2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
September 24–28, 2017, Vancouver, BC, Canada

Building Maps for Autonomous Navigation Using Sparse Visual SLAM Features

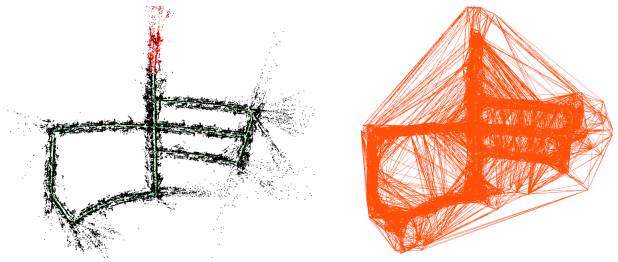
Yonggen Ling and Shaojie Shen

Abstract— **Autonomous navigation, which consists of a systematic integration of localization, mapping, motion planning and control, is the core capability of mobile robotic systems. However, most research considers only isolated technical modules. There exist significant gaps between maps generated by SLAM algorithms and maps required for motion planning. This paper presents a complete online system that consists in three modules: incremental SLAM, real-time dense mapping, and free space extraction. The obtained free-space volume (i.e. a tessellation of tetrahedra) can be served as regular geometric constraints for motion planning. Our system runs in real-time thanks to the engineering decisions proposed to increase the system efficiency. We conduct extensive experiments on the KITTI dataset to demonstrate the run-time performance. Qualitative and quantitative results on mapping accuracy are also shown. For the benefit of the community, we make the source code public.**

I. Introduction

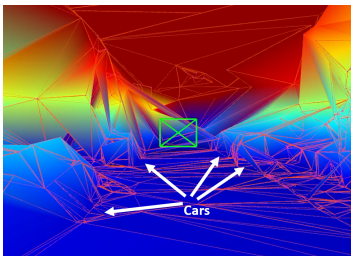
Dense 3D maps are fundamental components of autonomous robotic navigation as they serve as the perception input for path planning, mobile manipulation, traversability analysis, exploration, etc. In contrast to fine-grained 3D reconstruction methods which aim to recover detailed surface structures [1], autonomous navigation requires 3D maps that are scalable to large-scale environments and can be reconstructed incrementally in real-time. Traditionally, such dense 3D navigational maps are built in two stages. The first step is to estimate poses of sensors (such as cameras and laser scanners) together with positions of sparse features/semi-dense textures using SLAM algorithms [2]–[4]. The second step is to create some form of volumetric dense maps by projecting 3D points/range scans/disparity maps onto the correct global poses. Popular map representations include occupancy grid [5], OctoMap [6, 7], elevation maps [8, 9], or representations directly from the point cloud [10, 11]. In this way, the dense 3D map is conditioned on the optimized sensor poses from SLAM. As the robot acquires more sensory measurements, robot poses and sparse features are continuously refined by SLAM algorithms. However, the dense 3D map cannot be updated immediately due to the computational burden involved in map update operations such as space (re)allocation and ray-casting. This becomes particularly problematic when large-scale loop closure is detected, where the geometric configuration of the whole SLAM system is changed, and re-generation of the dense 3D map becomes prohibitively expensive.

In many practical robotic systems, a trade-off is made by only keeping a local dense map for local motion planning and solving the global navigation problem in a topological way [12]–[14]. However, this has drawbacks, as previously generated dense maps are discarded and cannot be reflected in the current scene even if loop closures are detected. It is also hard to obtain a topological map that contains connectivity information for areas that are observed, but not physically traveled to by the robot. Recently, [15] and [16] are proposed, where the environment is reconstructed with deformation graph. Both RGB-D sensors and high performance GPU are used, which is not practical for robots working in outdoor



(a) The optimized features and poses.(b) Space subdivision into tetrahedra.

(c) Captured image with multi-level features.



(d) The corresponding view of the reconstructed dense 3D map.

Fig. 1. An illustration of the 3D map built on the KITTI05 sequence. (a) Given the optimized sparse features and poses (Sect. II-A), we select quality features to build the dense 3D map incrementally. (b) A convex hull and the space subdivision are shown (Sect. II-B). (c) A captured image. (d) The corresponding view of the dense 3D navigation map is presented. Color varies according to the height (Sect. II-D). We are able to distinguish the shape of cars on the road.

All authors are with the Department of Electronic and Computer Engineering, The Hong Kong University of Science and Technology, Hong Kong SAR, China. ylingaa@connect.ust.hk, eeshaojie@ust.hk. This work was supported by HKUST project R9341 and HKUST institutional studentship.

environments or with limited computation resources.

It would be beneficial to have a unified dense map representation that incorporates all SLAM updates as soon as they become available. Moreover, to plan smooth trajectories, path planning algorithms prefer temporally consistent maps. Traditional stereo matching algorithms [17, 18], which only makes use of spatial correlations between stereo images, are not ready for temporally consistent maps. Motivated by the recent development of planning that involves geometric constrains [19], we find that some light-weight mapping approaches [20]–[25] developed in the graphics society have potentials for real time dense map generation and motion planning in large-scale environments. These algorithms build a 3D model from a sparse point cloud using 3D Delaunay triangulation and then figure out which part of it is empty or occupied using visibility constraints. Delaunay triangulation is powerful for its property [26]: the circumscribing sphere of tetrahedron dose not contain a vertex in its interior, which avoids degenerate shapes; the more the points are dense the more the model are fine. However, for real applications using Delaunay triangulation and visibility constraints, there are challenges ahead. Firstly, most of the developed algorithms run offline in a batch operation [24, 25], or run incrementally but assuming cameras poses are fixed or known beforehand [21]–[23], which is not true for online applications where cameras poses are not known in advance and are estimated incrementally (and refined over time). Secondly, except the work of [20], real-time performance at video frame rate is not reported. [20] runs online in small limited indoor environments. As shown in Sect. IV-D, [20] is not able to run real time in large-scale outdoor environments. Thirdly, loop closures in large-scale environments are ignored [20]–[25]. There is no chance to correct drifts of the 3D reconstruction. In additions, environment topology is missing and thus it is impossible to find shortcuts between locations. Last but not least, to the best of our knowledge, the whole online system pipeline, from incremental SLAM to dense 3D reconstruction, is not presented.

