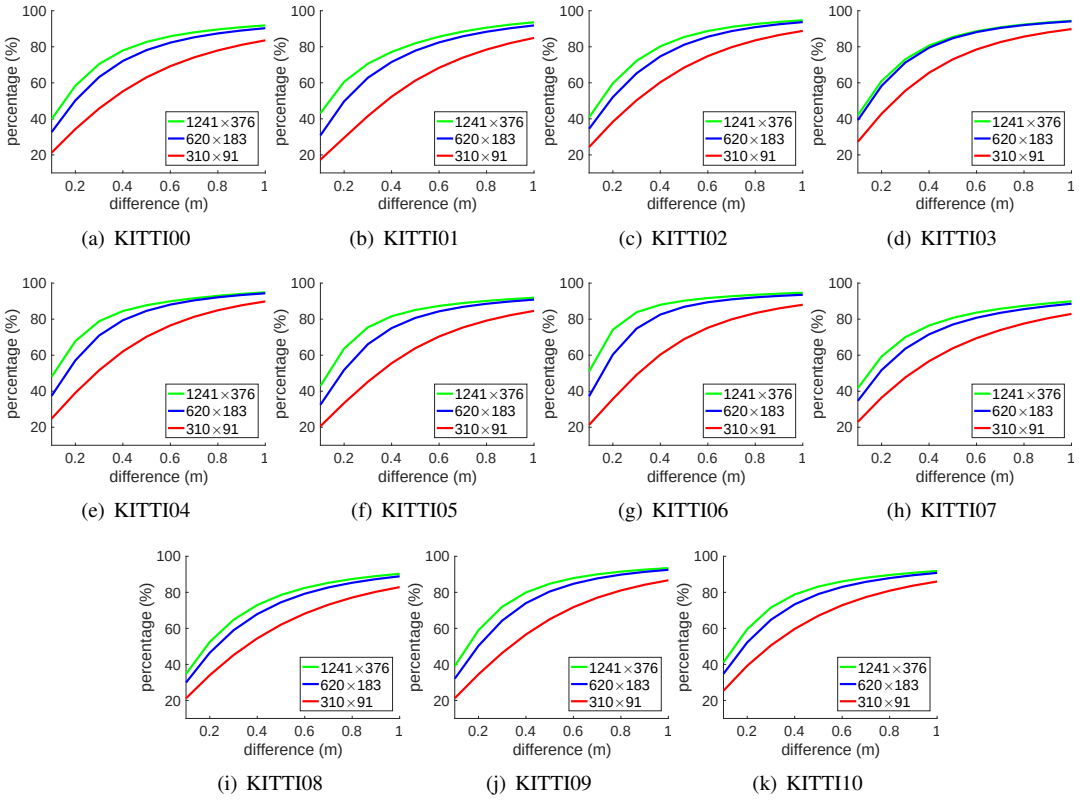
Our system runs with CPUs and GPUs by default. We have done additional tests on our system performance without GPUs, and note that only the depth map computation part (Sect. 3.3) of our system requires GPUs. We replace the GPU-based implementation with the CPU-based implementation using libraries from OpenCV. Since the computation time varies with respect to the image resolution, we test different image resolutions by the downsampling. Table 5 shows how different image resolutions affect the total dense map reconstruction delays, while corresponding mapping performances are shown in Figs. 20, 21, 22.

Downsampling the image resolution leads to a significant reduction of computation time at the cost of mapping accuracy loss. However, the loss of mapping accuracy is not notable at certain threshold ranges ($t_d > 1.0$, $s_d > 1.0$). With CPU-only platforms, downsampling the image resolution to 620×183 is recommended for a balance between the mapping performance and the map generation delay.

5.2 NewCollege Dataset

5.2.1 SLAM Pipeline Performance

The sequence of the NewCollege dataset is recorded by a robot traversing the NewCollege campus and outdoor environments. The total traveled distance is 2.2 km. It lasts for about 44 minutes, with stereo images captured at 20 fps and at a 512 × 382 image resolution. It contains several loops and fast rotations that make the sequence challenging. We compare our proposed system with ORB-SLAM2 (Mur-Artal and Tardós, 2017). For a fair comparison, we do not compare our system with the earlier ORB-SLAM (Mur-Artal et al., 2015) as it is designed for a monocular camera and the metric scale is unobservable. We set the maximum number of features detected in each frame in ORB-SLAM2 to be 800. For other parameters, we use the default values provided by the author's implementation. We empirically find that if the maximum number of features detected in each frame is below 800, ORB-SLAM2 will fail to track early in the testing sequence. In addition, ORB-SLAM2 crashes at about 24 minutes because the main memory is has become full. We thus compare the estimated trajectories in the first 24 minutes. Fig. 23 shows the results. The trajectories estimated by our system and ORB-SLAM2 are similar (Fig. 23(a)). The number of keyframes and features as time



(a) KITTI00 (b) KITTI01 (c) KITTI02 (d) KITTI03

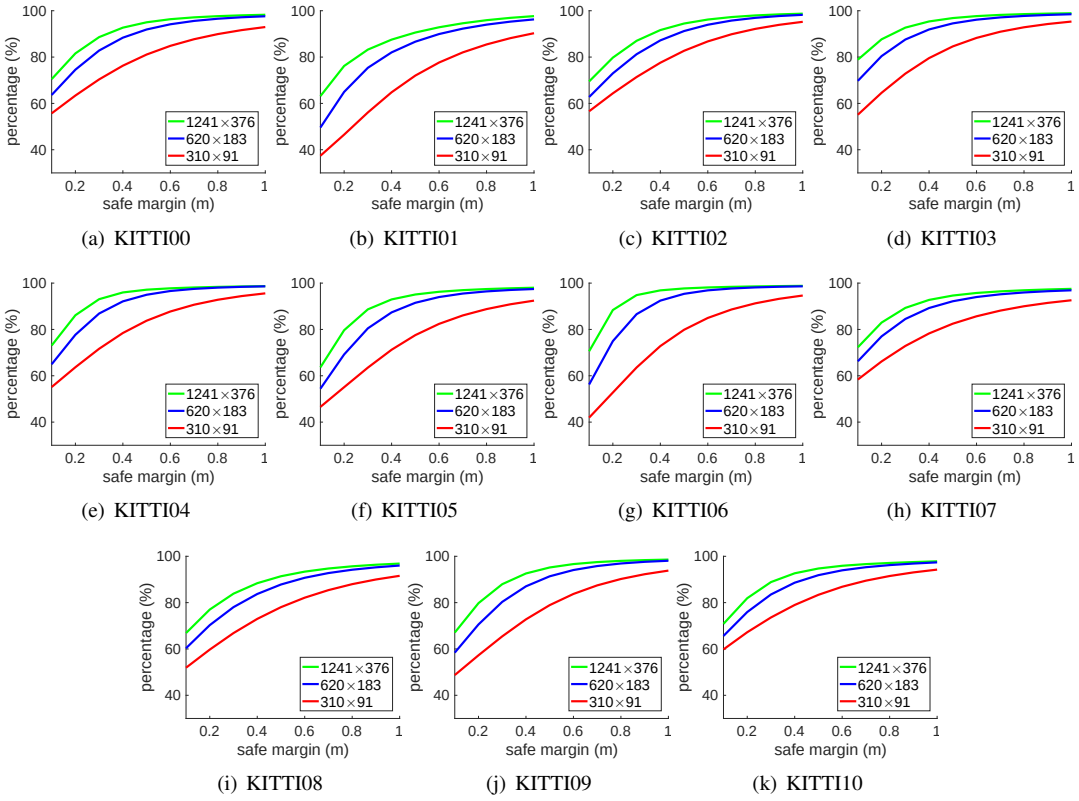
(e) KITTI04 (f) KITTI05 (g) KITTI06 (h) KITTI07

