IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 18, 2019, accepted January 3, 2020, date of publication January 15, 2020, date of current version January 28, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2966757

Real-Time Cloud Visual Simultaneous Localization and Mapping for Indoor Service Robots

YALI ZHENG[1], (Member, IEEE), SHINAN CHEN[2], AND HONG CHENG[1], (Senior Member, IEEE)

[1]School of Automation Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China
[2]Netease Company, Shenzhen, China

Corresponding author: Yali Zheng (zhengyl@uestc.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61971106, Grant 61603077, and Grant 61573084, in part by the China Postdoctoral Science Foundation funded project under Grant 2016M600732, and in part by the UESTC Fundamental Research Funds for the Central Universities of China under Grant ZYGX2016J076.

ABSTRACT Unlike traditional industrial robots, indoor service robots are usually required to possess high intelligence, such as the skills of flexible moving, precise spacial perceiving. And high intelligence is always accompanied by consuming complicated and expensive computation resources. One solution for indoor service robots is centralization of expensive computation resource so that it is possible to design a low cost client with a high-intelligence brain. However, as a fundamental intelligence function for mobile indoor robots, if a real-time visual Simultaneously Localization and Mapping (vSLAM) system is split into client and brain, it will be confronted with new challenges, such as the barrier of instant data sharing and performance degradation brought by network delay inbetween. To solve the problem, we focus on a framework and approach of cloud-based visual SLAM in this paper, and provide an efficient solution to offload the expensive computation and reduce the cost of robot clients. The integrated system is distributed in a 3-level Cloud with light-weight tracking, high precision dense mapping, and map sharing. Based on recent excellent algorithms, our system is able to run a real-time sparse tracking on the client, and a real-time dense mapping on the cloud server, which outputs an explicit 3D dense map. Only keyframes are sent to the local cloud center to reduce the network bandwidth requirement. Dense geometric pose estimation besides feature-based methods is computed to make the system resistant to feature-less indoor scenes. The camera poses associated with keyframes are optimized on the local computing cloud center, and are sent back to the client to decrease the trajectory drift. We evaluate the system on the Technical University of Munich (TUM) datasets, Imperial College London and National University of Ireland Maynooth (ICL-NUIM) datasets, and the real data captured by our robot in terms of visual odometry on the client side and dense maps generated on the server cloud. Qualitative and quantitative experiments show our cloud visual SLAM system is able to bear the network delay in Local Area Network (LAN), and it is an efficient vSLAM solution for indoor service robots with high intelligence from a centric brain.

INDEX TERMS Visual SLAM, cloud framework, service robots.

I. INTRODUCTION

Indoor service robot is a robot to assist human beings in indoor environments, such as cooking, cleaning, shopping, home maintenance in house, or doing guidance in exhibition room. The demands on indoor service robots in public or private spaces are rapidly increasing recently. Compared with traditional industrial robots, service robots are relatively required to have human-like intelligence. As a fundamental ability for mobile robots, autonomous navigation integrates a wide variety of processes from low-level actuator control to high-level strategic decision making. One critical issue is to develop techniques helping service robots to perceive the surrounding and localize itself in space using different kinds of sensors including LiDAR, Radar, sonar, inertial measurement unit (IMU) and camera, called SLAM technique. In the past decades vision-based SLAM has highly attracted lots of researchers, which takes videos as input, and computes the camera position and 3D maps as output. There are three main advantages for vision used in SLAM [1]: 1) camera is

The associate editor coordinating the review of this manuscript and approving it for publication was Seok-Bum Ko.

compactness and power saving which can be embedded in any robot; 2) vision allows to develop a variety of essential functionalities in robotics via vision perception techniques, such as obstacle detection, human tracking, visual servoing; 3) vision easily helps to implement relocalization and bounded errors on the position estimates of robot in low cost via loop detection and correction. Moreover, dense map is not only full of important geometric structure information but also offering potentials for detailed semantic scene understanding, which is usually accompanied by dense SLAM algorithms and is used to remote control, navigation, virtual reality and so on. However, dense mapping extremely consumes computing resource and storage resource. Most of recent map-centric dense visual SLAM algorithms even require to use expensive and big GPU to have real-time performance. For a practical indoor service robot setup, if we distribute vSLAM systems in cloud framework, the service robots could benefit from the use of Cloud brain to possess stronger computing and storage ability for producing precise indoor mapping and localizing but with a small and compact size.

Several groups have recently made preliminary trials, and made partial progress in the cloud-based visual SLAM. These work followed the tracking and mapping framework [16], which split tracking and mapping into two separate tasks. So it seems easy to distribute the whole systems in a 2-level cloud framework [2], [3]: the tracking thread runs on clients, and collaborative 3D mapping and map fusion is conducted by exploiting multiple clients in the cloud. The effect of network delay were not taken any account to the system in the works, since only keyframes were sent to the Cloud to build 3D map, so tracking and mapping are separated completely. The robot clients are only data collector for the server, so mapping could be done offline in their cases. However, when tracking and mapping are completely separated between threads, the tracking would suffer from drifting easily. As they discussed in the paper, the crucial problem of cloud SLAM is that procedure is very sensitive to the network delay and network bandwidth.

As all we know, more precise mapping requires more precise localizing, and more precise relocalizing requires more precise mapping in reverse, and SLAM algorithms are highly constrained by real-time requirement. Recently one state-of-the-arts of vSLAM algorithms on a single board computer was published, called ORB-SLAM, and it benefits significantly from the fact that the tracking and mapping threads are sharing a map model and optimized camera poses [11]. However, if this algorithms is transplanted into cloud framework, the most difficult thing is to handle network delay brought to the real-time system. It is even too hard to provide a comparable result. So the problem what we are trying to solve in this paper is: 1) when tracking is separated from mapping, how can we keep the performance of tracking; 2) even if the optimized camera pose can be sent back to the robot easily, but it will not be real time, so how could it affect the position of the robot.

We present a framework and approach which takes advantage of the powerful cloud computing and storage to not only reconstruct dense maps, but also estimate robust visual odometry for indoor service robots. Our system has real-time tracking with CPU on robot clients, and dense mapping on the local computing cloud with GPUs, and dense map sharing on the central computing center. The system is applied to a local area network (LAN). LAN refers to a network that interconnected computers within a limited area such as a neighborhood, school, laboratory, office buildings, or a house, which usually has a relative low network delay. To the best of our knowledge, this is the first work of real-time cloud visual SLAM which produces the comparable results of visual odometry on a single board computer. Beyond a cloud vSLAM, it generates high precision dense maps in the local cloud, which can be shared among private cloud. The main contributions of this paper are: a cloud framework of visual SLAM system with sparse tracking and dense mapping, which runs robust tracking on ARM cores of robot clients in real-time, and does accurate mapping on GPU cores of local computing servers. Camera pose optimization based on keyframes runs on local computing servers in real time, then optimized camera poses are sent back to correct the tracking poses of robots via local optimization. Experimental results demonstrate our system is able to tolerate the network delay of local area network(LAN), for instance, a domestic WiFi.

The remainder of this paper is organized as follows: in Section II, we provide a brief background vision-based SLAM technique in related applications. In Section III, we give the system overview for our cloud vSLAM. In Section IV, V, VI, we describe the details of the system with respect to the parts in the client side and the cloud side. The experimental results and extensive evaluations are reported on the TUM datasets, ICL-NUIM datasets and our real scenes in section VII. Finally, we give the conclusion in Section VIII.

II. RELATED WORK

Visual SLAM has been paid much attention in the computer vision and robotics community for the past few decades. In robotics field, visual odometry is the process of incrementally estimating the pose of vehicles from image streams captured by the onboard cameras, which is quite fundamentally related to visual SLAM. But visual SLAM does not only aim to the camera trajectory, but also the global consistent map. So far visual SLAM systems is evolved into complete and complicated systems, and usually consists of different techniques, including 3D geometric reconstruction, loop detection and closure, non-linear optimization, to improve the performance of systems and algorithms.

From the early beginning, monocular video was first used in visual SLAM system [8]. Davison and Murray tried sparse Harris features, and considered Extended Kalman Filtering to solve the visual SLAM problem. MonoSLAM was a real-time visual SLAM algorithm, which took videos from single cameras as input, and created online sparse but persistent

map of features in the probabilistic framework [15], [38]. Planar patch features [10] and line features [9], [46] were considered in visual SLAM approaches beside key point features. And as the development of sensors, stereo video [44] and RGBD video [45] are utilized to compensate for the weakness of monocular vision methods in different applications.

Klein and Murray [16] presented a seminal work in visual SLAM, which splits the system into two parallel tasks, tracking and mapping. This framework is widely applied to speed up real-time visual SLAM systems [11], [18], [19], [26], [45], and is extended to the use of computationally expensive optimization techniques. Grisetti *et al.* explained a graph SLAM approach in [17], in which involved to build a graph to constrain the connect camera poses from sensor measurement, and optimized in a nonlinear framework.