In this work, we build on the main ideas of incremental free-space carving [20] and the state-of-the-art sparse feature based SLAM system ORB-SLAM2 [27] to design a complete mapping approach for robotic applications with the real-time requirement on a CPU-only setting in large-scale environments. The main contributions are:

- The system includes all essential modules: tracking, local bundle adjustment, global optimization with large-scale loop closures, incremental Delaunay triangulation, space carving using visibility constraints, and free-space generation for path planning.
- Increased robustness, efficiency, and usability in large-scale environments over [20] thanks to the proposed improvements on vertexes selection, map update principle, lazy update scheme, engineering implementations, etc..
- Extensive experiments on the KITTI dataset for performance evaluation.
- We release the code as an open-source package available at: `https://github.com/ygling2008/lightweight_mapping`.

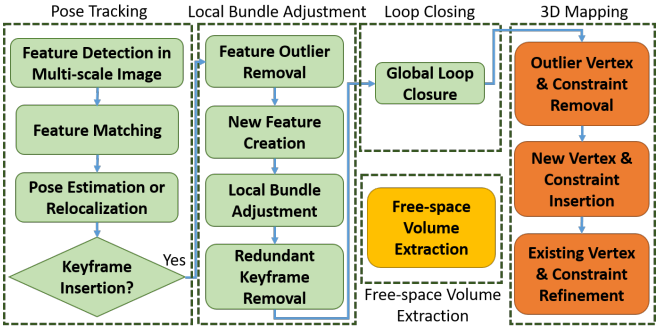


Fig. 2. The complete system pipeline. It consists of five threads running simultaneously on a multi-core CPU. The pose tracking thread estimates camera poses for every frame and decides whether to insert a new keyframe. If a new keyframe is inserted, the local bundle adjustment thread optimizes local poses and feature positions. Loop closures detection and the following global optimization are done in a separate thread. The 3D mapping thread converts the optimized sparse features into dense volumetric representation. The free-space volume extraction thread outputs the latest free-space volume periodically for motion planning usage.

In this work, we use stereo ORB-SLAM2 [27] as the visual SLAM module of our system and our system is also compatible with other SLAM pipelines. The manifold constraint and edge features of [21]–[23] can also be integrated to our proposed system for further mapping improvement. Our system runs real-time with a commodity laptop CPU. The online extracted free-space volume, which consists in regular shapes (i.e. tetrahedra), can be served as the input for planning algorithms [19].

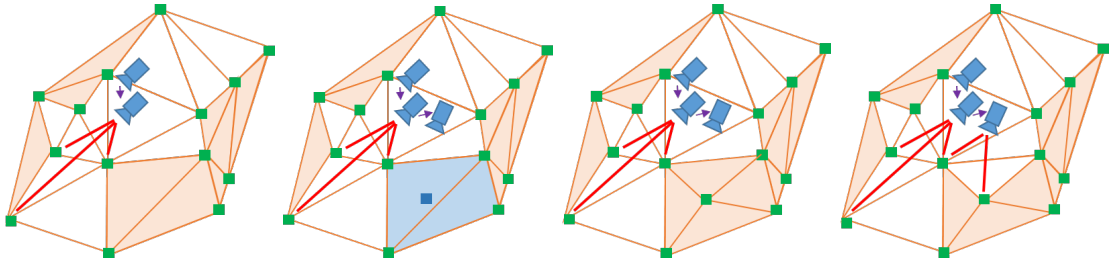
The rest of the paper is structured as follows. The proposed framework is presented in Sect. II. Detailed engineering considerations is discussed in Sect. III. We show the experimental evaluations in Sect. IV. Sect V draws the conclusion and points out possible future extensions.

II. Proposed Framework

The framework of our system is shown in Fig. 2. Five threads run simultaneously to utilize the multi-core architecture. We follow the same feature-based SLAM pipeline as ORB-SLAM2 [27] for the pose tracking, local bundle adjustment, loop closing and global optimization. The 3D mapping module converts the optimized sparse features into dense volumetric representation. The free-space volume extraction module outputs free-space volumes for path planning purpose.

A. Feature-Based SLAM Pipeline

1) Pose Tracking: The pose tracking thread estimates the latest camera pose and decides whether to insert a new keyframe or not. FAST corners with ORB descriptors are extracted from every incoming image. Features are matched between the current frame and the last keyframe. The latest pose is then optimized using motion-only bundle adjustment. If tracking is lost, global relocalization is enabled to relocate the pose. We follow the criteria in [27] for new keyframe insertion.



(a) The initial map. (b) Conflicts due to vertex insertion. (c) Re-triangulation. (d) Carving the empty space.

Fig. 3. A 2D illustration of the mapping process, for simplicity. We subdivide the 3D space into connected tetrahedra (triangulates) with vertexes (green). Each tetrahedron is associated with an occupancy label denoted as empty (white) or occupied (light orange). (a) An initial space configuration. (b) A new vertex (dark blue) is inserted. The empty circumsphere property of light blue tetrahedra is broken. (c) The tetrahedra in conflict are deleted and retriangulated. (d) New visibility constraints related to the newly inserted vertex carve away the concerned tetrahedra.