(i) KITTI08 (j) KITTI09 (k) KITTI10

Figure 20: The depth difference percentage at different image resolutions on the KITTI dataset. We enumerate the depth difference from 0.67% (0.1 meters) to 6.7% (1.0 meters) of the average scene depth, and calculate the percentage of depth differences within a given difference threshold. The horizontal axis is the difference threshold, while the vertical axis is the corresponding percentage. Results at different image resolutions are shown.

passes is shown in Fig. 23(b) and Fig. 23(c), respectively. Our system consists of fewer keyframes and features than ORB-SLAM2 while achieving a similar localization accuracy. The reason is that our system only contains sparse image corner features that are "informative" for localization, while ORB-SLAM2 processes sparse image corner features that are potentially redundant. Examples of detected image corner features in our system and ORB-SLAM2 are shown in Fig. 23(d) and Fig. 23(e), respectively. Image (d) is normalized by the contrast limited adaptive histogram equalization (Zuiderveld, 1994). Our detected image corner features are evenly distributed around the image as we set the minimum distance between the detected image corner features to be beyond a predefined threshold (Sect. 3.1.1). We also use colors to indicate the tracked length of detected image corner features. Corners are redder if they are tracked for a longer time, while image corner features are bluer if they are tracked for a shorter time. Corners in ORB-SLAM2 may overlap, leading to a redundant representation. Therefore, our system is more memory-efficient than ORB-SLAM2.

For the final trajectory comparison, we extract the result presented in ORB-SLAM (Mur-Artal et al., 2015) since ORB-SLAM2 crashes at 24 minutes. Fig. 24 shows the comparison. As highlighted in the red boxes, the robot travels the



(a) KITTI00 (b) KITTI01 (c) KITTI02 (d) KITTI03

(e) KITTI04 (f) KITTI05 (g) KITTI06 (h) KITTI07

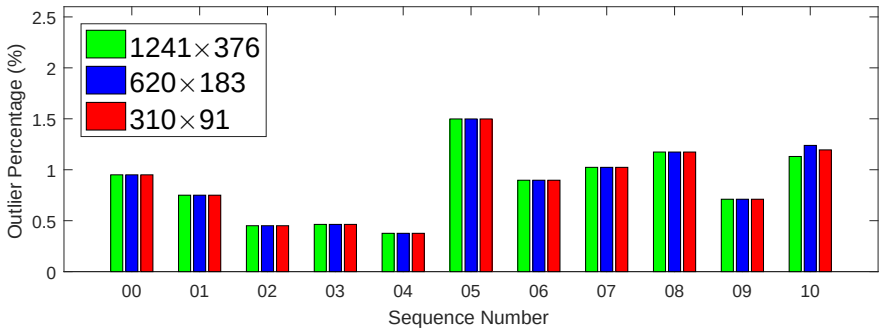
(i) KITTI08 (j) KITTI09 (k) KITTI10

Figure 21: The safe margin percentage at different image resolutions on the KITTI dataset. A depth estimate, which is less than the ground truth depth value plus the safe margin, is supposed to be a safe depth estimate. The horizontal axis is the safe margin, while the vertical axis is the corresponding percentage. Results at different image resolutions are shown.

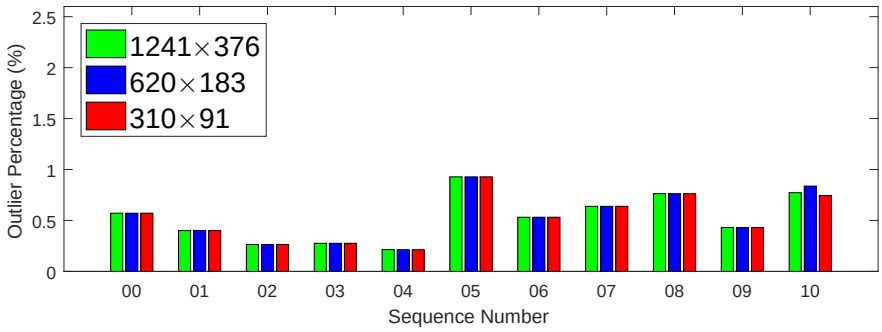
same route twice, and the estimated paths should roughly overlap. It can be seen that our estimated trajectory has less position drift compared to ORB-SLAM.

5.2.2 Effect of Histogram Equalization

We show the effect of the contrast limited adaptive histogram equalization (Zuiderveld, 1994) on the tracking performance (Fig. 25). For a fair comparison, we fix all parameters, except for the open/close of the histogram equalization. Loop closures are disabled for the illustration of the difference on tracked trajectories. We play the video sequence of the first round walk of the NewCollege dataset and run the SLAM pipeline with or without the histogram equalization. Images with tracked information are shown Fig. 25 (a) and (b). Tracked image corner features are shown in colors to indicate the tracked length of the detected image corner features. Corners are redder if they are tracked for a longer time, while image corner features are bluer if they are tracked for a shorter time. We see that, tracked image corner features on images with histogram equalization are denser and more evenly distributed among images. The reason is that the local gradients are stronger with the histogram equalization than the ones without histogram equalization.



(a) $o_d = 3$.



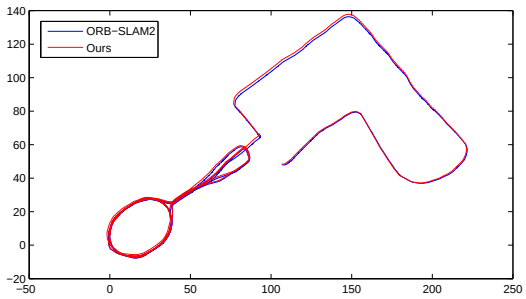
(b) $o_d = 4$.

Figure 22: The outlier percentage for different image resolutions on the KITTI dataset. The horizontal axis is the sequence number of the KITTI dataset (from 00 to 10), while the vertical axis is the outlier percentage given an outlier threshold (o_d). The outlier percentage is the percentage of depth estimates that are less than the ground truth depth values given a certain outlier threshold ($o_d = 3$ or 4). We compare the results at different image resolutions. The outlier percentages are similar at different image resolutions. Similar results are observed for larger outlier thresholds.

