Paper:

ORB-SHOT SLAM: Trajectory Correction by 3D Loop Closing Based on Bag-of-Visual-Words (BoVW) Model for RGB-D Visual SLAM

Zheng Chai and Takafumi Matsumaru

Graduate School of Information, Production and Systems, Waseda University
2-7 Hibikino, Wakamatsu-ku, Kitakyushu 808-0135, Japan
E-mail: icecc_sunny@163.com, matsumaru@waseda.jp
[Received August 1, 2016; accepted November 14, 2016]

This paper proposes the ORB-SHOT SLAM or OS-SLAM, which is a novel method of 3D loop closing for trajectory correction of RGB-D visual SLAM. We obtain point clouds from RGB-D sensors such as Kinect or Xtion, and we use 3D SHOT descriptors to describe the ORB corners. Then, we train an offline 3D vocabulary that contains more than 600,000 words by using two million 3D descriptors based on a large number of images from a public dataset provided by TUM. We convert new images to bag-of-visual-words (BoVW) vectors and push these vectors into an incremental database. We query the database for new images to detect the corresponding 3D loop candidates, and compute similarity scores between the new image and each corresponding 3D loop candidate. After detecting 2D loop closures using ORB-SLAM2 system, we accept those loop closures that are also included in the 3D loop candidates, and we assign them corresponding weights according to the scores stored previously. In the final graph-based optimization, we create edges with different weights for loop closures and correct the trajectory by solving a nonlinear least-squares optimization problem. We compare our results with several state-of-the-art systems such as ORB-SLAM2 and RGB-D SLAM by using the TUM public RGB-D dataset. We find that accurate loop closures and suitable weights reduce the error on trajectory estimation more effectively than other systems. The performance of ORB-SHOT SLAM is demonstrated by 3D reconstruction application.

Keywords: trajectory correction, loop closing, bag-of-visual-words (BoVW), 3D vocabulary, RGB-D SLAM

1. Introduction

1.1. Background of SLAM

In the field of robotics, simultaneous localization and mapping (SLAM) is a computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. Visual SLAM, which is becoming increasingly popular, is a type of SLAM that uses visual cameras such as RGB cameras [1] or RGB-D cameras [2] to capture images. State-of-the-art systems similar to ORB-SLAM2 [3] can work with both RGB and RGB-D cameras. Even if there is a stereo vision system [4] based on field programmable gate array in unmanned aerial vehicles, SLAM is also used as in [5]. LIDAR (light detection and ranging or laser imaging detection and ranging)-aided method [6] is also used with ORB-SLAM2.

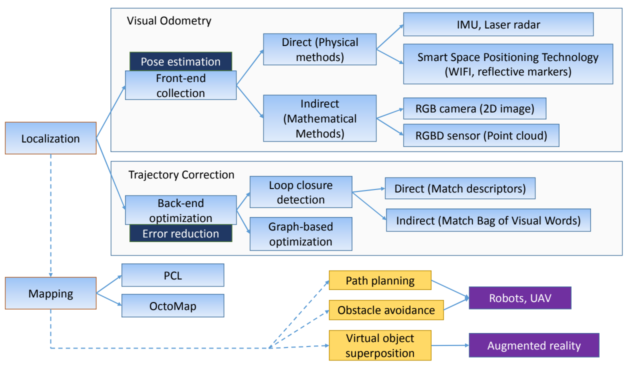

Fig. 1. SLAM framework.

SLAM is usually divided into two parts, namely, localization and mapping (**Fig. 1**). In this study, we focus on the localization part of the methodology, and then we show an application related to the mapping part at the end. The localization part is also divided into two parts: front-end and back-end. The front-end part is used to estimate the sensor pose (position and orientation), which we also call "visual odometry." The back-end part is used to reduce the accumulated error during continuous motions, which we also call "trajectory correction." Below, we present visual odometry and trajectory correction.