2) Local Bundle Adjustment: The local bundle adjustment thread processes new keyframes from the pose tracking thread and updates the local information of nearby keyframe poses and features. New feature points are triangulated once enough feature correspondences among keyframes are obtained. After the local bundle adjustment, outliers are detected and removed.

3) Loop Closure Detection and Global Optimization: The loop closing thread searches possible loops for every keyframe using a bag-of-word place recognizer. Pose graph optimization is then applied to correct drift after long-term operations. We do not perform full bundle adjustment the same as ORB-SLAM2 because the resulted accuracy gain is not significant.

B. Dense Mapping for Autonomous Navigation

Given the sparse 3D point cloud optimized by the SLAM pipeline, the 3D mapping thread subdivides the 3D space using 3D Delaunay triangulation and then carves away the space using the visibility constraints. An illustration is shown in Fig. 3.

3D Delaunay triangulation takes a sparse cloud of 3D points, $\mathbf{P} = \{\mathbf{p}_0, \mathbf{p}_1, ..., \mathbf{p}_{n-1}\}$, sampled on the unknown surface of the surroundings as input. It computes a set of tetrahedra, $\mathbf{T} = \{\mathbf{t}_0, \mathbf{t}_1, ..., \mathbf{t}_{m-1}\}$, that partition the convex hull of $\mathbf{P}$. Each vertex of a tetrahedron $\mathbf{t}_i$ is one of the points in $\mathbf{P}$. The intersection between two neighboring tetrahedra is a face (triangle) $\mathbf{f}_i$. A label is associated to every tetrahedron as either empty or occupied to denote the occupancy.

A visibility constraint is defined as the line segment between a camera center and an optimized feature. Tetrahedra that have intersections with any visibility constraints will be marked as empty, as shown in Fig. 4.

The entire 3D space is labeled as occupied in the beginning and then subdivided with tetrahedra along with the insertion or deletion of input vertexes. Afterward, visibility constraints are checked for features observed in different keyframes. By this way, the 3D map is built incrementally. We list basic operations needed:

1) Vertex Insertion: Vertexes are inserted as the estimation processes (Section. II-A.2). If a new vertex is inserted into the current 3D Delaunay triangulation, the empty circumsphere property will be broken. We first delete all the tetrahedra that contain the newly inserted vertex in their circumspheres, for which a hole appears. This hole is then retriangulated. We need to check whether the newly added tetrahedra intersect existing visibility constraints afterward. The procedure is illustrated in Fig. 3 (b, c).

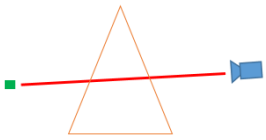


Fig. 4. A 2D illustration of a visibility constraint. The segment (red) between a camera center (blue) and an observed feature point (green) intersects a tetrahedron (orange). The tetrahedron is marked as empty.

2) Constraint Insertion: The insertions of new feature points (Sect. II-A.2) and the process of feature matching (Sect. II-A.1) introduce new visibility constraints. We apply the method in [28] for fast searching related tetrahedra that intersect new visibility constraints, and label them as empty.

3) Vertex Deletion: Outlier features may be removed after local bundle adjustment (Sect. II-A.2). For the vertex deletion operation, we first delete all the visibility constraints induced by the vertex to be deleted. The vertex and its incident tetrahedra are then deleted. A hole is formed and then retriangulated.

4) Constraint Deletion: Along with vertex deletion, related visibility constraints are removed. We traverse tetrahedra that intersect with the visibility constraints to be deleted using Gargallo's algorithm [28], check whether tetrahedra intersect other visibility constrains or not. Tetrahedra will be remained empty if they intersect visibility constraints other than the one to be deleted, or set to occupied otherwise. This operation is much faster than the dissociation event in [20], where all tetrahedra are traversed and checked.

5) Vertex Refinement: As feature positions are re-estimated after local bundle adjustment (Sect. II-A.2) or global optimization (Sect. II-A.3). We save the set of visibility constraints incident to these vertexes and execute vertex deletion and insertion operations to update the vertexes. We can not merely move the vertexes; otherwise, the empty circumsphere property of the tetrahedra may be broken. We

then add back the visibility constraints saved previously to the 3D map. For efficiency, we propose a lazy update scheme detailed on Sect. III-F.

6) Constraint Refinement: Apart from feature positions, poses of the camera are also recomputed after local bundle adjustment (Sect. II-A.2) or global optimization (Sect. II-A.3). Visibility constraints related to updated camera poses and feature positions are removed and then re-inserted to maintain the up-to-date 3D map. For efficiency, we propose a lazy update scheme detailed on Sect. III-F.

We summarize the run-time complexity of basic operations in Table I. Suppose that the number of vertexes in the 3D map is N_v and the number of tetrahedra is N_t. For each vertex, there are no than M_c incident visibility constraints, while for each visibility constraint, there are no more than M_t intersecting tetrahedra. We make full of tree data structures for quickly locating the vertexes. In actual cases, M_t and M_c are small and overall complexity is not high for all the concerned processing.

Operation	Complexity
Vertex Insertion	$O(logN_t)$
Constraint Insertion	$O(M_t)$
Vertex Deletion	$O(M_cM_t)$
Constraint Deletion	$O(M_t)$
Vertex Refinement	$O(M_cM_t)$
Constraint Refinement	$O(M_t)$

TABLE I

SUMMARY OF THE OPERATION COMPLEXITY.

C. Loop Closure

When a large loop is detected, pose graph optimization is applied to get a fast response of the camera poses drift. During the optimization, both SLAM module (Sect. II-A) and the mapping (Sect. II-B) runs as usual. These two modules will be informed once the global optimization is done. Features locations are updated with the latest camera poses. Map points and observations are merged if they are similar in the ORB descriptor space as well as consistent with the camera transformations. Finally, vertex and constraint refinement are performed on local vertexes and constraints to update the 3D model.