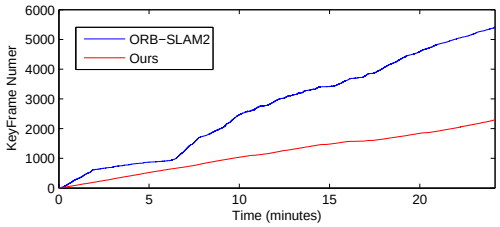
The histogram equalization is beneficial not only for image corner feature detection but also for image corner feature tracking using Lucas-Kanade sparse optical flow that is based on local gradients (Baker and Matthews, 2004). Top-down views of trajectories with and without the histogram equalization are plotted in Fig. 25 (c) and (d). Trajectories are shown by sets of estimated camera poses (in purple). We notice a large displacement on the estimated trajectory on almost identical paths if the histogram equalization is disabled. While this displacement is not significant on the estimated trajectory if the histogram equalization is enabled.

5.2.3 Memory Management

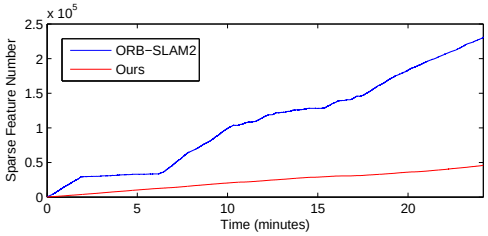
The system memory will be filled up if it runs for a long time. We simulate this situation by limiting the number of maximum keyframes m_f used in our system (Sect. 4.1). Estimated trajectories are shown in Fig. 27. Results with an unlimited number of keyframes, $m_f = +\infty$, are shown in Fig. 24 (b). Our system is able to save as much useful information as possible subject to the limited memory on a real-time system. We plot the number of erased keyframes, the number of remaining keyframes, the number of erased features, and the number of remaining features with respect



(a) The top-down view of the estimated trajectory.

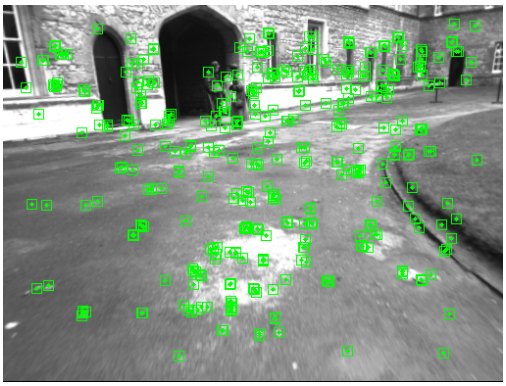


(b) The number of keyframes with respect to the time index.



(c) The number of sparse features with respect to the time index.

(d) An example of detected image corner features in our system.



(e) An example of detected image corner features in ORB-SLAM2.

Figure 23: The comparison between our system and ORB-SLAM2 (Mur-Artal and Tardós, 2017) on the first 24 minutes of NewCollege sequences. (a) Trajectories estimated by our system and ORB-SLAM2 are similar. (b) Our keyframe number is much smaller than that of ORB-SLAM2. (c) Our feature number is much smaller than that of ORB-SLAM2. (d) An example of detected image corner features in our system. Since the ground plane is textureless, we use the histogram equalization in Sect. 3.1.1 to normalize the captured images. Our detected image corner features are shown in colors (redder if they are tracked for a longer time). (e) An example of detected image corner features in ORB-SLAM2. Our system is more memory-efficient than ORB-SLAM2, while for the localization performances both approaches are similar.

to the time index in Fig. 26. We observe that a large number of features are removed while a small number of sparse features are erased. Since the robot travels forward without stops, not many redundant keyframes can be removed. The textureless ground plane leads to a large number of unstable features, which are tracked in a short period of time. They are not "informative" and are erased to save memory.

(a) The final trajectory reported in ORB-SLAM (green). (b) The final trajectory estimated by our system (blue).

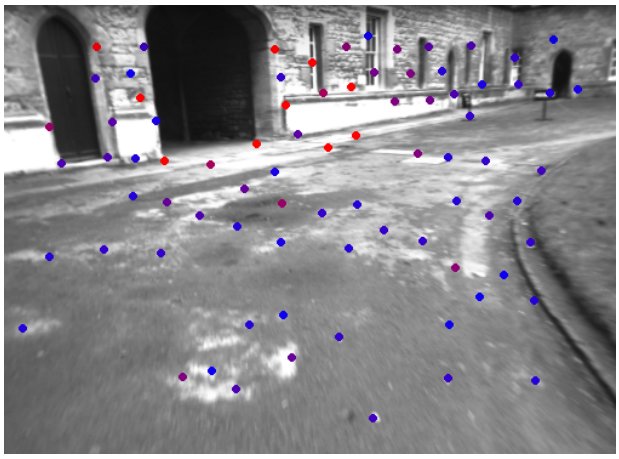
Figure 24: The top-down view of the final trajectory. We qualitatively compare our final estimated trajectory with the result reported in ORB-SLAM (Mur-Artal et al., 2015). (a) The final trajectory from ORB-SLAM. (b) The final trajectory estimated by our system. From the right-hand side of both paths (highlighted in the red boxes), we see that the position drift of our system is less than that of ORB-SLAM because the robot travels the same route twice, and the estimated paths should roughly overlap.

5.2.4 Dense Mapping Results

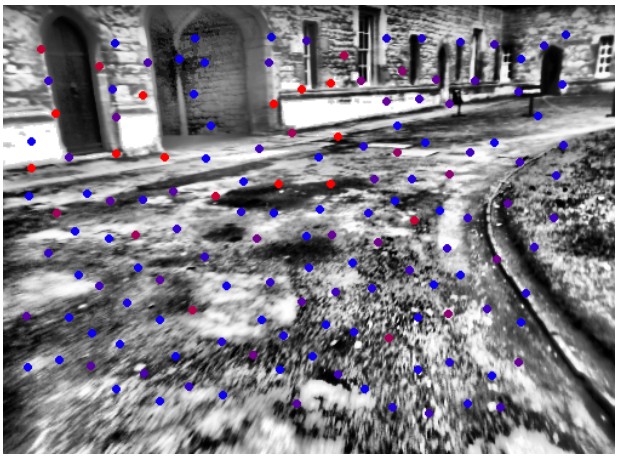
We show the qualitative mapping results on the NewCollege dataset in Fig. 28. The depth map integration distance (Sect. 3.2.3) ranges from 0.1 meters to 8.0 meters. The sliding window size (N_w) of our system is 16. The window length of the instant working volume (l_c) is 20, while the number of neighborhood keyframes (n_l) is 150. The voxel size is 0.2 meters. Captured images during the sequence, after closing loops, and at the end of the dataset are shown in sub-figures (a), (d), and (g), respectively. Corresponding reconstructed local dense maps are shown in sub-figures (b), (e), and (h). Local keyframes (Sect. 3.3) are shown in purple, while the remaining keyframes under the memory management (Sect. 4.1) are shown in blue. The latest robot pose is marked in yellow. We also plot the top-down views in sub-figures (c), (f), and (i). These local maps reconstructed on the fly represent the maximum information around the instant keyframe pose regarding what a real-time system is able to process with the real-time processing requirement.