Not only the sparse features extraction was used in visual SLAM, but also the dense pixels were used in [5], [18], [19]. Newcombe *et al.* assumed a dense model of scenes, and proposed dense tracking and mapping with whole image alignment against that model [18]. The system was implemented in real time as well. A dense RGB-D SLAM algorithm by minimizing both the photometric and the depth error over every pixels was proposed by Kerl *et al.* [5]. And they selected keyframes and detected loop closure using entropy based similarity measurement, and optimized the pose graph in g2o framework. In the work of [20], Kerl *et al.* focused on the problem brought by rolling shutter RGB-D cameras to the dense SLAM. The continuous trajectory representation was used to compensate the rolling shutter effect, and showed superior quality in tracking and mapping. Engel *et al.* utilized dense tracking, semi-dense map estimation and map optimization for large-scale SLAM in a probabilistic method [19]. No matter sparse or dense SLAM, loop closing is demonstrated to improve significantly the final SLAM results, and is utilized by most of the recent visual SLAM work [5], [6], [11], [14], [47]. And deep learning technology is explored in visual SLAM [21], [22] in the most recent literatures.

Cloud computing is a technique that provides shared computer processing resources and data through Internet connection to computers and other devices on demand. The most important benefit of cloud computing is data and computing resources sharing, so using cloud computing is a trend in applications such as social robots [23]. Visual odometry [25], vision based 3D mapping [13], [26], [37], robot navigation [27], [36] and multi-robots collaboration [30], [31] are the typical applications for cloud robotics. Limosani *et al.* presented a system in the cloud robotics paradigm to help autonomous robots navigating in indoor environment. ARTags and QR codes were applied to mark rooms, corridors, entrances, and atriums for localization. The references [28], [29] provided different cloud computing frameworks for robotics. In this paper, our purpose is to develop a cloud visual SLAM system not only offloads the expensive computing and storage, but also keeps performance without any loss for indoor service robots.

III. SYSTEM OVERVIEW

In this section, we provide a system overview for our cloud visual SLAM. Our purpose is to present a cloud visual SLAM system, not only offloading the expensive computing and storage, but also keeping high performance as on stand-alone system even better for indoor service robots. We follow the tracking and mapping framework [16]. We implement the tracking with CPU in order to keep clients low cost, and the dense mapping on the local cloud computing center in LAN using the NVIDIA Compute Unified Device Architecture (CUDA) and Open Graphics Library (OpenGL) to obtain a high precise map. The CUDA and OpenGL API enables us to implement temporal filtering, point fusion on a GPU to reach real-time performance. For indoor service robots, the tracking part provides the camera simultaneous poses. While real-time tracking runs on the client, keyframes are selected and sent to the cloud server, once the corresponding camera poses are optimized, then the data will be sent back to the robot client to rectify trajectory drift of the robot in the local optimization. Fig. 1 shows the chart of our proposed system overview. Our system is a 3-level framework, including a robot client, a local computing cloud in LAN, and a private cloud on Wide Area Network (WAN). The cloud in our paper consists of the local computing server and the private cloud.

When frame is captured, we first extract ORB features, and estimate local camera pose initially after guided matching and optimize the camera pose in a local optimization. Keyframes are selected under some specific conditions, and sent to the local computing cloud with the corresponding robot state (if we have good feature matching and good estimation for camera poses, visual tracking succeeds, and denoted by 0, otherwise visual tracking fails, which means the robot is lost, and denoted by 1). When a keyframe is received by the local computing cloud, visual tracking and dense map fusion based on the keyframe is executed when the corresponding robot state is 0, otherwise visual relocalizing is launched. When relocalizing fails, a dense tracking based on depth images is utilized to estimate camera poses to enhance the robustness of the system. Loop detection and optimization is always carried out once a keyframe is received. When the dense map is built completely, it is first compressed and upload to a private cloud for storage and sharing.

IV. LIGHT-WEIGHT TRACKING ON THE CLIENT

In our robot system, we use RGB-D cameras to capture images. To build light-weight tracking in the client side, we use the Oriented FAST and Rotated BRIEF (ORB) features [12]. This is a binary descriptor based on FAST keypoint detection, which enables feature matching on low-power devices without GPU acceleration. We take the first 3D depth measurements $\mathbb{D}_1$ associated with 2D sparse features to initialize the 3D map when camera starts to move. The first camera pose is assumed at the origin of the coordinate. Once 2D ORB features are extracted from each color image, and the corresponding 3D points in the local map are projected by a predictive motion model, the most similar

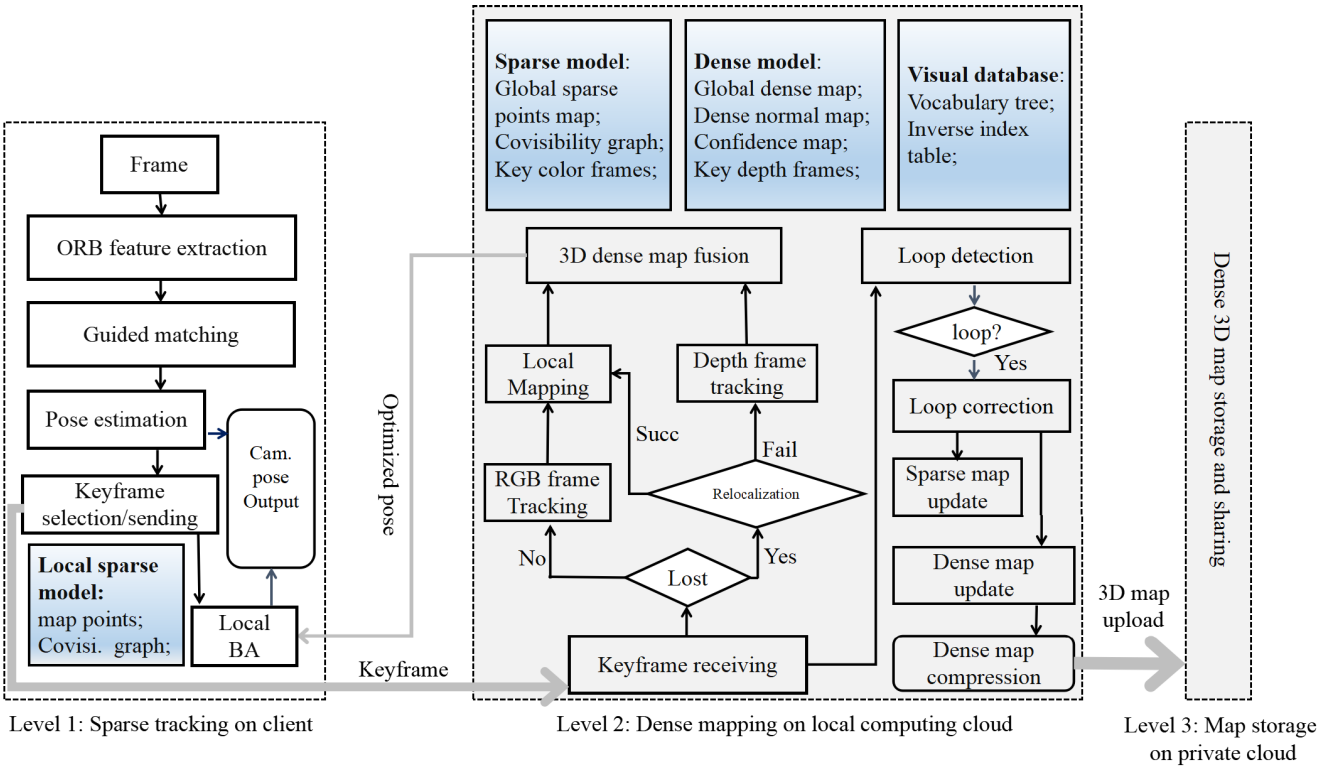


FIGURE 1. The overview of our proposed system.

features are searched in a local area in the current frame via guided matching. The pose estimation is computed by EPnP algorithm from detected correspondences [39], the quantity and quality of correspondences in fact indicates if we have a good pose estimation. The quantity of correspondences between the current frame and the last frame is low, that means we have a bad matching, then the state of the robot is thought of being lost. We maintain a local sparse model to improve the consecutive tracking estimation. Our sparse model consist of 3D local map points and a local covisibility graph, which is continuously updated while a new keyframe is creating and culling. If the optimized camera pose of the last keyframe is received from the local computing cloud and if it is in the covisibility graph, the connected camera poses and 3D map points will be optimized in local Bundle Adjustment (LBA). In the following subsection, we will describe the details of keyframes selection, local model update and LBA.

A. KEYFRAME SELECTION

The framerate of stream from RGB-D cameras is about 25 frames per second. These frames contain a number of redundant information, so a lot of visual SLAM algorithms are based on keyframes, which are selected in an appropriate way. In our system when a frame is selected as a keyframe, it will be sent to the local computing server, so the keyframe selection is a crucial step. If too many frames are selected, it would bring much pressure to network transmission. If too few frames are selected, we can not have an effective reconstruction for the whole system. So it is not an uniform time selection, and we need to balance keyframe selection and network transmission. Under the demand of limited network bandwidth, we select keyframes as many as possible by the following rules:

C_1: ID of current frame is δ_1 greater than ID of last keyframe;

C_2: The ratio between the number of inliers and the number of all ORB features in current frame is less than δ_2;

C_3: The difference of the camera pose vectors between current frame and its closest keyframe in Euclidean space is greater than δ_3;

C_4: Tracking fails.

Where $\delta_1, \delta_2, \delta_3$ are three thresholds set in advance. We take the following rule to select keyframe:

If $(C_1$ *and* $C_2)$ *or* C_3 *or* C_4, then selected.

In the above expression, we can see for two consecutive keyframes, they will not overlap much controlled by C_1 and C_3. Condition C_2 controls the matching quality of two consecutive keyframes. Once the rule is satisfied, the keyframe is used to build a local model on the robot side, at the same time it is sent to the local computing center for dense mapping.