D. Free-Space Volume Extraction

A free-space volume of the 3D space is simply the set of tetrahedra that are empty, and the environment surface is a set of faces between empty tetrahedra and occupied tetrahedra. This thread periodically visits tetrahedra and their faces that are near the latest camera pose, constructs a local map, which consists of empty tetrahedra as well as local environment surface, and outputs the local map to the path planning module.

III. ENGINEERING CONSIDERATIONS

We take careful engineering considerations on the implementations to increase the overall system robustness, efficiency, and usability.

A. Detecting Features in Multiple Image Scales

We detect corners in multi-scale levels of the image resolution in the pose tracking thread (Sect.II-A.1), which not only increases the tracking robustness in textureless environments, but also helps to model the geometry of the 3D space. Multi-scale corners represent the depth discontinuity of the surface, and areas without textures are usually planar pieces corresponding to faces of tetrahedra. Edges points in [22, 23] may be included as the future work.

B. Map Update after Local Bundle Adjustment

Unlike the previous work [20]–[25] that updates the 3D model every K_f frames, we perform map update (Sect. II-B) only after a keyframe is inserted and local bundle adjustment is finished, or loops are detected and global optimization is done. We find that feature positions, as well as camera poses, are not accurate without local bundle adjustment. Early integration of new vertexes and constraints leads to a number of unnecessary refinement operations.

C. Batch Updates

When new vertexes and visibility constraints come along with new keyframe arrival, we start with vertex insertion (Sect. II-B.1) for all new vertexes and saving all visibility constraints to be checked (we do not check in this step), then perform constrain insertion (Sect. II-B.2) for all new visibility constraints. Finally, we do the constraint check needed in the first step. Conversely, adding each pair of vertex and constraint one by one leads to a number of redundant constraint check, which slows down the process.

D. Constraint List

As for the constraint deletion (Sect. II-B.4), we have to check whether tetrahedra intersect the remaining visibility constrains, which is a very time-consuming step. An alternative approach is to associate each tetrahedron with a constraint list that records all intersecting visibility constraints. This constraint list is updated together with the constraint insertion and deletion operation. Tetrahedra whose constraint list size are positive will be empty and there is no need for the additional check as before.

E. Vertex Selection

Some features may not be well-constrained due to insufficient observations or parallax, we get rid of unnecessary operations by merely updating 3D maps with well-constrained features and their associated visibility constraints. The used metric is feature position confidence: Let a feature $\mathbf{p}_i$ be observed in k keyframes, with corresponding image coordinates are $\mathbf{u}_i^1$, $\mathbf{u}_i^2$, ..., $\mathbf{u}_i^k$. Rotations and translations of these k keyframes are $\mathbf{R}_1$, $\mathbf{R}_2$, ..., $\mathbf{R}_k$ and $\mathbf{t}_1$, $\mathbf{t}_2$, ..., $\mathbf{t}_k$ respectively. Suppose the ground truth position of $\mathbf{p}_i$ is $\tilde{\mathbf{p}}_i$, and the current estimate is $\hat{\mathbf{p}}_i$. We apply the Gaussian Newton approach which iteratively solves for $\Delta\mathbf{p}_i = \tilde{\mathbf{p}}_i - \hat{\mathbf{p}}_i$ such that:

$$arg\min_{\Delta\mathbf{p}} \sum_{j=1}^{k} ||\pi(\mathbf{R}_j\hat{\mathbf{p}}_i + \mathbf{t}_j) - \mathbf{u}_i^j + \mathbf{J}_j^T\Delta\mathbf{p}||^2, \qquad (1)$$

where $\pi(\cdot)$ is the image 3D-2D projection function and $\mathbf{J}_j$ is the first-order Jacobian of $\pi(\mathbf{R}_j\hat{\mathbf{p}}_i + \mathbf{t}_j) - \mathbf{u}_i^j$. The solution to (1) is

$$\Delta\mathbf{p}_i = (\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\mathbf{r}, \tag{2}$$

where $\mathbf{J} = \begin{bmatrix} \mathbf{J}_1^T \\ \mathbf{J}_2^T \\ \dots \\ \mathbf{J}_k^T \end{bmatrix}$ and $\mathbf{r} = \begin{bmatrix} \pi(\mathbf{R}_1\hat{\mathbf{p}}_i + \mathbf{t}_1) - \mathbf{u}_i^1 \\ \pi(\mathbf{R}_2\hat{\mathbf{p}}_i + \mathbf{t}_2) - \mathbf{u}_i^2 \\ \dots \\ \pi(\mathbf{R}_k\hat{\mathbf{p}}_i + \mathbf{t}_k) - \mathbf{u}_i^k \end{bmatrix}$.

We assume that all the detected corners $\mathbf{u}_i^k$ are corrupted by Gaussian noise with zero mean and variance σ^2. We ignore covariances of poses for simplicity. As $\hat{\mathbf{p}}_i$, $\mathbf{R}_j$ and $\mathbf{t}_j$ are constants, the covariance of $\mathbf{r}$ is $\Sigma_r = \sigma^2\mathbf{I}$. And the covariance of $\tilde{\mathbf{p}}_i$ is the same as the covariance of $\Delta\mathbf{p}_i$:

$$\Sigma_{\Delta\mathbf{p}_i} = ((\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T)^T\Sigma_r(\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T \tag{3}$$

$$\approx (\sigma^2\mathbf{J}^T\mathbf{J})^{-1}. \tag{4}$$

If the position estimation of $\mathbf{p}_i$ is under-constrained, the ratio (denoted as r_p) between the smallest and largest eigenvalue of $\mathbf{J}^T\mathbf{J}$ is small. We select features with $r_p > \tau_p$ (See Sect. IV-D for details about how τ_p affects the mapping performance).