5.3 Online Campus Reconstruction

In this subsection, we present real-time dense reconstruction results of the traveled environment during a large-scale long walk around the HKUST university campus. The walking distance is about 5.8 km, while the traveling time is about 1 hour 20 minutes. The route covers several loops for loop closures. Synchronized stereo images are captured with a handheld experimental sensor suite VI-sensor (Nikolic et al., 2014). Captured stereo images are at 25 Hz. The image resolution is 752 $\times$ 480. Since the output of the VI-sensor contains IMU measurements (at 500 Hz), we fuse the inertial data with visual measurements. We plot the final estimated trajectory on top of the Google map (Fig. 29). It can be seen that the estimated path is almost aligned with the Google map. The number of erased keyframes, remaining



(a) A tracked images without the histogram equalization.



(b) A tracked images with the histogram equalization.

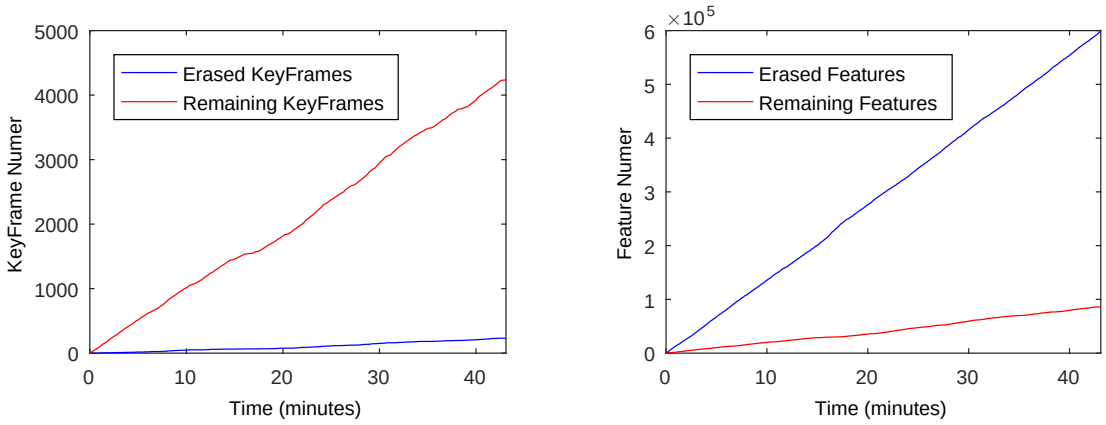
(c) The top-down view of the estimated trajectory of the first round walk without the histogram equalization.

(d) The top-down view of the estimated trajectory of the first round walk with the histogram equalization.

Figure 25: The comparison of the tracking performance (a),(c) without or (b),(d) with the histogram equalization on the first round walk of the NewCollege sequence.

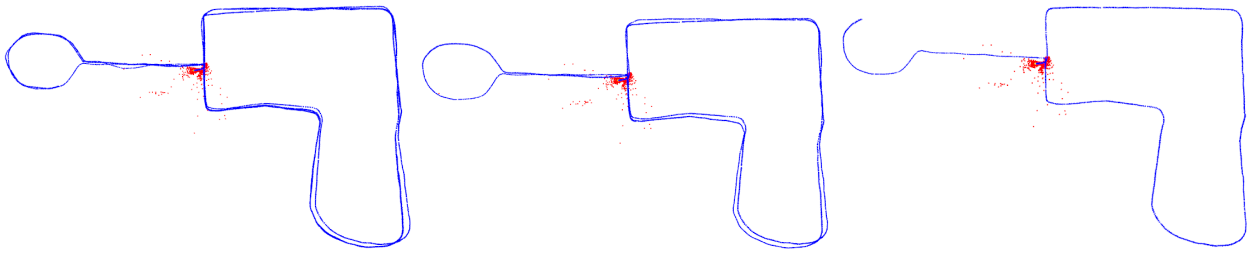
keyframes, erased features, and remaining features with respect to the time index are plotted in Fig. 30. Our system actively removes redundant keyframes and sparse features for memory efficiency.

Qualitative mapping results of the real-time campus reconstruction are shown in Fig. 31. The depth map integration distance (Sect. 3.2.3) ranges from 0.1 meters to 10.0 meters. The sliding window size N_w of our system is 16. The window length of the instant working volume (l_c) is 20, while the number of neighborhood keyframes n_l is 150. The voxel size is 0.2 meters. Snapshots of the captured sequence are shown in sub-figures (a), (d), and (g). Third person views of reconstructed local dense maps are shown in sub-figures (b), (e), and (h). We also plot the third-person views in sub-figures (c), (f), and (i). Local keyframes (Sect. 3.3) are shown in purple, while remaining keyframes under the memory management (Sect. 4.1) are shown in blue. The instant working window is shown in orange (Sect. 3.2.1).



(a) (b)

Figure 26: The number of (a) erased/remaining keyframes and (b) erased/remaining features with respect to the time index of the NewCollege dataset.



(a) $m_f = 3000$. (b) $m_f = 2000$. (c) $m_f = 1000$.

Figure 27: Estimated trajectories with the memory management on the NewCollege dataset. We set the maximum keyframe number m_f to be 3000, 2000, and 1000 (Sect. 4.1), and the estimated trajectories are shown in (a), (b), and (c), respectively. Results with an unlimited number of keyframes, i.e. $m_f = +\infty$, are shown in Fig. 24 (b).