B. LOCAL MODEL UPDATE

Our local model $\mathscr{L}$ includes 3D map points $\mathscr{L}_P$ and a local covisibility graph $\mathscr{L}_G$. The covisibility graph is a undirect

weighted graph built on the keyframes [11], in which each node represents a keyframe, and each edge between nodes represents if points can be observed in both nodes, weighted by the number of points. If the edge with a weight greater than a threshold β is regarded as a valid edge, and retained in the graph, otherwise deleted, to reduce complexity of the graph. Assume client robots have limited space and limited computation ability. It only allows to keep a local covisibility graph and local map points from a limited number of keyframes. Once a keyframe is determined, it will be added into the covisibility graph. And we also take some strategies to cull keyframes from the covisibility graph. Instead of taking the consecutive frames in a sliding window, we employ a flexible way to cull a keyframe in a dynamic window by the following steps:

1) when a new keyframe is determined, a counter $C[n]$ is set for the keyframe (initially set as 0), where n is the index of the keyframe in $\mathscr{L}_G$. New local map points from the nth keyframe are created by triangulating ORB features from the connected keyframes in the local covisibility graph [11]. Before doing LBA, let $C[i] = C[i] + 1, i = 1, \ldots, n$, we will give the details of LBA in the subsection IV-C.

2) If 3D points and keyframes optimized in LBA can be seen by K keyframes, assume $\mathbf{S}$ is the index set for the K keyframes, then after doing LBA, the count numbers associated with the K keyframes do $C[k] = C[k] - 1$, for all $k \in \mathbf{S}$.

3) When the count number $C[r]$ is greater than a threshold δ_4, then culling the rth keyframe from the covisibility graph, and also culling these map points can be seen uniquely in the rth keyframe.

In fact, the count numbers measure how important keyframes to the current frame. If the number is big, it indicates less important, since we do not optimize the keyframe in LBA for a while, when it is bigger than the threshold δ_4, then culling from the covisibility graph. This strategy is useful to keep the local map and the covisibility graph in a limited size, and it is also able to handle the loopy case easily.

C. LOCAL BUNDLE ADJUSTMENT

When the optimized camera pose $T^o_{t_1}$ corresponding to the keyframe T_{t_1} is received by the client, assume the current frame T_{t_c} is selected as the keyframe at the moment. Then keyframe T_{t_c} is added to the covisibility graph. Before doing LBA, if the keyframe T_{t_1} still exists in the covisibility graph (one case is that it may be culled out from the covisibility graph in subsection IV-B), all poses from the keyframe T_{t_1} to the current keyframe T_{t_c} in the covisibility graph multiply the transformation matrix $T^{-1}_{t_1} T^o_{t_1}$. Then LBA [40] helps to optimize camera poses of the current keyframe with its connections in covisibility graph, and 3D points in sparse map. This step enables the tracking thread to reduce the trajectory drift. Even if there are some optimized poses lost due to the network loss, the optimized camera poses will help since we maintain the local model.

V. DENSE MAPPING ON LOCAL COMPUTING SERVERS

Dense map is full of appearance and geometric information beyond sparse map, and it is meaningful for many applications, for example, robot path planning, indoor navigation. Once keyframes are received continuously from the robot clients, the dense map is incrementally built in the computing center cloud. In order to implement real-time dense mapping, we apply CPU-GPU hybrid programming.

We maintain a sparse model $\mathscr{S}$, a dense model $\mathscr{D}$ and a visual database $\mathscr{B}$ through the whole processing on the local computing cloud. The sparse model is mainly from feature-based estimation, includes a global sparse point map $\mathbb{S}_p$, a covisibility graph $\mathbb{S}_G$, and a color images $\mathbb{S}_c$. It is denoted as follows,

$$\mathscr{S} = \{\mathbb{S}_p, \mathbb{S}_G, \mathbb{S}_c\} \tag{1}$$

The sparse model will be updated continuously in the local mapping and the loop correction parts.

The dense model is used for dense mapping, which includes a global dense map $\mathbb{M}$, a dense normal map $\mathbb{N}$, a confidence map $\mathbb{C}$ and key depth images $\mathbb{D}_d$, and it is represented as,

$$\mathscr{D} = \{\mathbb{M}, \mathbb{N}, \mathbb{C}, \mathbb{D}_d\} \tag{2}$$

The depth images are accumulated into the dense map with the normal map and confidence map. The dense model will be updated incrementally in the 3D dense map fusion and dense map update parts.

The visual database is built incrementally for relocalization and loop closing, which records where the robot client has gone [32]. It includes a vocabulary tree and inverse index table for fast searching images according to words, where a ORB vocabulary dictionary is learnt ahead.

A. KEYFRAME TRACKING

Our keyframe is a RGB-D frame, we consider motion tracking on both RGB keyframe and depth keyframe in this paper. When a keyframe is received by the local computing cloud and the attached flag is detected as 0, features are first extracted from RGB frames, and matched with the previous three keyframes. Then the current camera pose is estimated by EPnP algorithm [39].

If the attached flag is detected as 1, that means the camera tracking on the robot client is lost. Further, if it can not find a reliable candidate from visual relocalization, that means visual relocalization fails, then depth tracking is employed to register the depth keyframe with the dense model. This happens when there are few visual features or images with low quality features. If we know the last camera pose, we may project the last depth frame from the global depth map. Assume the current depth frame is transformed into the global coordinate with T, we take the following function as the objective to minimize the point-to-plane error metric [48]:

$$E_{icp} = \sum_{\mathbf{u}} ||(T\mathbb{M}_i(\mathbf{u}) - \mathbb{M}^g_{i-1}(\mathbf{u})) \cdot \mathbb{N}^g_{i-1}(\mathbf{u})||^2, \tag{3}$$

where T is the transformation matrix to the global coordinate. $\mathbb{M}_i(\mathbf{u})$ denotes the depth at $\mathbf{u}$ of the current ith image, and $\mathbb{M}^g_{i-1}(\mathbf{u})$ denotes the depth at $\mathbf{u}$ of the $(i-1)$th projective image in the global coordinate. $\mathbb{N}^g_{i-1}(\mathbf{u})$ denotes the normal map of $\mathbb{M}^g_{i-1}(\mathbf{u})$. And once T is optimized over by the objective function, the camera pose associated with visual lost keyframe is estimated. Then the camera depth image can be registered with the dense map through the pose. And the pose would be sent to the client as well if it has a low error value.

B. LOCAL MAPPING

When the keyframe with a zero flag is received, a new node will be created in the covisibility graph for the keyframe. In the local mapping part, we optimize the current keyframe with its connect keyframes in the covisibility graph, and all 3D points observed in these keyframes by local BA. To keep the 3D points in the sparse map as accurate as possible, we take the similar strategies as the processing in [11] in term of map points culling, new point creation and keyframe culling. The optimized camera pose associated with the current keyframe would be sent back to the robot client to correct the localization of the robot.

C. LOOP DETECTION AND CORRECTION

Loop closing is always a critical step in SLAM, it is able to effectively suppress the trajectory drift with loop constraints. When a keyframe is received, it would be detected if it is a loop in parallel by DBoW2 [32]. If it has multiple loop candidates, it is determined as a loop stably, the loop correction is activated. Assume the keyframe is denoted by K_i, and the matching candidate from the database is denoted by K_c. The relative transformation T_{ic} between K_i and K_c is used to correct the current keyframe, all its neighbors in the covisibility graph and the 3D points seen by the keyframe and all its neighbors. When the loop frame is detected and added to the covisibility graph, the covisibility graph would be updated by adding more edges if the keyframes satisfy the condition of covisibility graph. In order to speed up the optimization, we build Essential graph as in [11]. All 3D points and the camera poses in Essential graph are optimized in global BA.

D. DENSE MAP FUSION

In our dense map fusion part, the depth image is incrementally fused into a dense map frame by frame in real-time performance. For the dense map fusion, we not only add more points in the dense map, but also process point ghosts. We set a confidence number c_k for each point in the global map, which is initially set as 0. When a point $\mathbf{u}$ merged with the point p_k in the global model, we increase the confidence number c_k. We first project the points of the global model M with the current camera pose estimated from subsection V-A, where the point index k is recorded, and associate the points in the current frame with the global map by nearest neighbor. In order to create a precise dense map, we also discard the points with large distance or large angle between the point and merged candidates, and remain the points with high confidence number. Then we take the point averaging method of the work to fuse the new depth image with the dense map incrementally with a strict outlier removal strategy [7]. If a 3D point observed across multiple depth frames, it would have a big confidence number, and the point fusion in the global map would be effected by the weighted sum of all these points. The main map fusion steps are as follows,

$$\begin{aligned} \mathbb{M}^p_k &= \frac{c_k \mathbb{M}^p_k + \alpha \mathbb{M}^c_k(\mathbf{u})}{c_k + \alpha}, \\ \mathbb{N}^p_k &= \frac{c_k \mathbb{N}^p_k + \alpha \mathbb{N}^c_k}{c_k + \alpha}, \\ c_k &= c_k + \alpha, \end{aligned} \tag{4}$$

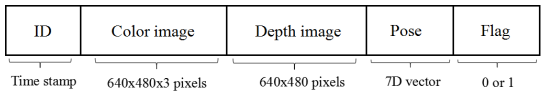
$\mathbb{M}^p_k$ denotes the kth point in the current depth projection from the dense map $\mathbb{M}$, and $\mathbb{M}^c(\mathbf{u})$ denotes the current depth at position $\mathbf{u}$, c_k is the confidence weight stored in the confidence map, and α is the update weight set as a gaussian function $e^{-\frac{(\gamma^2+d^2)}{2\delta^2}}$, in which γ is the normalized radial distance of the current depth measurement from the camera center, and d is a normalized effective distance factor. $\mathbb{N}$ denotes the normal map, which is used in the keyframe tracking as well. As the points in $\mathbb{M}^p$ and $\mathbb{N}^p$ in Equ.4 are updated, then the current depth image are fused into the dense model $\mathbb{M}^p$, and the dense model becomes larger gradually.