1.2. Visual Odometry

In robotics and computer vision, visual odometry is the process of determining the position and orientation of a robot equipped with sensors by analyzing the associated visual images. Visual odometry is divided into two parts:

feature tracking and pose estimation. In this study, we select the corner as the feature by using the ORB (oriented fast and rotated brief) feature descriptor. ORB is a fast robust local feature detector, first presented by Ethan Rublee et al. in 2011 [7]. We detect the ORB corners and use brute-force matching algorithm to track them. The corresponding 3D points are computed by a coordinate transformation based on the depth image captured by an RGB-D sensor. The pose estimation part is a perspective-n-point (PnP) [8] problem on estimating the sensor pose because we have *n* number of 2D keypoints on the 2D images and corresponding 3D points in 3D point clouds. We solve the PnP problem by minimizing the reprojection error based on an iterative optimization method.

1.3. Trajectory Correction

Visual odometry is a frame-to-frame operation that produces accumulated errors during exploration. A common method to reduce the accumulated error is to find a loop closure and use it in the graph-based optimization process. We build a graph for visual odometry. The vertices are the locations of a robot at different times, and the edges are the transformations of different poses. A loop closure means that the robot has almost returned to where it has been before. If we detect the correct loop closure, we adjust all sensor poses based on this new constraint.

There are two ways to detect loop closures: direct (match by descriptors) and indirect (match by visual words) methods. The direct method detects the features and computes the descriptors that are used to match images. This process is extremely slow on the second time scale. A random strategy [2] saves time for direct methods; however, it also makes the system unstable. The indirect method is based on the bag-of-words (BoW) model, which is commonly used in text analyses and language processing areas. In recent years, many researchers use it to recognize similar scenes [9, 10]. It converts the images to vectors and matches the images using those vectors [11]. This method is fast and allows all images to be compared with the new image. Moreover, the probability of missing the true loop closure is greatly reduced.

1.4. Motivation

In state-of-the-art systems, there is only a 2D constraint (adequately matched features on keypoints and 2D descriptors) for loop closures; this causes large errors on loop closure detection, and the weights of corresponding edges are single species and are indistinguishable. The weights should represent the reliability of loop closures. More details are discussed in Section 3.3. To improve the accuracy of loop closure detection, we train a visual vocabulary based on 3D descriptors and add a 3D constraint (adequately matched features on keypoints and 3D descriptors) for loop closure detection. In addition, to correct the trajectory more efficiently, we assign different weights to loop closures in the pose graph.

1.5. Paper Structure

In this paper, first we present the background and motivation of our research in Section 1. Then in Section 2, we discuss the related works as well as their limitations and remaining problems. Next in Section 3, we describe the details of the proposed ORB-SHOT SLAM (or OS-SLAM) system for offline vocabulary training as well as online loop closing by adding a 3D constraint and assigning different weights for loops in the pose graph. Then in Section 4, we discuss the experiments that we performed to determine the best feature detector-descriptor combination and to prove that the OS-SLAM system has the best performance for trajectory correction. Finally, we describe the design of a 3D reconstruction application based on the optimal trajectory in Section 5 and present the conclusion of our research in Section 6.

2. Related Works

2.1. RGB-D SLAM

2.1.1. Description

The first state-of-the-art SLAM system is RGB-D SLAM [2] for RGB-D cameras designed by the famous Computer Vision Group of Technical University of Munich (TUM). For the visual odometry part, this system has three choices on keypoints tracking which are used to calculate the camera pose: ORB, SIFT [12], and SURF [13]. This system requires landmark positions as keypoints to estimate the robot motion between two states.

For the trajectory correction part, the RGB-D SLAM system first computes a minimal spanning tree of limited depth from the pose graph, with the sequential predecessor as root node. Then, it removes *n* immediate predecessors from the tree, and randomly draws *k* frames from the tree with a bias toward earlier frames. When the robot revisits a place, once a loop closure is found, this procedure exploits the knowledge on the loop by preferring candidates near the loop closure in the sampling. To find large loop closures, RGB-D SLAM randomly samples *l* frames from a set of designated keyframes. A frame is added to the set of keyframes when it cannot be matched with the previous keyframe. In this way, the number of frames for sampling is greatly reduced, while keeping two keyframes having some part of the shared view.