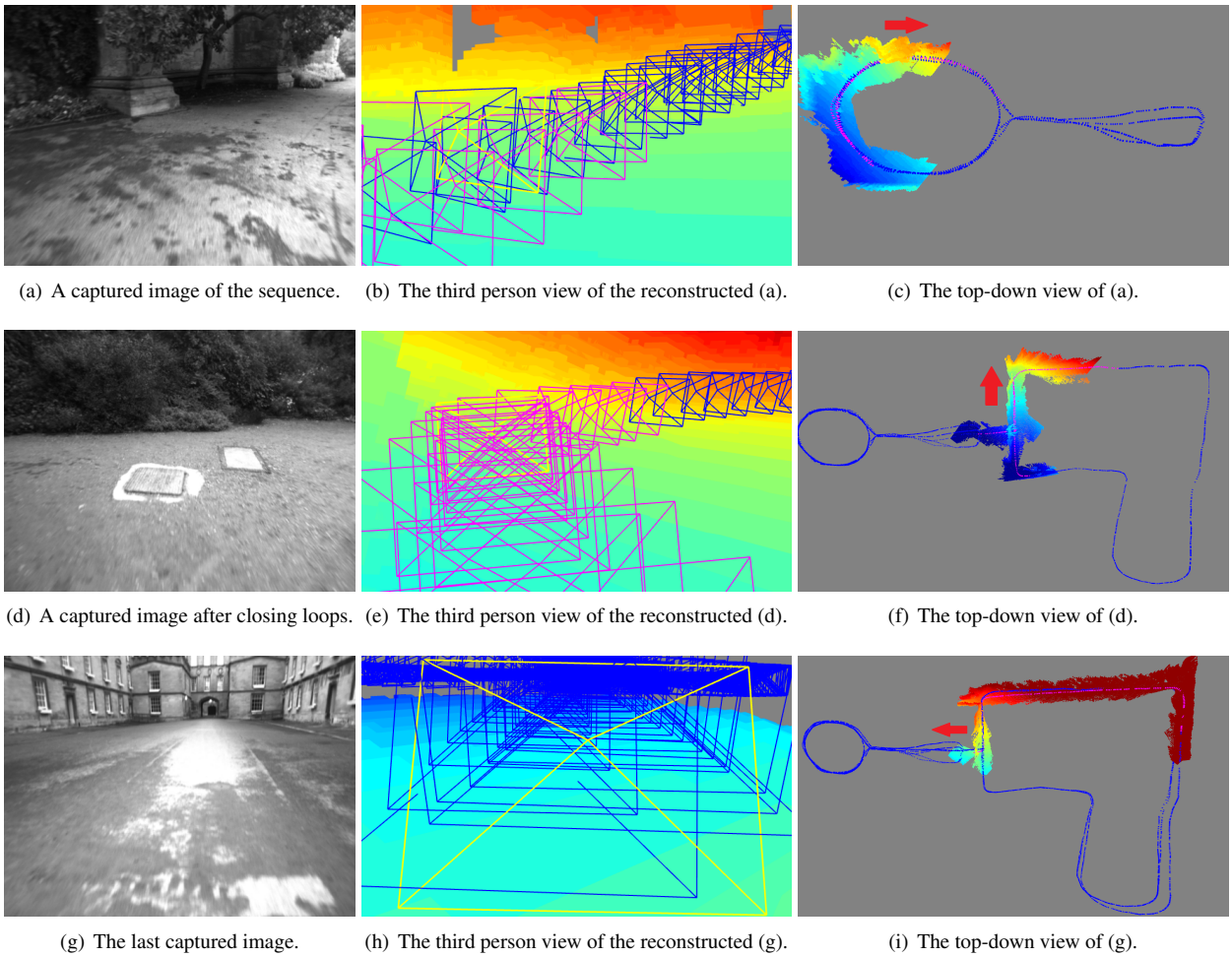
These local maps reconstructed in real time represent the maximum information around the instant keyframe pose regarding what a real-time system is able to process with the real-time processing requirement. Another mapping result is shown in the beginning of this paper (Fig. 1). More dense mapping results during the traveling are presented in the accompanying video available at

`https://1drv.ms/v/s!ApzRxvwAxXqQmk-uQ3JGZSYcnEdn`.

6 Discussions & Future Work

6.1 Sparse Feature Based SLAM Pipeline

From Table 1, we see that the localization accuracy of our approach is not as good as that of ORB-SLAM2 (Mur-Artal and Tardós, 2017). There are two main reasons why we have not adopted ORB-SLAM2 as our SLAM pipeline.



(a) A captured image of the sequence. (b) The third person view of the reconstructed (a). (c) The top-down view of (a).

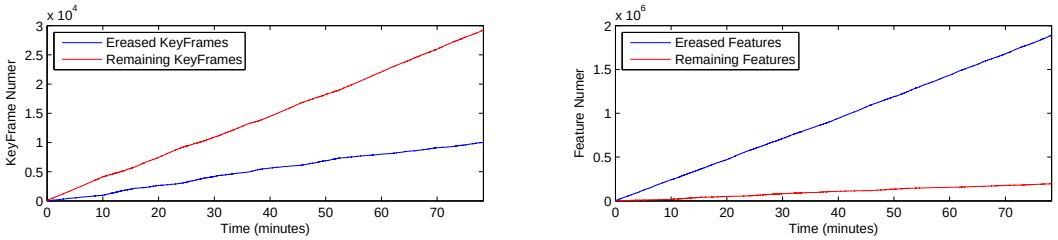
(d) A captured image after closing loops. (e) The third person view of the reconstructed (d). (f) The top-down view of (d).

(g) The last captured image. (h) The third person view of the reconstructed (g). (i) The top-down view of (g).

Figure 28: Qualitative mapping results on the NewCollege dataset. (a,) (d) and (g) The captured image of the sequence, after closing a loop, and at the end of the NewCollege dataset, respectively. (b), (e) and (h) The third person view of the reconstructed local dense maps. (c), (f) and (i) The corresponding top-down views of the reconstructed local dense maps. The color varies with respect to the height to show the environment structure.

First, the bundle adjustment window size of ORB-SLAM2 is varied for each optimization, which leads to a varying computation time of the complexity-bounded optimization thread and an overall system delay. The bundle adjustment window size of the presented SLAM pipeline in this work, however, is fixed for each optimization. The computation time is thus relatively fixed. We prefer a relatively fixed computation time for real-time robotics applications as the overall time delay is relatively fixed. On the other hand, the integration of IMU measurements on ORB-SLAM2 is non-trival. ORB-SLAM2 selects keyframes spatially, while the integration of IMU measurements requires frames connecting sequentially in time. In the last experiment (Sect. 5.2.4), we tested our algorithm with and without inertial measurements. We found that incooperating inertial measurements leads to a significant gain in tracking accuracy because the roll and pitch angles are observable. Inertial measurements can be easily integrated in the presented SLAM pipeline in this work. The main focus of this work is not on the SLAM pipeline alone, but a joint consideration

Figure 29: The estimated trajectory (shown in red) on a long walk around the HKUST university campus. We plot it on the top of the Google map of the campus. The path is almost aligned with the map.



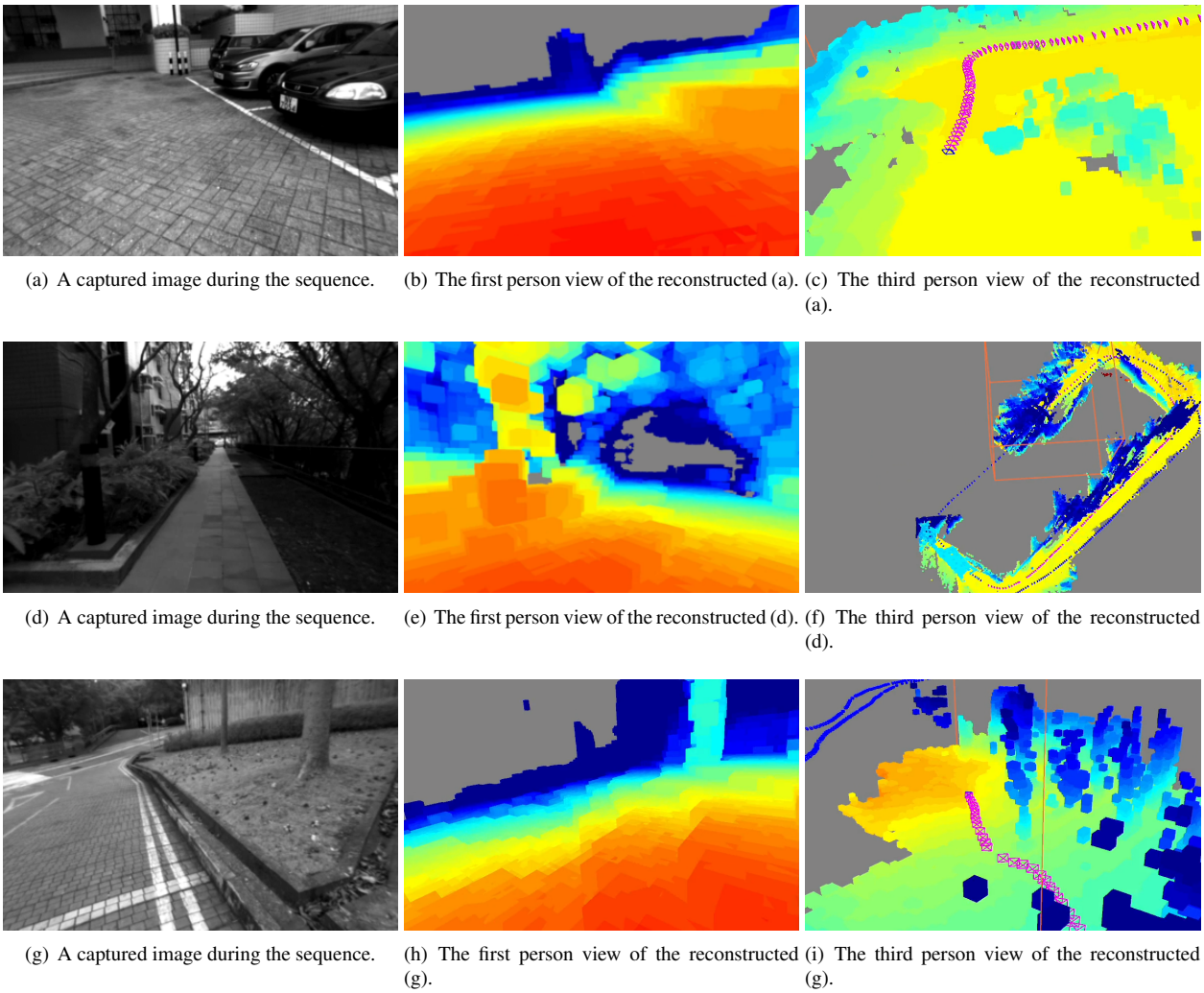
(a) (b)