E. SPARSE AND DENSE MODEL UPDATE

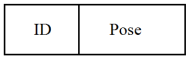
Since the loop correction significantly improves the camera pose estimation, so the camera poses in our sparse model are optimized with the global constraints while a loop is detected, then the sparse map is updated accordingly. When the optimization of camera poses reaches a stable state finally, the dense map is rebuilt according to the final optimized camera poses and the depth keyframes in the dense model. We use subsection V-D to recreate a more precise dense map with GPU in a single thread.

VI. DENSE 3D MAP COMPRESSION AND SHARING ON PRIVATE CLOUD

Once a 3D dense map reconstruction finishes on the local cloud, it is uploaded to the private cloud for storage and sharing. This holds two benefits: 1) reduce the storage consumption on computing servers; 2) easy to share the 3D dense map with other clients. In fact uploading and downloading of dense maps between local computing cloud and private cloud does not require the real-time performance. In order to reduce the pressure of network communication and storage space, we compress the 3D maps with the Octree point cloud compression algorithm [49], then the compressed dense map is uploaded and stored on the private cloud. The Octree compression algorithm is a 0-1 coding algorithm, which is based on an octree decomposition of space, and achieve high compression rate.



(a) The data structure from client to server



(b) The data structure from server to client

FIGURE 2. The structures defined for data transport between the client and the server. The network consumption of the data is mainly from image data uploading.

VII. EXPERIMENTAL RESULTS AND ANALYSIS

We evaluate our system on the indoor datasets, and also test our system in five different real indoor scenarios. We take the robot 'Hori' built by ourselves as the client, which has a laptop with Intel Core (i5-3210, 2.5GHz, 4G RAM, and no GPU). Our local computing cloud is a server with Intel Core (i7-4790, 4GHz, 16G RAM) and a stand-alone GPU (NVIDIA GTX970). The robot and the local cloud are connected via WiFi-5G network. The whole system is built on ROS. We keep the parameters the same in all experiments ($\delta_1 = 10, \delta_2 = 0.55, \delta_3 = 0.25, \delta_4 = 150, \beta = 50$). For our real scenarios, an ASUS Xtion RGB-D camera is equipped on Hori to capture RGB-D data and perceive its surrounding. Our cloud SLAM system runs online to reconstruct dense maps.

A. DATA TRANSMISSION BETWEEN THE ROBOT AND THE LOCAL CLOUD

We first introduce the structure for data transmission between the client and the local cloud. Time stamp is utilized as identification, then color image and depth image are attached. A 7D vector represents pre-estimate pose includes rotation in quaternion and translation. The last flag represents the state of the robot. Fig. 2(a) and (b) shows the data structures used in our system. As we can see, network bandwidth is consumed mainly by image data. Since we only send keyframes, so the network burden is reduced significantly as the work [2], [3].

B. VISUAL ODOMETRY EVALUATION ON THE TUM RGB-D DATASET

TUM dataset is a RGB-D SLAM benchmark, provided by Computer Vision Group of Technique University Munchen [33]. It consists of RGB-D data captured by Microsoft Kinect sensor (the first generation) and ground truth obtained by a high-accuracy motion-capture system with eight high-speed tracking cameras for evaluating visual odometry and visual SLAM system. We first test visual odometry of our system on the robot side.

ORB-SLAM2 is a real-time state-of-the-art system running on a single computer [11]. We compare our system with ORB-SLAM2 on 20 sequences of the TUM RGB-D benchmark to evaluate camera tracking. And we also report camera tracking on the robot without the optimized poses received from the computing cloud, because camera tracking without the optimized poses was employed in the existing cloud SLAM frameworks [2], [3]. The table 2 reports the root-mean-square error (RMSE) of trajectory recovery on the robot client. In all sequences, the tracking in our system provides comparable results with ORB-SLAM2. And ORB-SLAM2 and our system show better performance than the results without the local cloud optimization. Because we have keyframe optimization on the local computing cloud, and the drift of trajectories is controlled effectively.

TABLE 1. Tracking error comparison on the robot client in TUM RGB-D benchmark (Unit: m).

Datasets	ORB-SLAM2	Ours	Tracking without opt.
fr1_xyz	0.009468	**0.009220**	0.009922
fr1_desk2	**0.023101**	0.032055	0.037747
fr2_desk	**0.008880**	0.015321	0.016891
fr2_xyz	**0.003483**	0.004341	0.004583
fr3_office	**0.008744**	0.015397	0.020045
fr3_nostructure_texture_far	0.057304	**0.036255**	0.047785
fr3_nostructure_texture_near_withloop	0.018250	**0.015748**	0.020042
fr3_structure_texture_far	0.012007	**0.011511**	0.012055
fr3_structure_texture_near	0.010786	**0.010412**	0.011562
fr1_plant	**0.013756**	0.025524	0.028742
fr2_dishes	**0.019894**	0.035550	0.042507
fr2_flowerbouquet_brownbackground	0.852061	**0.846772**	0.895149
fr2_metallic_sphere2	0.064784	**0.063231**	0.113972
fr3_large_cabinet	**0.039987**	0.063808	0.081678
fr2_large_no_loop	0.208254	**0.199771**	0.299903
fr2_desk_with_person	**0.007654**	0.007974	0.008578
fr3_sitting_halfsphere	0.030172	**0.028015**	0.053120
fr3_sitting_static	0.0079925	**0.007234**	0.008001
fr3_sitting_xyz	0.009150	**0.009144**	0.009214
fr3_walking_static	**0.053427**	0.112244	0.153633

We also report the camera localization for keyframes of our system running on the local computing cloud server in Table 2. From the table, we can see that our system achieves comparable performance with ORB-SLAM2. Although we have more keyframes generated, this part in our system runs on the local computing cloud, so the processing speed is not a problem at all. Moreover, our algorithm requires less memory and less computing consumption on the robot, which can be implemented on embedded system. We will evaluate the memory requirement in the subsection VII-E.

C. MAP EVALUATION ON ICL-NUIM DATASET

We evaluate 3D map reconstruction generated from our system with DVO SLAM system [5], RGB-D SLAM system [33] and ElasticFusion [6] on ICL-NUIM dataset. The ICL-NUIM dataset is a resent released dataset by Handa *et al.* of Imperial College London and National University of Ireland [42]. This dataset does not only provide ground truth poses for benchmarking visual odometry but also for the 3D surface reconstruction to evaluate surface reconstruction accuracy. Because only the four sequences of Living room have 3D surface ground truth, and the rest four sequences in office room scene do not have 3D model with it. So all four sequences in the living room scene are used to evaluate mapping in

TABLE 2. Camera pose estimation comparison for keyframes in TUM RGB-D benchmark (Unit: m).

Dataset	Ours		ORB-SLAM2	
	RMSE	No. KF	RMSE	No. KF
fr1_xyz	**0.009157**	53	0.010998	42
fr1_desk2	0.028757	114	**0.025675**	127
fr2_desk	**0.008173**	200	0.008531	176
fr2_xyz	**0.003362**	112	0.0034	40
fr3_office	**0.008831**	268	0.008891	240
fr3_nostructure_texture_far	**0.027314**	38	0.082681	39
fr3_nostructure_texture_near_withloop	0.022369	41	**0.017178**	43
fr3_structure_texture_far	**0.011344**	38	0.015132	28
fr3_structure_texture_near	0.012205	55	**0.011577**	52
fr1_plant	0.019245	181	**0.01257**	208
fr2_dishes	0.034091	287	**0.020613**	227
fr2_flowerbouquet_brownbackground	0.817227	655	**0.765155**	779
fr2_metallic_sphere2	**0.043312**	234	0.065946	238
fr3_large_cabinet	0.044666	138	**0.040176**	132
fr2_large_no_loop	0.280561	570	**0.177935**	584
fr2_desk_with_person	**0.005620**	206	0.007104	147
fr3_sitting_halfsphere	**0.022762**	117	0.023355	91
fr3_sitting_static	0.014036	11	**0.012857**	12
fr3_sitting_xyz	**0.010721**	42	0.012639	34
fr3_walking_static	0.140272	42	**0.066336**	42

TABLE 3. Comparison of 3D reconstruction accuracy on the evaluated synthetic datasets of [42].