2.1.2. Limitations

The RGB-D SLAM system uses a direct method to check whether two images are similar or not. It matches a pair of keypoint descriptors by using their distance in the descriptor space. This matching method could find the corresponding keypoints in two images; however, it requires too much time. The random strategy also has a problem of missing the true loop closure sometimes.

2.2. ORB-SLAM2

2.2.1. Description

The second state-of-the-art SLAM system is ORB-SLAM2 for monocular, stereo, and RGB-D cameras. This system has the best result on the RGB-D public dataset of TUM [14]. The best result means the smallest error between the estimated trajectory and the ground truth.

The ORB-SLAM2 system has three parallel threads: tracking, local mapping, and loop closing. First, it initializes the map that will contain the corresponding 3D points from previous frames, and the 3D points are computed by triangulating the ORB keypoints. It uses the EPnP (efficient perspective-n-points) [9] method to estimate the pose for the new frame from the previous frame based on the 3D points and corresponding ORB keypoints. If the new frame satisfies some particular conditions, we select it to be a keyframe and push it into a queue. Meanwhile, ORB-SLAM2 uses the BoW model to convert the new keyframe to a BoW vector and push it into an incremental database, which is an indirect method for loop closure detection.

For the trajectory correction part, ORB-SLAM2 has a complete strategy. It first checks whether there are some keyframes in the queue. If so, it deals with them one by one to find all keyframes. Then for each keyframe in the queue, it detects the loop closure in the database by the score of BoW vectors. If the score of two vectors is larger than a threshold and there are more than 10 frames between two frames, the system sets these two keyframes as a loop candidate. Then for each loop candidate, ORB-SLAM2 performs a geometry verification by computing the transformation between two keyframes, which means that if the extent of rotation or translation is substantially large, that loop will be rejected. Retrieving of map points is undertaken in the loop keyframes and neighboring frames to find more matches. Here, the map points are 3D points corresponding to the 2D features. If the number of matches is relatively larger than 40, which is determined from the open source code of ORB-SLAM2, the system accepts this is a loop. Finally, ORB-SLAM2 uses a graph-based optimization to correct the trajectory and update the map points.

2.2.2. Limitations

The ORB-SLAM2 system uses an indirect method to check whether two images are similar or not. We call it indirect because the system performs one-to-one matching not by descriptors but by using BoW vectors. This procedure saves a considerable amount of time so that it can detect all keyframes in the database one by one. The problem however is that, there is only one type of edge in the pose graph, i.e., only the 2D constraint of the loop closures would make errors when detecting the loop closures. The reason why we think of it as a problem is that we cannot decide and rely on a loop closure only by one type of 2D measurement. In Section 4.2, we find that providing more weight to a bad loop closure also leads to a bad result.

2.3. Depth-SLAM

2.3.1. Description

The LCDDI (loop closure detection using depth image) [15] evaluates different combinations of detectors and descriptors using the BoW model on depth images captured by RGB-D sensors. The authors of [15] provide a benchmark dataset consisting of a total number of 15 log files with several loops in various environments. The LCDDI implements a modular and easily extensible loop closure detector, then the authors use this to evaluate whether the depth features can work satisfactorily or not on their benchmark dataset. Finally, they find that the 3D descriptors have a better result than the 2D descriptors for loop closure detection except for SIFT, which is one type of 2D feature generation that includes keypoint detectors and descriptors.

2.3.2. Remaining Problems

The first problem is on the judgment standard for a true loop closure: two images are counted as a loop closure if their translation is less than 2 m and their rotation is less than 30° in [15]. The first translation constraint is not effective, for example, when we translate the sensor for 2 m keeping its rotation fixed, and the views are extremely different if the objects are near the sensor (e.g., a table). This case could not be considered to be detected as a true loop closure.