Figure 30: The number of erased/remaining keyframes (a) and erased/remaining features (b) with respect to the time index of the real-time campus reconstruction experiment.

of the SLAM pipeline and the real-time dense mapping for motion planning. Advantages of ORB-SLAM2 or other state-of-art methods will be integrated into the presented SLAM pipeline in future work.

6.2 Subvolume Blending

For the subvolume blending (Sect. 3.2.4), subvolumes may be blended wrongly not due to no loop closures but due to the low accuracy of the loop transformation. On the one hand, subvolumes are extracted if they are out of the instant working volume. They are attached to nearest keyframes and stored in the memory. In other words, they are 'disappeared' in the global metric space. They are 'in the graph space' and 'recalled' if loops are closed. If the robot goes back to an already mapped area, subvolumes will be blended with nothing if no loops are closed. Once loops are detected and closed, subvolumes will be blended after the local pose graph optimization that incooperates the loop information and the local consistency. On the other hand, if the accuracy of the computed loop transformation is not high, objects between the first and second traversals will be superposed. Or if objects between the first and second



(a) A captured image during the sequence. (b) The first person view of the reconstructed (a). (c) The third person view of the reconstructed (a).

(d) A captured image during the sequence. (e) The first person view of the reconstructed (d). (f) The third person view of the reconstructed (d).

(g) A captured image during the sequence. (h) The first person view of the reconstructed (g). (i) The third person view of the reconstructed (g).

Figure 31: Qualitative mapping results of the real-time campus reconstruction. (a), (d) and (g) Snapshots of the captured sequence. (b) (e) and (h) Corresponding first person views of the reconstructed local dense maps. The color varies with respect to the object distance. (c), (f) and (i) Corresponding third person views of the local dense maps. The color varies with respect to the height to show the environment structure.

traversals are not consistent (such as moving objects), subvolumes blending may not work appropriately. In this case, the order of the depth map integration step and the subvolume wrapping and blending step is important. Referring to Fig. 8 where the dense map integration step is conducted after the subvolume blending step, we prefer the latest depth map information to the past information (obtained by subvolume blending). The spacing carving operation in the dense map integration step will chisel away part of the inconsistent blended subvolumes.

6.3 Mapping Performance & Failure Cases

The goal of this work is not in achieving better mapping accuracy than the other state-of-the-art methods, such as SGM (Hernandez-Juarez et al., 2016) and ELAS (Geiger et al., 2010), which have been mentioned in this work, but on the tradeoff between the mapping accuracy, the mapping density, and the outlier rate. We adopt a conservative strategy, where the priority is as follows: mapping density $>$ outlier rate $>$ mapping accuracy. A denser map is beneficial for planning safety, while the outlier rate and the mapping accuracy are beneficial for planning efficiency. The mapping density is usually in conflict with the mapping accuracy as well as the outlier rate. Depths in regions without distinguished textures are unlikely to be estimated based on the local appearance. They are 'guessed' via utilizing the spatial correlation between neighboring depth estimates. Potential reductions on the mapping accuracy or a growth of outlier rate is expected if the mapping density is increased. The key idea of this work is to make full use of the temporal correlation (Sect. 3.2.3) and the spatial correlation (Sect. 3.2.2 and Sect. 3.2.4) on consecutive depth estimates to increase the mapping density. We also suppress inconsistent mapping results by space carving (Sect. 3.2.3). Some mapping failures occurred during the experiments due to the limitation of visual measurements (such as the textureless glass, occlusions/disocclusions, insufficient visual disparities, and dynamic objects shown in Fig. 17). To overcome difficulties in textureless regions and moving objects, we can utilize information provided by other sensors, such as the laser scanner. Future work will investigate how to fuse the information from different sensors to further improve the mapping performance. The multi-camera setup will also be considered for mapping on different viewing directions.

6.4 Hardware Limitations

We have to take limited computations (CPUs and GPUs) and limited memory into considerations for real-time robotics systems. This requires that the computational complexity of the present system does not scale with respect to the travel time or the travel distance. We inspect all modules on our presented system, and identity the loop detection, the global pose graph optimization, and the keyframe/feature/subvolumes storage as the three modules affected by the hardware limitation. Based on the experiments presented in this paper, we find that the biggest bottleneck is the limited memory. If the keyframe number is larger than 5000, the consumed memory space will be larger than 12 Gb, though we have an active redundancy removal module (Sect. 3.1.2). When the keyframe number is about 5000, it takes about 5 ms and 300 ms to get the loop detection and the global pose graph optimization done, respectively. The computation time spent on the loop detection and the global pose graph optimization linearly scales with respect to the number of keyframes. Future work will consider storing keyframe/feature/subvolume data onto the hard disk. The computation time of the loop detection is not significant. The global pose graph optimization is run on a low-priority thread. Its

output for the local pose graph optimization (Sect. 3.1.2) is the relative transformations between keyframes. Note that, the local pose graph optimization (Sect. 3.1.2) is for the local consistency, while the global pose graph optimization (Sect. 3.1.3) is for the global consistency. The time delay induced by the global pose graph optimization will not affect the local consistency of the local structure around the latest robot pose, which is important for the local map generation for motion planning. Future work will also consider performing the loop detection and the global pose graph optimization on cloud computing.