Systems	lr kt0	lr kt1	lr kt2	lr kt3
DVO SLAM	0.032m	0.061m	0.119m	0.053m
RGB-D SLAM	0.044m	0.032m	0.031m	0.167m
ElasticFusion	0.007m	0.007m	0.008m	0.028m
Our system	**0.006m**	**0.005m**	**0.007m**	**0.007m**

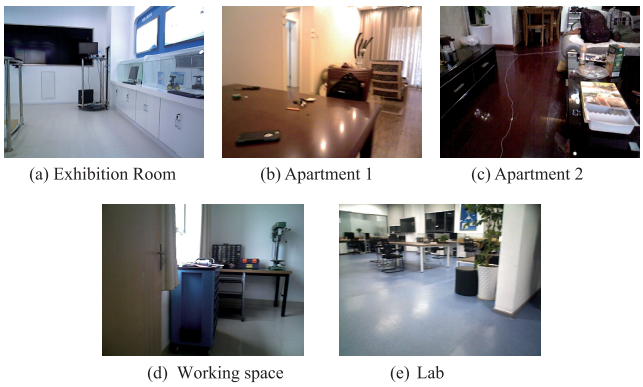
our system. Table 3 summarizes 3D map reconstruction result. The numbers in the table are the mean distances from each point to the nearest in the ground truth model. It is seen that our 3D reconstruction results are superior to all other systems in the four sequences. The 'lr kt3' triggers a global loop closure in our system, so the results from our system is significantly improved compared with other methods. Note that the number of map points from our system might be less than ElasticFusion due to the fact that the dense mapping running on the local cloud server not on the sequential frames but on keyframes.

D. PERFORMANCE ON THE REAL DATA

The five real scenes data captured by our robot includes Exhibition room, Apartment 1, Apartment 2, Workspace, and Lab. The details of the datasets are reported in Table 4. The first row shows the duration of the sequences, the second row shows the total number of frames. The last row reports the number of keyframes selected and sent to the local computing cloud by our system. Fig. 3 (a)(b)(c)(d)(e) shows some example images from the real data in the view of the robot. Since we do not have ground truth, we only qualitatively

TABLE 4. Details of our real sequences.

Dataset	Exhi. Room	Apar. 1	Apar. 2	Workspace	Lab
Duration (s)	212	296	523	220	426
No. Frames	5909	8779	15545	6543	9479
No. KF	451	496	738	475	540
KF Ratio	7.63%	5.65%	4.78%	7.26%	5.70%



(a) Exhibition Room (b) Apartment 1 (c) Apartment 2

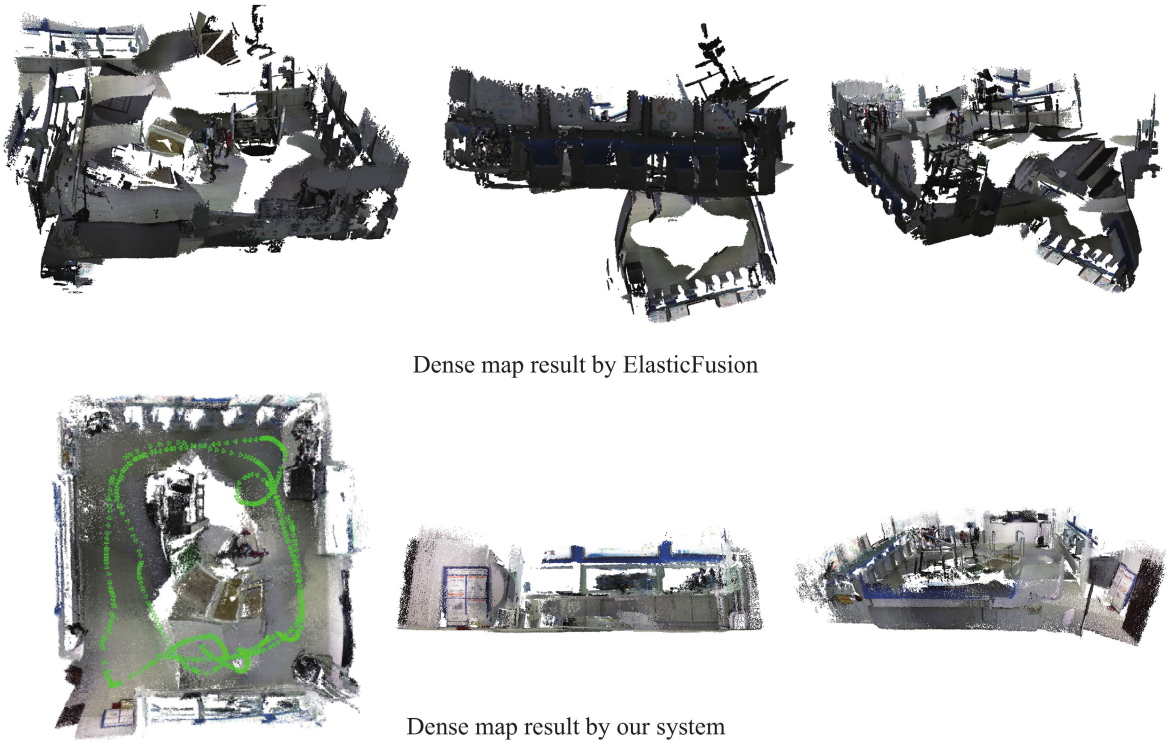
(d) Working space (e) Lab

FIGURE 3. Example images of five real scenes.

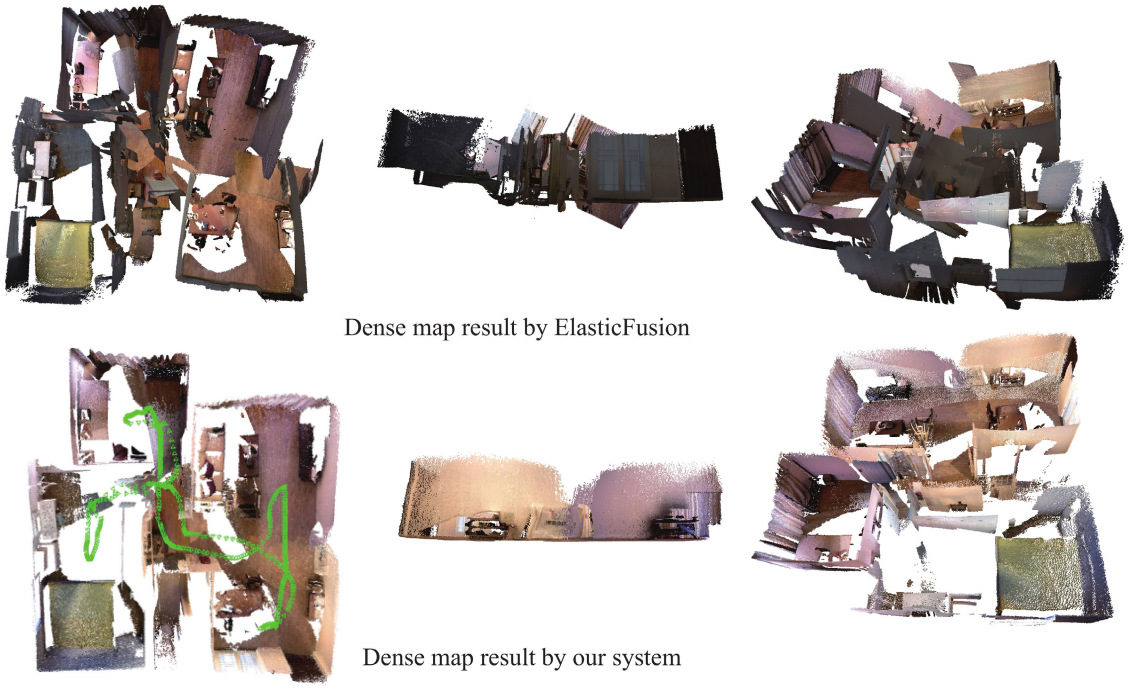
compare the dense mapping results with ElasticFusion [6]. Fig. 4, Fig. 5 and Fig. 6 shows the dense maps reconstructed by ElasticFusion and our proposed system in three different views. The first row of Fig. 4(a)(b), Fig. 5(a)(b) and Fig. 6(a) shows the results of ElasticFusion in three different views. The second row in each are produced by our systems in top view, horizontal side view and tilted view, respectively. From the details of the maps we can see, our method produces more accurate dense maps than ElasticFusion, because we have flatter floors, and more square rooms for all of five sequences. We print robot trajectories produced by our method on the dense maps in green as well.

E. MEMORY EVALUATION ON THE ROBOT

One benefit of our system is that we only require a very low memory load in the robot client since we remove the computing and memory consumption to the local computing cloud. In the subsection, we analyze the memory load on the client side to demonstrate the advantage of our system. We compare our memory consumption with ORB-SLAM2 algorithm on the longest sequence of the real data Apartment 2, which has 15545 frames totally. Fig. 7(a) shows the memory load comparison result. The axis x denotes the frame ID, the axis y denotes the number of points. The red line is from ORB-SLAM2, and the green line is generated from our system. As we can see, the memory load goes up as the frames processed in ORB-SLAM2 algorithm, but the memory load always keeps in a limited size in our system since we only maintain a local limited model with local constraints. But our system does not suffer from the performance degeneration, that is because the optimized camera poses from the local cloud server exert the effect on the trajectories on the robot client.