F. Lazy Update of Vertex Refinement and Constraint Refinement

During the experiments, we find that most of the mapping time are spend on operations of vertex refinement and constraint refinement due to the incremental estimation process (this conforms the complexity Table I). We propose a lazy update scheme that greatly improve mapping efficiency subject to little accuracy loss: each vertex has two positions $\mathbf{p}_{old}$ and $\mathbf{p}_{new}$, where $\mathbf{p}_{old}$ denotes the latest position in the 3D model, and $\mathbf{p}_{new}$ denotes the latest position as SLAM processes. We set $\mathbf{p}_{old}$ to be $\mathbf{p}_{new}$ as well as perform vertex refinement (Sect. II-B.5) if and only if $||\mathbf{p}_{new} - \mathbf{p}_{old}||_2 > \tau_v$; Similarly, each visibility constraint has four end points e_{old}^0, e_{new}^0, e_{old}^1 and e_{new}^1. We set e_{old}^0 to be e_{new}^0, e_{old}^1 to be e_{new}^1, perform constraint refinement (Sect. II-B.6) if and only if $||e_{old}^0 - e_{new}^0||_2 + ||e_{old}^1 - e_{new}^1||_2 > \tau_e$. From the experiment (Sect. IV-D), this lazy update scheme works effectively. Our proposed lazy update scheme is different from [20] that discards constraint refinement.

IV. Experimental Results

We evaluate our system performance on the large-scale car-driving KITTI dataset with ground truth poses from GPS and Velodyne laser scanner data for comparison. Sequences 00, 02, 05, 06, 07 and 09 contain loops that are supposed to be detected. Our system runs in real time and processes stereo images at the same frame rate that they are captured (10 Hz). All the experiments are carried out in a commodity Lenovo laptop Y510 with an i7-4720HQ CPU (2.4GHz) and 16G of RAM. We build our system on top of an open-source sparse feature-based system ORB-SLAM2 [27]. All algorithms are implemented in C++, with ROS as the interfacing robotics middleware. We set $\tau_p = 0.01$, $\tau_v = 0.5$, $\tau_e = 1.0$ for all the evaluations except Sect. IV-D. We utilize the CGAL library for 3D Delaunay triangulation and efficient access to tetrahedra traversals. [1]

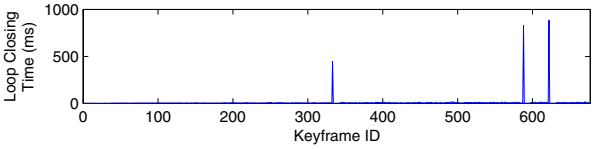


Fig. 5. The computation time of the loop closing thread with respect to the keyframe index. We encounter three global loop closures.

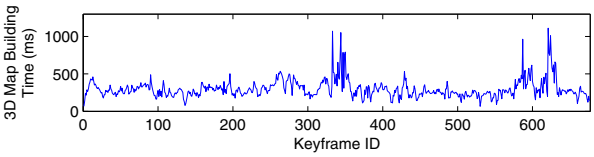


Fig. 6. The computation time of the 3D mapping thread with respect to the keyframe index. If a global loop closure occurs, the computation time will increase. The computation time is bounded and only related to the number of sparse features observed in the keyframe neighborhood (Sect. II-C). It does not grow with respect to the total number of keyframes (Sect. II-B).

A. Run-time Performance

The real-time requirement is a vital aspect of our proposed system. While most algorithms have focused on accuracy, the goal of our proposed system is to achieve a good trade-off between the accuracy and run-time performance. We use the results obtained from an experiment runs on the KITTI 05 sequence to illustrate our system processing time. The KITTI 05 sequence lasts for 276 seconds, with a long distance in both dimensions and three global loops detected by our system. The mean processing time for the pose tracking thread is 20.7 ms with standard deviation 6.9 ms, while the mean processing time for the local bundle adjustment thread is 166.8 ms with standard deviation 88.6 ms. Since the computation time spent by the loop closing thread and the 3D mapping thread are highly related to the occurrence of global loop closure, we plot them in Fig. 5 and Fig. 6. From these two figures, we see that both the maximum computation time of these two threads are less than 1000 ms. More importantly, though the computation time of loop closing thread scales w.r.t. the pose graph size, the computation time of 3D mapping thread is only related to the locally observed features. Moreover, the keyframe insertion frequency is roughly 2-3 Hz; thus both the loop closing thread and 3D mapping thread are able to process it in real time. The majority of the computation time spent by the 3D mapping thread is on constraint insertion and deletion operations. These operations can be processed in a parallel way and greatly accelerated using GPUs. We left the GPU implementation as future work.

Another advantage of our system is the nice property that the complexity of our free-space volume extraction is linear with respect to the number of keyframes: The number of vertexes is linear with respect to the number of keyframes, while the number of tetrahedra is linear with respect to the

[1] `http://www.cgal.org/`

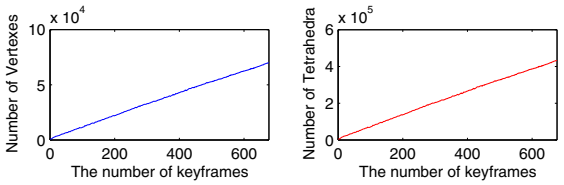
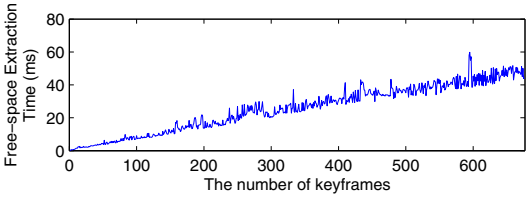
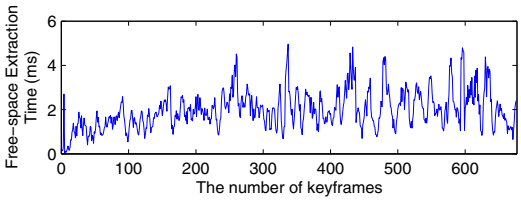


Fig. 7. The number of vertexes (left) is linear with respect to the number of keyframes, and the number of tetrahedra (right) is linear with respect to the number of keyframes too.