7 Conclusions

We presented a dense mapping system for autonomous navigation. The proposed system has multi-threads run asynchronously. This system was built on a presented sparse feature based SLAM pipeline, which provides the latest poses for dense mapping. The key of our dense mapping approach is the presented map representation that combines the advantages of both metric maps and topological maps. By using this map representation, depth estimates at different locations are easily integrated and dense maps are flexibly deformed if loops are closed. For the benefit of handling dynamic objects, we introduced a subvolume wrapping and blending scheme that operated on truncated signed distance functions. Moreover, we have shown how to reconstruct local maps around instant poses in real time based on the presented map representation. The complexity of our method does not scale with respect to the travel time or the travel distance. In the end, we verified that our approach obtains a good tradeoff between the mapping accuracy, the mapping density and the outlier rate on the KITTI and NewCollege datasets, and a large-scale outdoor experiment. The built maps are suitable for real-time planning where safety is the first concern.

References

Amazon Robotics LLC (2015). *AmazonRobotics*. Amazon Robotics LLC.

Angeli, A., Doncieux, S., Meyer, J. A., and Filliat, D. (2009). Visual topological SLAM and global localization. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom.*

Baker, S. and Matthews, I. (2004). Lucas-Kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 56(3):221–255.

Bellavia, F., Fanfani, M., and Colombo, C. (2015). Selective visual odometry for accurate AUV localization. *Autonomous Robots*, pages 1–11.

Boissonnat, J. D., Faugeras, O. D., and Bras-Mehlman, E. L. (1988). Representing stereo data with the delaunay triangulation. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom.*

Bry, A. and Roy, N. (2011). Rapidly-exploring random belief trees for motion planning under uncertainty. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom.*

Chen, J., Liu, T., and Shen, S. (2016). Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom.*

Choset, H. and Nagatani, K. (2001). Topological simultaneous localization and mapping (slam): Toward exact localization without explicit localization. *IEEE Transactions on Robotics and Automation*, 17:125–137.

Cole, D. M. and Newman, P. M. (2006). Using laser range data for 3D SLAM in outdoor environments. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom.*

Curless, B. and Levoy, M. (1996). A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*.

Davison, A., Reid, I., Molton, N. D., and Stasse, O. (2007). MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067.

Duckett, T., Marsland, S., and Shapiro, J. (2000). Learning globally consistent maps by relaxation. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom.*

Dudek, G., Jenkin, M., Milios, E., and Wilkes, D. (1991). Robotic exploration as graph construction. *IEEE Transactions on Robotics and Automation*, 7(6):859–865.

Elfes, A. (1989). Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46 – 57.

Engel, J., Schöps, T., and Cremers, D. (2014). LSD-SLAM: Large-scale direct monocular SLAM. In *European Conference on Computer Vision*.

Fairfield, N., Kantor, G. A., and Wettergreen, D. (2007). Real-time SLAM with octree evidence grids for exploration in underwater tunnels. *Journal of Field Robotics*, 24(1-2).

Filliat, D. and Meyer, J. (2002). Global localization and topological map-learning for robot navigation. In *Seventh International Conference on Simulation of Adaptive Behavior*.

Fioraio, N., Taylor, J., Fitzgibbon, A., Stefano, L. D., and Izadi, S. (2015). Large-scale and drift-free surface reconstruction using online subvolume registration. In *Proc. of the IEEE Intl. Conf. on Comput. Vis. and Pattern Recognition*.

Fischler, M. A. and Bolles, R. C. (1981). Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Comm. of the ACM*, pages 726–740.

Forster, C., Carlone, L., Dellaert, F., and Scaramuzza, D. (2015). IMU preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. In *Proc. of Robot.: Sci. and Syst.*

Fraundorfer, F., Engels, C., and Nister, D. (2007). Topological mapping, localization and navigation using image collections. In *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.*, pages 3872–3877.

Gabe, S., Christopher, M., Ian, R., and Paul, N. (2010). Vast-scale outdoor navigation using adaptive relative bundle adjustment. *Intl. J. Robot. Research*, 29(8):958–980.

Gálvez-López, D. and Tardós, J. D. (2012). Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197.

Gao, F. and Shen, S. (2016a). Online quadrotor trajectory generation and autonomous navigation on point clouds. In *Proc. of the IEEE International Symposium on Safety, Security, and Rescue Robotics.*, pages 139–146.

Gao, F. and Shen, S. (2016b). Online quadrotor trajectory generation and autonomous navigation on point clouds. In *Proc. of the IEEE International Symposium on Safety, Security, and Rescue Robotics.*

Geiger, A., Roser, M., and Urtasun, R. (2010). Efficient large-scale stereo matching. In *Asian Conference on Computer Vision*.

Grisetti, G., Kümmerle, R., Stachniss, C., and Burgard, W. (2010). A tutorial on graph-based SLAM. *IEEE Intelligent Transportation Systems Magazine*, 2:31–43.

Hartley, R. and Zisserman, A. (2003). *Multiple View Geometry in Computer Vision (2nd)*. Cambridge University Press.

Hebert, M., Caillas, C., Krotkov, E., Kweon, I. S., and Kanade, T. (1989). Terrain mapping for a roving planetary explorer. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom.*

Hernandez-Juarez, D., Chacón, A., Espinosa, A., D.Vázquez, Moure, J., and López, A. (2016). Embedded real-time stereo estimation via semi-global matching on the GPU. In *International Conference on Computational Science*.

Hesch, J. A., Kottas, D. G., Bowman, S. L., and Roumeliotis, S. I. (2014). Consistency analysis and improvement of vision-aided inertial navigation. *IEEE Trans. Robot.*, 30(1):158–176.

Hirschmuller, H. (2008). Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2).

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

Huang, A. S., Bachrach, A., Henry, P., Krainin, M., Maturana, D., Fox, D., and Roy, N. (2011). Visual odometry and mapping for autonomous flight using an RGB-D camera. In *Proc. of the Intl. Sym. of Robot. Research*, Flagstaff, AZ.

Kaelbling, L. P. and Shatkay, H. (2002). Learning geometrically-constrained hidden Markov models for robot navigation: Bridging the topological-geometrical gap. *Journal Of Artificial Intelligence Research*, 16:167–207.

Klingensmith, M., Dryanovski, I., Srinivasa, S., and Xiao, J. (2015). CHISEL : Real time large scale 3D reconstruction onboard a mobile device using spatially-hashed signed distance fields. In *Proc. of Robot.: Sci. and Syst.*

Konolige, K. and Agrawal, M. (2008). FrameSLAM: From bundle adjustment to real-time visual mapping. *IEEE Transactions on Robotics*, 24(5):1066–1077.

Kuipers, B. and Byun, Y.-T. (1988). A robust qualitative method for spatial learning in unknown environments. In *Proceedings of the National Conference on Artificial Intelligence*.

Kutulakos, K. and Seitz, S. (1999). A theory of shape by space carving. In *Proc. of the IEEE Intl. Conf. Comput. Vis.*

Kuwata, Y., Karaman, S., Teo, J., Frazzoli, E., How, J. P., and Fiore, G. A. (2009). Real-time motion planning with applications to autonomous urban driving. *IEEE Trans. Contr. Sys. Techn.*, 17:1105–1118.