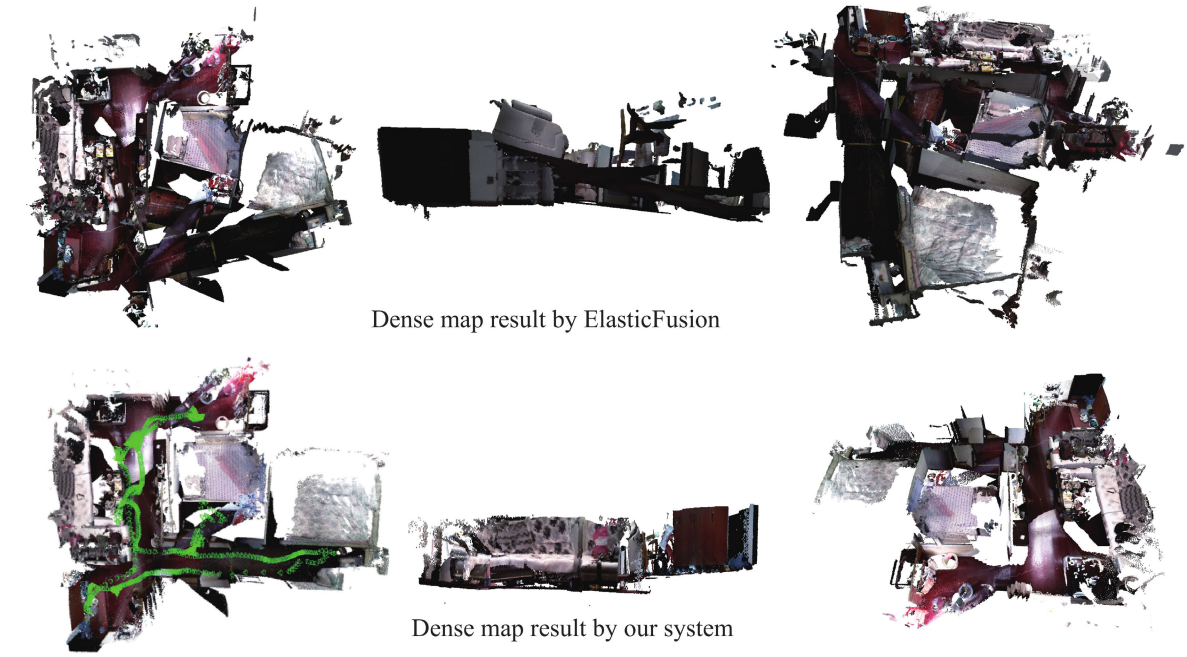


(a) Comparison results in three diffent views on the reconstruciton of Exhibition Room

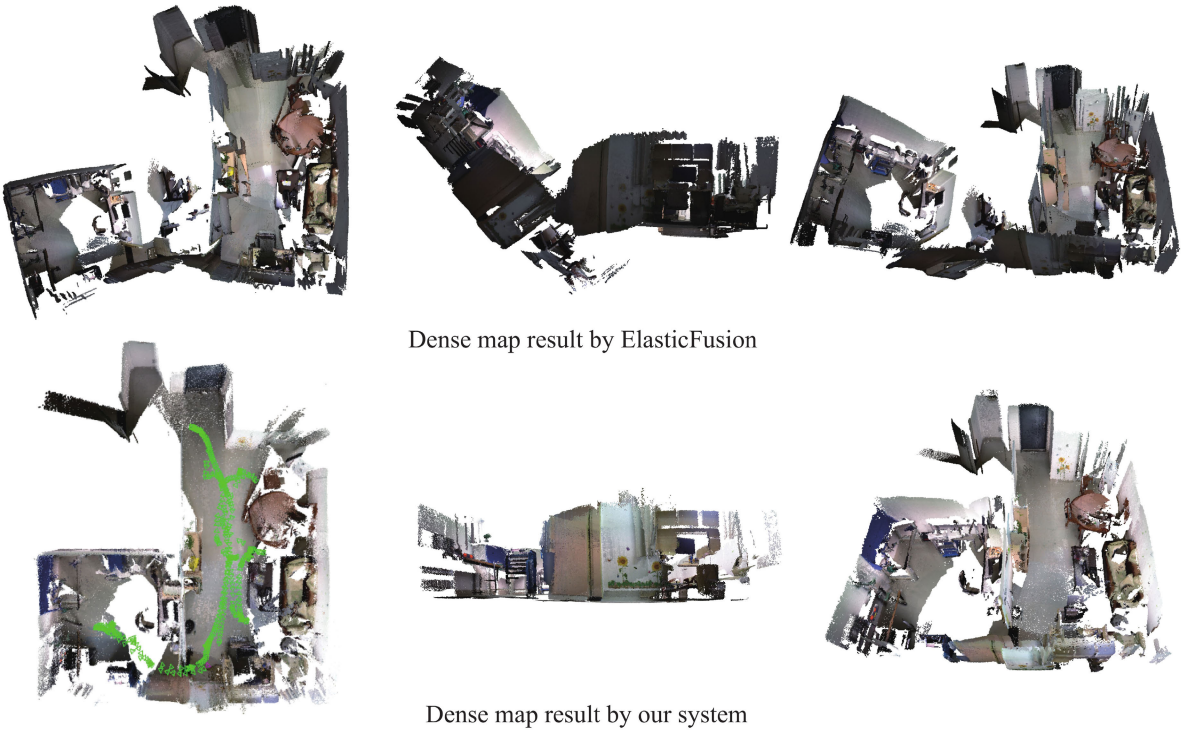


(b) Comparison results in three different view on the reconstruction of Apartment 1

FIGURE 4. 3D dense map generated by ElasticFusion and our proposed system on Exhibition Room and Apartment 1 in three different views. (a) shows the comparison results on Exhibition Room, (b) shows the comparison results on Apartment 1. The first row of (a)(b) show the results of ElasticFusion in three different views. The second row of (a)(b) are produced by our systems in top view, horizontal side view and tilted view, respectively. The green trajectories in the dense map are the camera poses generated by our system.

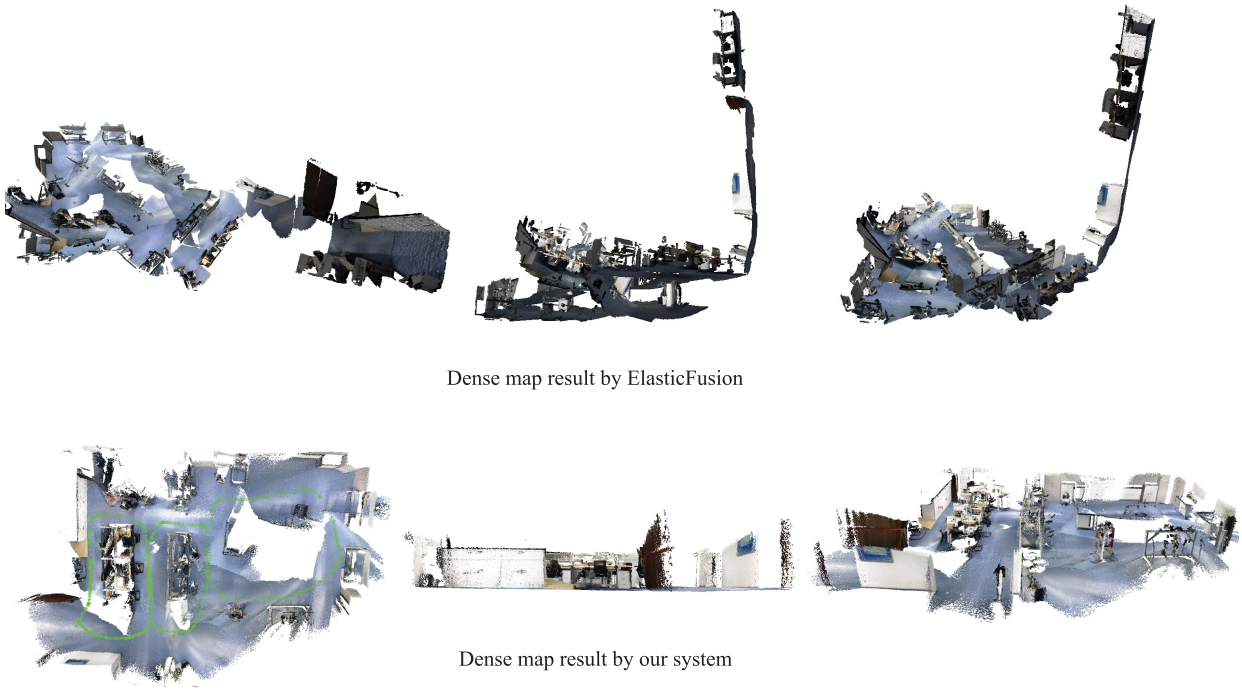


(a) Comprison result in three different views on the reconstruction of Apartment 2



(b) Comprison result in three differnt views on the reconstruction of Workspace

FIGURE 5. 3D dense map generated by ElasticFusion and our proposed system on Apartment 2 and Workspace in three different views. (a) shows the comparison results on Apartment 2, (b) shows the comparison results on Workspace. The first row of (a)(b) show the results of ElasticFusion in three different views. The second row of (a)(b) are produced by our systems in top view, horizontal side view and tilted view, respectively. The green trajectories in the dense map are the camera poses generated by our system.



(a) Comparison results in three diffent views on the reconstruction of Lab

FIGURE 6. 3D dense map generated by ElasticFusion and our proposed system on Lab in three different views. The first row show the results of ElasticFusion in three different views. The second row are produced by our systems in top view, horizontal side view and tilted view, respectively. The green trajectory in the dense map is the camera poses generated by our system.