(a) Global map reconstruction.



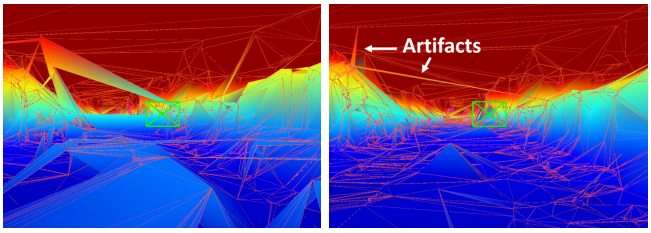
(b) Local map reconstruction.

Fig. 8. (a) The computation time of free-space volume extraction is linear with respect to the number of keyframes. (b) We can select to only extract free space related to keyframes nearby current pose. This constructs local maps. A small disturbance occurs due to the varying system overload.

number of vertexes (Fig. 7). We traverse all the tetrahedra to get the free-space volume, whose consumption time is linear with respect to the number of tetrahedra (Fig. 8 (a)). This nice property indicates that we are able to extract the free-space volume within a bounded time by visiting tetrahedra related to the closest K_t (turnable, $K_t = 10$ in this paper) neighboring keyframes (Fig. 8 (b)).

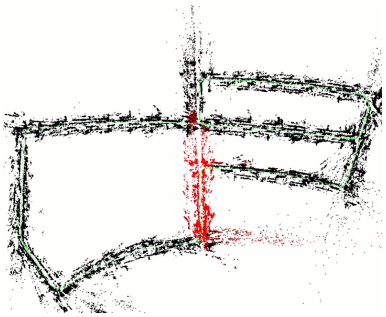
B. Qualitative Analysis of Mapping Performance

We present the qualitative results in this section. The results obtained by testing our system on the KITTI 05 sequence are shown in Fig. 1. The optimized features and poses are plotted in sub-figure (a). These features are detected in multi-level image pyramids to model the space geometry (Sect. III-A), as highlighted with green circles in sub-figure (c) (the relative sizes of the circles denote the relative scales of corresponding features detected). We ignore features that are under-constrained (Sect. III-E). The remaining features serve as vertexes to subdivide the space (Sect. II-B). The convex hull and separation details can be found in sub-figure (b). A captured image among the sequence is shown in sub-figure (c), and its following view is shown in sub-figure (d). The color varies according to the height of the local environment. The green virtual camera represents the estimated capture camera. From the reconstructed local surface, we see that our system labels the occupancy (empty or occupied) of the tetrahedra well, such that it approximates the real scene structure. For example, we are able to distinguish the cars on the road in sub-figure (d).



(a) Before global loop closure. (b) After global loop closure.

(c) The captured image with multi-level features.



(d) The estimated trajectory.

Fig. 9. (a) Before global loop closure is encountered, the built map is disordered due to the significant drift after long-term operations. b) After global loop closure, we merge and update the local surroundings in real-time (Sect. II-C). (c) The captured image with multi-level features to model the geometry (Sect. III-A). (d) The driving car goes back to the origin (shown in red) and closes a large global loop. There are a few artifacts (see sub-figure (b)) due to the lack of visibility constraints that carve the empty space. These artifacts are usually long and thin, and do not make a big difference on the free-space volume for motion planning. They can be further removed by using methods in [21]–[23].

We also demonstrate the qualitative results before and after a global loop closure in Fig. 9. The driving car goes back to the origin after traveling a long distance (sub-figure (d)). Significant drift occurs, and thus the built map is disordered (sub-figure (a)). After the global optimization, we update the 3D map (sub-figure (b)). The captured image with multi-level features is shown in sub-figure (c). There are a few artifacts (see sub-figure (b)) due to the lack of visibility constraints that carve the empty space. These artifacts are usually long and thin, and do not make a significant difference on the free-space volume. Most of these artifacts will be disappeared along with insertion of new vertexes and visibility constraints. The remaining can be further removed by integrating methods in [21]–[23]. According to available computational resources, we can tune the system complexity by tuning the number of sparse features in the system. Increasing the number of sparse features helps to reduce the occurrence of the artifact. This is because the related tetrahedron will be split into smaller tetrahedra, and more visibility constraints will be inserted. More results can be found in the accompanying video.

Sequence \ Difference PCT	$< 1.5\%$	$< 3.0\%$	$< 6.0\%$	$< 12.0\%$
KITTI 00	37.7661	50.5732	64.2892	77.4659
KITTI 01	28.2305	40.1803	52.7449	63.1427
KITTI 02	43.1737	56.3444	69.4084	80.7317
KITTI 03	16.6989	29.7078	47.2295	64.3512
KITTI 04	33.5860	45.8266	59.2409	72.5650
KITTI 05	39.4686	51.6673	64.1485	78.1123
KITTI 06	38.4727	51.3787	64.7964	76.3172
KITTI 07	37.9296	49.6814	62.3120	75.1020
KITTI 08	36.5144	47.7206	59.5570	71.0865
KITTI 09	42.0707	54.7159	67.5800	79.0824
KITTI 10	42.3470	55.2040	68.4565	80.3366

TABLE II

DEPTH ACCURACY ON THE KITTI DATASET. WE CALCULATE THE PERCENTAGE OF PIXELS WITHIN DIFFERENT RANGES PROPORTION TO THE AVERAGE SCENE DEPTH.