Labatut, P., Pons, J.-P., and Keriven, R. (2007). Efficient Multi-View Reconstruction of Large-Scale Scenes using Interest Points, Delaunay Triangulation and Graph Cuts. In *Proc. of the IEEE Intl. Conf. Comput. Vis.*

Li, M. and Mourikis, A. (2013). High-precision, consistent EKF-based visual-inertial odometry. *Intl. J. Robot. Research*, 32(6):690–711.

Ling, Y. and Shen, S. (2017). Building maps for autonomous navigation using sparse visual SLAM features. In *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.*

Ling, Y., Wang, K., and Shen, S. (2018). Probabilistic Dense Reconstruction from a Moving Camera. In *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.*

Litvinov, V. and Lhuillier, M. (2013). Incremental solid modeling from sparse and omnidirectional structure-from-motion data. In *British Machine Vision Conference*.

Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169.

Lovi, D., Birkbeck, N., Cobzas, D., and Jagersand, M. (2010). Incremental free-space carving for real-time 3D reconstruction. In *3D Data Processing, Visualization, and Transmission*.

Lynen, S., Sattler, T., Bosse, M., Hesch, J., Pollefeys, M., and Siegwart, R. (2015). Get out of my lab: Large-scale, real-time visual-inertial localization. In *Proc. of Robot.: Sci. and Syst.*, Rome, Italy.

Maddern, W., Milford, M., and Wyeth, G. F. (2012). CAT-SLAM : Probabilistic localisation and mapping using a continuous appearance-based trajectory. *Intl. J. Robot. Research*, 31(4):429–451.

Meagher, D. (1982). Geometric modeling using octree-encoding. *Computer Graphics and Image Processing*, 19.

Middelberg, S., Sattler, T., Untzelmann, O., and Kobbelt, L. (2014). Scalable 6-DOF localization on mobile devices. In *European Conference on Computer Vision*.

Mur-Artal, R., Montiel, J. M. M., and Tardós, J. D. (2015). ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163.

Mur-Artal, R. and Tardós, J. (2017). ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Trans. Robot.*, 33(5):1255–1262.

Nießner, M., Zollhöfer, M., Izadi, S., and Stamminger, M. (2013). Real-time 3D reconstruction at scale using voxel hashing. *ACM Trans. Graph.*, 32(6).

Nikolic, J., Rehder, J., Burri, M., Gohl, P., Leutenegger, S., Furgale, P. T., and Siegwart, R. (2014). A synchronized visual-inertial sensor system with FPGA pre-processing for accurate real-time SLAM. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom.*, pages 431–437.

Nister, D., Naroditsky, O., and Bergen, J. (2004). Visual odometry. In *Proc. of the IEEE Intl. Conf. on Comput. Vis. and Pattern Recognition*.

Oleynikova, H., Burri, M., Taylor, Z., Nieto, J., Siegwart, R., and Galceran, E. (2016). Continuous-time trajectory optimization for online UAV replanning. In *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.*

Pepperell, E., Corke, P. I., and Milford, M. J. (2014). All-environment visual place recognition with smart. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom.*, pages 1612–1618.

Pfaff, P., Triebel, R., Stachniss, C., Lamon, P., Burgard, W., and Siegwart, R. (2007). Towards mapping of cities. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom.*

Ratliff, N., Zucker, M., Bagnell, J., and Srinivasa, S. (2009). CHOMP: Gradient optimization techniques for efficient motion planning. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom.*

Richard, N., Shahram, I., Otmar, H., David, M., David, K., Andrew, D., Pushmeet, K., Jamie, S., Steve, H., and Andrew, F. (2011). Kinectfusion: Real-time dense surface mapping and tracking. In *The IEEE International Symposium on Mixed and Augmented Reality*.

Richter, C., Bry, A., and Roy, N. (2013). Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Proc. of the Intl. Sym. of Robot. Research*.

Romanoni, A. and Matteucci, M. (2015a). Efficient moving point handling for incremental 3D manifold reconstruction. In *International Conference on Image Analysis and Processing*.

Romanoni, A. and Matteucci, M. (2015b). Incremental reconstruction of urban environments by edge-points Delaunay triangulation. In *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.*

Ryde, J. and Hu, H. (2010). 3D mapping with multi-resolution occupied voxel lists. *Autonomous Robots*, 28:169.

Sanfourche, M., Vittori, V., and Besnerais, G. L. (2013). eVO: A realtime embedded stereo odometry for MAV applications. In *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.*, Tokyo, Japan.

Schoeps, T., Sattler, T., Hane, C., and Pollefeys, M. (2015). 3D modeling on the go: Interactive 3D reconstruction of large-scale scenes on mobile devices. In *Proceedings of International Conference on 3D Vision*.

Shen, S., Michael, N., and Kumar, V. (2015). Tightly-coupled monocular visual-inertial fusion for autonomous flight of rotorcraft MAVs. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom.*, Seattle, WA.

Shi, J. and Tomasi, C. (1994). Good features to track. In *Proc. of the IEEE Intl. Conf. on Comput. Vis. and Pattern Recognition*.

Steder, B., Grisetti, G., Grzonka, S., Stachniss, C., Rottmann, A., and Burgard, W. (2007). Learning maps in 3D using attitude and noisy vision sensors. In *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.*, pages 644–649.

Tolman, E. C. (1948). Cognitive maps in rats and men. *Psychological Review*, 55:189–208.

Toon, G., Marnix, N., Tinne, T., and Luc, V. G. (2007). Omnidirectional vision based topological navigation. *International Journal of Computer Vision*, 74(3):219–236.

Triebel, R., Pfaff, P., and Burgard, W. (2006). Multi-level surface maps for outdoor terrain mapping and loop closing. In *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.*

Triggs, B., McLauchlan, P. F., Hartley, R. I., and Fitzgibbon, A. W. (1999). Bundle adjustment - a modern synthesis. In *International Workshop on Vision Algorithms: Theory and Practice*.

Whelan, T., Kaess, M., Johannsson, H., Fallon, M., Leonard, J., and McDonald, J. (2014). Real-time large scale dense RGB-D SLAM with volumetric fusion. *Intl. J. of Robotics Research*.

Whelan, T., Salas-Moreno, R. F., Glocker, B., Davison, A. J., and Leutenegger, S. (2016). Elasticfusion: Real-time dense SLAM and light source estimation. *Intl. J. of Robotics Research*.

Wilhelms, J. and Gelder, A. V. (1992). Octrees for faster isosurface generation. *ACM Trans. Graph.*, 11(3):201–227.

Wu, M., Lam, S.-K., and Srikanthan, T. (2017). A framework for fast and robust visual odometry. *IEEE Transaction on Intelligent Transportation Systems*, 18:3433–3448.

Yang, Z. and Shen, S. (2016). Tightly-coupled visual-inertial sensor fusion based on IMU pre-integration. Technical report, Hong Kong University of Science and Technology.

Zhang, J. and Singh, S. (2014). LOAM: Lidar odometry and mapping in real-time. In *Proc. of Robot.: Sci. and Syst.*

Zuiderveld, K. (1994). *Graphics Gems IV*. Academic Press Professional, Inc.