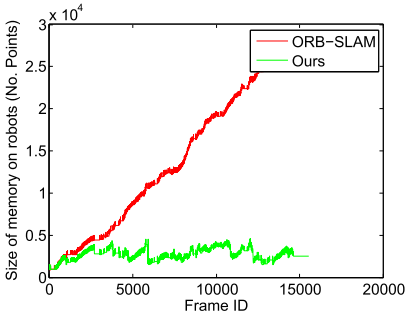


FIGURE 7. Memory load comparison for sequence Apar. 2 between ORB-SLAM2 and ours.

F. NETWORK DELAY ANALYSIS BETWEEN THE ROBOT AND THE LOCAL CLOUD

One of the big problems for the cloud visual SLAM is to treat network delay between the robot and the local computing cloud server. In this subsection, we analyze the network delay of our system in detail for the application of indoor service robot. In our experimental setup, the robot client is connected with a 5G WiFi router, and the router is connected with the local computing cloud server with a cable, which is a very common LAN. We observe that the most network delay comes from the robot and the router via wireless transmission, and there only exists very few network delay between the router and the local cloud server. Our system has a real-time tracking on the robot (around 25 fps), and optimized camera poses associate with keyframes from the cloud are used to correct tracking trajectory through the optimization in local BA. As long as the optimized camera pose is in the local sparse model, it exerts an effect on the tracking. The ratio between the keyframes and the frames in five real data is reported in the last row of Table 4, which shows it is nearly 2 keyframes selected per second. In most of the time, assume we maintain about 100 keyframes in the local model on the robot, which means the optimized camera pose could be the keyframe sent about $50s-$ ago. Of course, the closer to the current keyframe in the local sparse model is, the bigger effect it has.

In order to demonstrate how our system is able to bear the network delay of LAN, we do the experiments as follows. We measure the keyframe processing time on local computing cloud denoted by T_2 and the duration denoted by T_1 between the moment of a keyframe being sent and the moment of an optimized pose received on the robot client. So $T_1 - T_2$ is the total data transmission time between the robot and the local server, includes the time of a keyframe uploading and the time of the pose downloading. Fig.8 shows the measurement results in the five real dataset. The biggest of keyframe processing time T_1 on local computing cloud is below 25 ms with GPU, and the most of data transmission time between the robot and the local cloud is about 200 ms, which is thought of as an approximated estimation of the network delay. As we can see, there are few keyframes with a very high network delay in most of the data. The network delay on Exhibition Room is much lower than the rest of the data, this is because the Exhibition Room is a single room

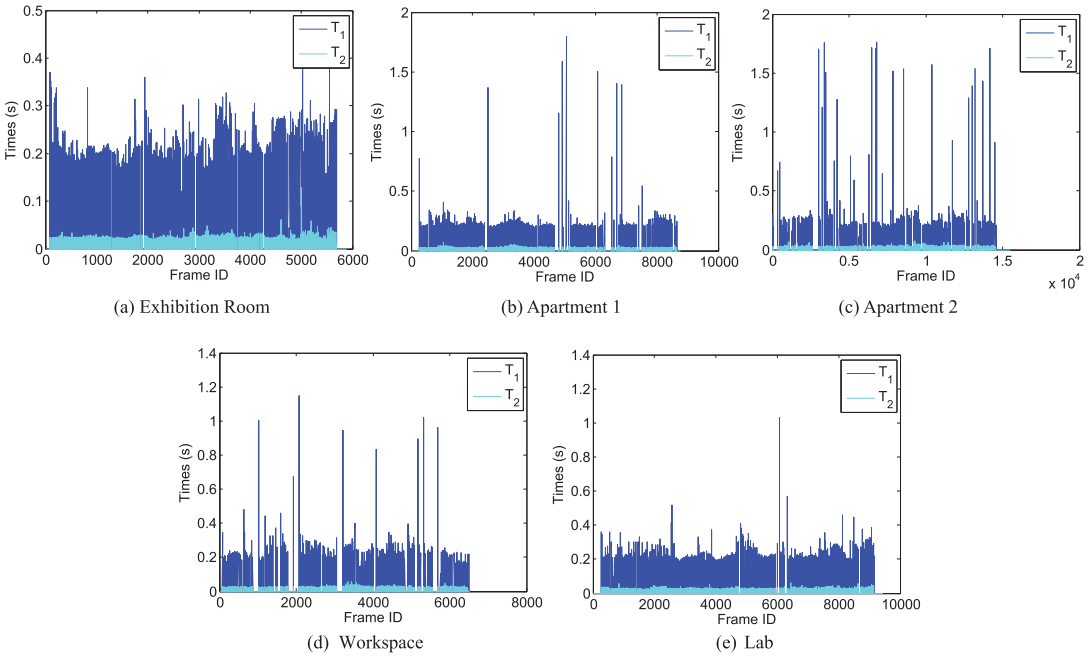


FIGURE 8. Network delay measurement and analysis. T_1 denotes the duration between the moment of a keyframe being sent and the moment of a optimized pose returning on the robot client. T_2 denotes the keyframe processing time on the local computing cloud. $T_1 - T_2$ can be thought of as an approximation for network delay.

TABLE 5. Map compression comparison.

Dataset	Original map		Compressed map			
	No. map points	File size	Resolution	Compression ratio	File size	Compression time (s)
Exhi. Room	970030	43.7 M	1 mm^3	12.58%	5.5 M	0.46
			5 mm^3	6.18%	2.7 M	0.50
			1 cm^3	3.67%	1.6 M	0.55
Apar. 1	1687653	76.9 M	1 mm^3	11.83%	9.1 M	0.8
			5 mm^3	5.20%	4.0 M	0.77
			1 cm^3	2.60%	2.0 M	0.63
Apar. 2	1743553	75.7 M	1 mm^3	11.49%	8.7 M	0.69
			5 mm^3	4.76%	3.6 M	0.67
			1 cm^3	1.85%	1.4 M	0.54
Workspace	1225263	56.1 M	1 mm^3	11.94%	6.7 M	0.56
			5 mm^3	5.35%	3.0 M	0.55
			1 cm^3	2.85%	1.6 M	0.47
Lab	1807656	81.1 M	1 mm^3	12.45%	10.1 M	1.16
			5 mm^3	5.80%	4.7 M	0.97
			1 cm^3	3.45%	2.8 M	0.88

with 8×9 m^2. Apartment 1, Apartment 2 and Workspace have multiple rooms, so WiFi signals could be not good as the in Exhibition Room. Lab is a much bigger open space but it is not a standard rectangle, so WiFi signal in part of the space could be blocked somehow. The network delay in Lab is bigger than Exhibition Room, but shorter than the scenarios with multiple rooms. However, even the network delay is up to 1.8s, our system is still able to bear and run successfully.

G. MAP COMPRESSION AND TRANSMISSION BETWEEN THE LOCAL CLOUD AND THE PRIVATE CLOUD

We use Octree point cloud compression algorithm before data transmission. We set three different compression resolution parameters for the algorithm — 1 cm^3 resolution, 5 mm^3 resolution, and 1 mm^3 resolution. 1 mm^3 is the highest resolution, it produces the highest compression ratio with the highest reconstruction precision. Table 5 shows the map compression comparison results. Even the number of points is up to 1 million, it only takes less than 1s of compression time. And through the compression processing, the map size is reduced significantly compared with the original map size with limited reconstruction error. The compressed map is used to upload to the private could and store.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we present an online cloud visual SLAM system which is suitable for LAN environment. Our system is different from ORB-SLAM2 in the following aspects:

1) ORB-SLAM2 is a real-time system on single PC, while our system, considering the network delay, is a real-time system on a network framework. It distributes the sparse tracking and dense mapping separately in the robot client and the local computing cloud. 2) ORB-SLAM2 does not output dense map explicitly, but our system does. 3) We have different keyframes selection strategies. More keyframes are sent to the local cloud, more accurate map is built, but requires more bandwidth for the real-time performance. Our keyframe selection is denser, and we balance the bandwidth volume and 3D map reconstruction accuracy. 4) In the tracking of the robot client, we maintain a local model for local optimization with the updates from cloud side to reduce drift of tracking part, and provide a dynamic selection strategy for the local model. 5) In the dense mapping, we apply not just the feature-based reconstruction, but also a depth registration to enhance keyframe tracking.

Moreover, the map centric method — ElasticFusion can build dense map with the consecutive depth frames, but we build dense map only up to depth keyframes, and the accuracy of our dense map is higher than ElasticFusion from the quantitative evaluation.

With a robust visual odometry running on the robot, an accurate dense map is generated on the local computing cloud, our system can be used to reduce the cost of the robot client potentially. And it is easy to share the dense map on the private cloud with different clients. As all we know, our system is resistant to the network delay of LAN, so it provides an effective vSLAM solution for indoor service robots. Our future work is to extend our cloud SLAM system to outdoor environments, and make the system applicable to a larger scale area, which can be used in such as intelligent vehicles.

ACKNOWLEDGMENT

(Yali Zheng and Shinan Chen contributed equally to this work.) Shinan Chen was with the School of Automation Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China.

REFERENCES

[1] T. Lemaire, C. Berger, I.-K. Jung, and S. Lacroix, "Vision-based SLAM: Stereo and monocular approaches," *Int. J. Comput. Vis.*, vol. 74, no. 3, pp. 343–364, Jul. 2007.

[2] L. Riazuelo, J. Civera, and J. Montiel, "C2TAM: A cloud framework for cooperative tracking and mapping," *Robot. Auto. Syst.*, vol. 62, no. 4, pp. 401–413, Apr. 2014.