Sequence \ Safe Margin PCT	$< 1.5\%$	$< 3.0\%$	$< 6.0\%$	$< 12.0\%$
KITTI 00	80.5224	89.8671	92.4686	94.5498
KITTI 01	68.5774	76.3287	84.0849	89.7499
KITTI 02	84.2877	92.4302	94.6705	96.4235
KITTI 03	82.6347	85.5102	89.1914	92.9539
KITTI 04	82.4458	88.3155	91.3178	93.2261
KITTI 05	78.9738	83.9451	85.8857	88.7400
KITTI 06	77.2445	82.1994	85.1913	91.1199
KITTI 07	80.6444	88.0556	91.2381	94.1015
KITTI 08	80.5850	86.5718	90.8511	92.6113
KITTI 09	82.4873	90.6764	93.6242	95.4036
KITTI 10	86.0530	92.6589	94.4553	96.2711

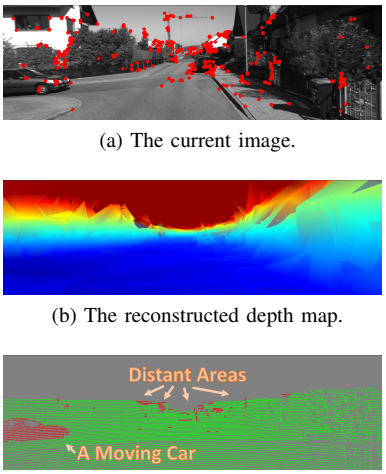
TABLE III

SAFE DEPTH ACCURACY ON THE KITTI DATASET. WE CALCULATE THE PERCENTAGE OF PIXELS WITHIN DIFFERENT MARGINS PROPORTION TO THE AVERAGE SCENE DEPTH.

C. Mapping Accuracy

We compare the accuracy of the built 3D map with the ground truth Velodyne laser scanner data. This is achieved by projecting the 3D map and Velodyne laser scanner measurements into the keyframe views and taking the difference between them. The average scene depth of the captured images is 15 meters. We enumerate the depth difference from 1.5% (0.225 meters) to 12% (1.8 meters) of the average scene depth, and calculate the percentage of depth differences within the threshold range. Only part of the results is shown in Table II for simplicity. About 65% of pixel depth are within a 6% average scene depth difference of the ground truth depth, while about 75% of pixel depth are within a 12% average scene depth difference of the ground truth depth.

We define another metric, namely the safe depth percentage, to count the portion of depth that is less than the ground truth depth. Safe margins start from 1.5% average scene depth (0.225 meters) and end at 12% average scene depth (1.8 meters). The estimated depth, which is less than the ground truth depth plus the safe margin, is supposed to be the safe depth. Part of the results is presented in Table III. The safe depth percentage is significantly higher than the accuracy percentage, which is good for autonomous navigation. The percentage is about 90% for depth differences within 6% average scene depth while more than 90% for depth differences within 12% average scene depth. We visualize the safe and unsafe regions in Fig. 10. Ground truth depth is provided by a Velodyne laser scanner. Pixels without ground truth depth are marked in gray. Pixels are marked in green if the estimated depth value is smaller the safe margin (12% of the average scene depth) plus ground truth depth ("safe"); Pixels are marked in red otherwise ("unsafe"). We can see the depth of dynamic objects such as moving vehicles cannot be estimated for the same reason that the dependent SLAM pipeline is unable to detect dynamic objects. The depth of distant areas is also poorly estimated. However, we argue that this does not pose a significant threat to safe navigation. As the system processes, more and more sparse features will be detected and included in the reconstructed map, which turns the "unsafe" regions into "safe" regions.



(a) The current image.

(b) The reconstructed depth map.

(c) Visualization of safe and unsafe regions.

Fig. 10. (a) One of the image in the KITTI 05 sequence. Sparse SLAM features are shown as red dots. (b) The reconstructed depth map by our algorithm. (c) Visualization of safe and unsafe regions. Ground truth depth is provided by a Velodyne laser scanner. Pixels without ground truth depth are marked in gray. Pixels are marked in green if the estimated depth value is smaller the safe margin (12% of the average scene depth) plus ground truth depth ("safe"); Pixels are marked in red otherwise ("unsafe"). We can see the depth of dynamic objects such as moving vehicles cannot be estimated for the same reason that the dependent SLAM pipeline is unable to detect dynamic objects. The depth of distant areas is also poorly estimated. However, we argue that this does not pose a significant threat to safe navigation.

D. Improvements Compared to the Free-Space Carving [20]

We compared the proposed system with the state-of-the-art real-time system [20] to show the effectiveness and efficiency of our presented engineering considerations (Sect. III). For all the testing, we set $\tau_e = 2 * \tau_v$ for simplicity. The KITTI 05 sequence is used. The evaluation is conducted in one thread. Average map update time after keyframe insertion, difference PCT at 12% average scene depth, and safe margin PCT at 12% average scene depth are used for evaluation. Same vertexes and visibility constraints are used for all experiments. Table IV summarizes the results. Test 0 is the baseline method [20], we find that this method can not run in real-time at large scale environments where lots of vertexes and visibility constraints are involved (plus potential outliers and refinements). In Test 1, only improvements presented in

Test	τ_p	τ_v	Time (ms)	Difference PCT ($< 12\%$)	Safe Margin PCT ($< 12\%$)
0	-	-	2478.6	79.9689	88.9376
1	0	0	697.1	78.8697	88.3034
2	0.005	0	683.0	79.1632	88.5590
3	0.01	0	570.7	78.8599	88.5630
4	0.02	0	424.2	72.7353	82.1562
5	0.04	0	102.6	58.6319	68.5692
6	0.01	0.1	383.6	78.7819	88.5630
7	0.01	0.5	309.8	78.1123	88.7400
8	0.01	1	285.8	69.5595	78.7316
9	0.01	5	270.1	65.0463	75.7322

TABLE IV

IMPROVEMENTS COMPARED TO FREE-SPACE CARVING [20] (TEST 0) ON KITTI 05.