[3] G. Mohanarajah, V. Usenko, M. Singh, R. D'andrea, and M. Waibel, "Cloud-based collaborative 3D mapping in real-time with low-cost robots," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 2, pp. 423–431, Apr. 2015.

[4] H. Bistry and J. Zhang, "A cloud computing approach to complex robot vision tasks using smart camera systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2010.

[5] C. Kerl, J. Sturm, and D. Cremers, "Dense visual SLAM for RGB-D cameras," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Nov. 2013.

[6] T. Whelan, S. Leutenegger, R. Salas Moreno, B. Glocker, and A. Davison, "ElasticFusion: Dense SLAM without a pose graph," in *Proc. Robot., Sci. Syst. XI*, Jan. 2016.

[7] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb, "Real-time 3D reconstruction in dynamic scenes using point-based fusion," in *Proc. Int. Conf. 3D Vis.*, Jun. 2013.

[8] A. Davison and D. Murray, "Simultaneous localization and map-building using active vision," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 865–880, Jul. 2002.

[9] P. Smith, I. Reid, and A. J. Davison, "Real-time monocular SLAM with straight lines," in *Proc. Brit. Mach. Vis. Conf. (BMVC)*, 2006.

[10] N. Molton, A. Davison, and I. Reid, "Locally planar patch features for real-time structure from motion," in *Proc. Brit. Mach. Vis. Conf. (BMVC)*, 2004.

[11] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: A versatile and accurate monocular SLAM system," *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1147–1163, Oct. 2015.

[12] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *Proc. Int. Conf. Comput. Vis. (ICCV)*, Nov. 2011.

[13] D. Zou and P. Tan, "CoSLAM: Collaborative visual SLAM in dynamic environments," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 2, pp. 354–366, Feb. 2013.

[14] C. Mei, G. Sibley, and P. Newman, "Closing loops without places," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2010.

[15] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "MonoSLAM: Real-time single camera SLAM," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 6, pp. 1052–1067, Jun. 2007.

[16] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *Proc. IEEE ACM Int. Symp. Mixed Augmented Reality*, 2007.

[17] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based SLAM," *IEEE Intell. Transp. Syst. Mag.*, vol. 2, no. 4, pp. 31–43, 2010.

[18] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "DTAM: Dense tracking and mapping in real-time," in *Proc. Int. Conf. Comput. Vis.*, Nov. 2011.

[19] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2014, pp. 834–849.

[20] C. Kerl, J. Stuckler, and D. Cremers, "Dense continuous-time tracking and mapping with rolling shutter RGB-D cameras," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015.

[21] X. Qi, S. Yang, and Y. Yan, "Deep learning based semantic labelling of 3D point cloud in visual SLAM," in *Proc. IOP Conf. Ser., Mater. Sci. Eng.*, vol. 428, Oct. 2018, Art. no. 012023.

[22] S. Milz, G. Arbeiter, C. Witt, B. Abdallah, and S. Yogamani, "Visual SLAM for automated driving: Exploring the applications of deep learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2018.

[23] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 2, pp. 398–409, Apr. 2015.

[24] J. Zhang and S. Singh, "Visual-lidar odometry and mapping: Low-drift, robust, and fast," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2015.

[25] M. Wu, F. Huang, L. Wang, and J. Sun, "Cooperative multi-robot monocular-SLAM using salient landmarks," in *Proc. Int. Asia Conf. Inform. Control, Autom. Robot.*, Feb. 2009.

[26] P. Benavidez, M. Muppidi, P. Rad, J. J. Prevost, M. Jamshidi, and L. Brown, "Cloud-based realtime robotic Visual SLAM," in *Proc. Annu. IEEE Syst. Conf. (SysCon)*, Apr. 2015.

[27] R. Limosani, A. Manzi, L. Fiorini, F. Cavallo, and P. Dario, "Enabling global robot navigation based on a cloud robotics approach," *Int. J. Soc. Robot.*, vol. 8, no. 3, pp. 371–380, Jun. 2016.

[28] D. Hunziker, M. Gajamohan, M. Waibel, and R. D'andrea, "Rapyuta: The RoboEarth cloud engine," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2013.

[29] R. Arumugam, V. R. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong, A. S. Kumar, K. D. Meng, and G. W. Kit, "DAvinCi: A cloud computing framework for service robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2010.

[30] P. Zhang, H. Wang, B. Ding, and S. Shang, "Cloud-based framework for scalable and real-time multi-robot SLAM," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jul. 2018.

[31] H. Yu, H. Li, and Z. Yang, "Collaborative visual SLAM framework for a multi-UAVs system based on mutually loop closing," in *Proc. Int. Conf. Wireless Satell. Syst.*, 2019, pp. 653–664.

[32] D. Galvez-López and J. D. Tardos, "Bags of binary words for fast place recognition in image sequences," *IEEE Trans. Robot.*, vol. 28, no. 5, pp. 1188–1197, Oct. 2012.
[33] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2012.
[34] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. Mcdonald, "Robust real-time visual odometry for dense RGB-D mapping," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2013.
[35] A. Wendel, M. Maurer, G. Graber, T. Pock, and H. Bischof, "Dense reconstruction on-the-fly," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2012.
[36] A. Favenza, C. Rossi, M. Pasin, and F. Dominici, "A cloud-based approach to GNSS augmentation for navigation services," in *Proc. IEEE/ACM 7th Int. Conf. Utility Cloud Comput.*, Dec. 2014.
[37] D. Portugal, B. D. Gouveia, and L. Marques, "A distributed and multithreaded SLAM architecture for robotic clusters and wireless sensor networks," in *Cooperative Robots and Sensor Networks*. 2015, pp. 121–141.
[38] A. J. Davison, "Real-time simultaneous localisation and mapping with a single camera," in *Proc. 9th IEEE Int. Conf. Comput. Vis. (ICCV)*, 2003.
[39] V. Lepetit, F. Moreno-Noguer, and P. Fua, "EPnP: An accurate O(n) solution to the PnP problem," *Int. J. Comput. Vis.*, vol. 81, no. 2, pp. 155–166, Feb. 2009.
[40] M. I. A. Lourakis and A. A. Argyros, "SBA: A software package for generic sparse bundle adjustment," *ACM Trans. Math. Softw.*, vol. 36, no. 1, pp. 1–30, Mar. 2009.
[41] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G^2o: A general framework for graph optimization," in *Proc. IEEE Int. Conf. Robot. Automat.*, May 2011.
[42] A. Handa, T. Whelan, J. Mcdonald, and A. J. Davison, "A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2014.
[43] R. Schnabel and R. Klein, "Octree-based point cloud compression," in *Proc. Eurograph. Symp. Point-Based Graph.*, 2006.
[44] P. Liu, L. Heng, T. Sattler, A. Geiger, and M. Pollefeys, "Direct visual odometry for a fisheye-stereo camera," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017.
[45] A. Concha and J. Civera, "RGBDTAM: A cost-effective and accurate RGB-D tracking and mapping system," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017.
[46] X. Zuo, X. Xie, Y. Liu, and G. Huang, "Robust visual SLAM with point and line features," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017.
[47] L. Han, G. Zhou, L. Xu, and L. Fang, "Beyond SIFT using binary features in loop closure detection," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017.
[48] S. Izadi, A. Davison, A. Fitzgibbon, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, and D. Freeman, "KinectFusion: Real-time 3D reconstruction and interaction using a moving depth camera," in *Proc. 24th Annu. ACM Symp. User Interface Softw. Technol. (UIST)*, 2011.
[49] R. Schnabel and R. Klein, "Octree-based point-cloud compression," in *Proc. 3rd Eurograph. Symp. Point-Based Graph.*, 2006, pp. 111–121.

YALI ZHENG (Member, IEEE) was born in 1980. She received the Ph.D. degree from Chongqing University, China, in 2012. She joined the School of Automation Engineering, University of Electronic Science and Technology of China, in 2013. She is currently an Associate Professor and leads the Vision Measuring and Learning Laboratory, UESTC. Her research interests include computer vision, machine learning, and pattern recognition, especially on multiple-view geometry theory, graph matching, deep learning, and their applications in industry fields.

SHINAN CHEN was born in Hainan, China, in 1995. He received the B.S. and M.S. degrees in automation engineering from the University of Electronic Science and Technology of China, Chengdu, China, in 2014 and 2017, respectively. Then, he joined Netease Company, China, as a Software Algorithm Engineer. He is interested in computer vision, especially in visual SLAM.

HONG CHENG (Senior Member, IEEE) received the Ph.D. degree in pattern recognition and intelligent systems from Xian Jiaotong University, in 2003. He was a Visiting Scholar with the School of Computer Science, Carnegie Mellon University, USA, from 2006 to 2009. He has been an Executive Director of the Center for Robotics, since 2014, has been with UESTC, since 2010, and has been an Associate Professor with Xian Jiaotong University, since 2005. He is currently a Full Professor with the School of Automation and Engineering, University of Electronic Science and Technology of China (UESTC). He has over 100 academic publications, including two books: *Digital Signal Processing* (Tsinghua University Press, 2007) and *Autonomous Intelligent Vehicles: Theory, Algorithms and Implementation* (Springer, 2011). His current research interest includes machine learning in human–robot hybrid systems. He served/is serving as a General Chair of VALSE 2015, a Program Chair of CCPR 2016, and a General Chair for CCSR 2016.