(Sect. II-B.4, Sect. III-C and Sect. III-D) are applied, resulting in a significant gain on the mapping efficiency. In addition to engineering considerations in Sect. II-B.4, Sect. III-C and Sect. III-D, we apply vertex selection (Sect. III-E) and lazy update scheme (Sect. III-F) in Test 2-9. Test 2-5 show how vertex selection affects the efficiency against mapping accuracy. As expected, a tighter vertex selection criteria leads to fewer vertexes in Delaunay triangulation and visibility constrains, which reduces the computation time at cost of less mapping accuracy. We fix $\tau_p = 0.01$ for Test 6-9. The lazy update scheme does not affect the mapping accuracy if τ_v small than a certain threshold (let say 0.5), this is because of the sparse nature of the point cloud. Overall, setting $\tau_p = 0.01$ and $\tau_v = 0.5$ and using all the proposed engineering considerations have significant gains in mapping efficiency subject to little accuracy loss compared to the baseline state-of-the-art real-time method [20].

V. CONCLUSION AND FUTURE WORK

Motivated by the recent advancement of motion planning algorithms requiring temporally consistent maps and geometric constraints, we present a complete system that includes three needed modules: incremental SLAM, real-time dense mapping, and free space extraction. Our system runs real time with a commodity laptop CPU thanks to the proposed improvements on engineering considerations. Extensive experiments on the KITTI dataset show the capability in large-scale environments with balanced run-time performance, qualitative results and mapping accuracy.

There are potential improvements that could be made in the future. The first is to further reduce the occurrence of floating artifacts (see Fig. 9 (b) and the accompanying video) in the reconstructed maps, and the second is to introduce more supporting vertexes that may not be sparse SLAM features (such as edge pixels) to model the space geometry more accurately. Integration with planning algorithms and hardware platforms (such as UAV) will be done also.

REFERENCES

[1] R. A. Newcombe, S. Lovegrove, and A. J. Davison, "DTAM: Dense tracking and mapping in real-time," in *IEEE International Conference on Computer Vision*, 2011, pp. 2320–2327.

[2] S. Shen, N. Michael, and V. Kumar, "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, May 2015.

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

[4] J. Zhang and S. Singh, "LOAM: Lidar odometry and mapping in real-time," in *Proc. of Robot.: Sci. and Syst.*, 2014.

[5] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46 – 57, June 1989.

[6] D. Meagher, "Geometric modeling using octree-encoding," *Computer Graphics and Image Processing*, vol. 19, 1982.

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

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

[9] H. H. Julian Ryde, "3d mapping with multi-resolution occupied voxel lists," *Autonomous Robots*, vol. 28, p. 169, 2010.

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

[11] T. Schops, T. Sattler, C. Hane, and M. Pollefeys, "3D modeling on the go: Interactive 3D reconstruction of large-scale scenes on mobile devices," in *International Conference on 3D Vision*, 2015.

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

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

[14] A. Bry and N. Roy, "Rapidly-exploring random belief trees for motion planning under uncertainty," in *Proc. of the IEEE Intl. Conf. on Robot. and Autom.*, 2011.

[15] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger, "Elasticfusion: Real-time dense slam and light source estimation," *Intl. J. of Robotics Research*, 2016.

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

[17] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2008.

[18] A. Geiger, M. Roser, and R. Urtasun, "Efficient large-scale stereo matching," in *Asian Conference on Computer Vision*, 2010.

[19] M. Watterson and V. Kumar, "Safe receding horizon control for aggressive MAV flight with limited range sensing," in *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.*, 2015.

[20] D. Lovi, N. Birkbeck, D. Cobzas, and M. Jagersand, "Incremental free-space carving for real-time 3D reconstruction," in *3D Data Processing, Visualization, and Transmission*, 2010.

[21] V. Litvinov and M. Lhuillier, "Incremental solid modeling from sparse and omnidirectional structure-from-motion data," in *British Machine Vision Conference*, 2013.

[22] A. Romanoni and M. Matteucci, "Incremental reconstruction of urban environments by edge-points Delaunay triangulation," in *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.*, 2015.

[23] ——, "Efficient moving point handling for incremental 3D manifold reconstruction," in *International Conference on Image Analysis and Processing*, 2015.

[24] K.N. Kutulakos and S.M. Seitz, "A theory of shape by space carving," in *Proc. of the IEEE Intl. Conf. Comput. Vis.*, 1999.

[25] P. Labatut, J.-P. Pons, and R. Keriven, "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.*, 2007.

[26] J. D. Boissonnat, O. D. Faugeras, and E. L. Bras-Mehlman, "Representing stereo data with the delaunay triangulation," in *Proc. of the IEEE Intl. Conf. on Robot. and Autom.*, 1988.

[27] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras," in *ArXiv preprint arXiv:1610.06475*, 2016.

[28] P. Gargallo, "Modelisation de surfaces en vision 3D," *Master's thesis*, 2003.