The second problem is that the authors want to use the 3D features to detect the loop closure in a full RGB-D SLAM system based on depth images. Currently however, we find that there is no 3D constraint for loop closing applied in RGB-D SLAM. All SLAM systems that use the 3D features are based on expensive sensors such as LIDAR. However, this solution is not suitable because of high price. We use depth cameras such as Kinect or Xtion to reduce the cost, and there has been no solutions using 3D constraint on this type of cheap depth sensor. SLAM systems using RGB-D sensors similar to that in [2] simply use 2D features for loop closing.

3. Proposed System

In this paper, we propose the ORB-SHOT SLAM (or OS-SLAM), which is a novel 3D loop closing method that use different weights for loop closures. First, we use 3D SHOT (signature of histograms of orientations) [16] descriptors based on normalized point cloud to describe ORB corners. Those 3D descriptors are computed from a large set of images to train 3D visual vocabulary offline (**Fig. 2**); more details are explained in Section 3.1. **Fig. 3** shows the online system. We obtain color images as well as depth images from RGB-D sensors such as Kinect or Xtion; moreover, we create corresponding point clouds from depth images. Then, we convert each new keyframe into two types of BoVW (bag of visual words) vectors. One is composed of words in the offline 2D ORB vocabulary, which are provided by ORB-SLAM2, and the other

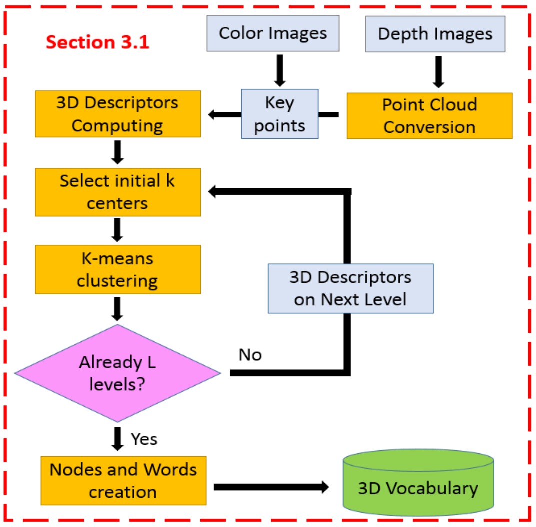

Fig. 2. Flow chart of the offline vocabulary creation.

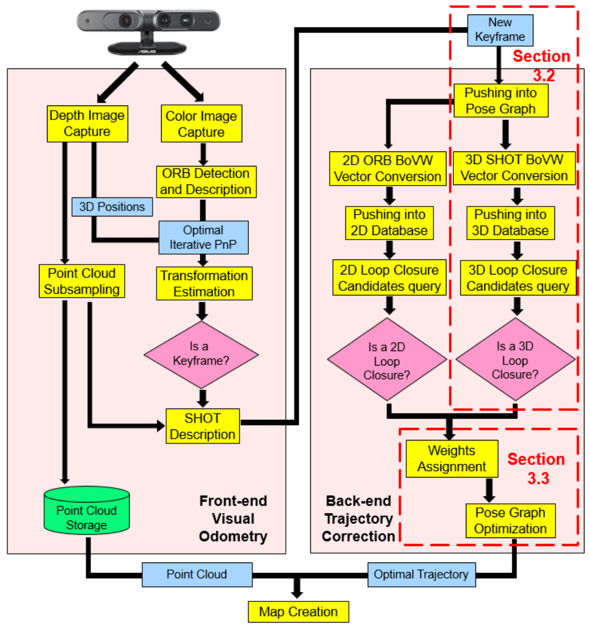

Fig. 3. Flow chart of the proposed online system.

is composed of words in the offline 3D SHOT vocabulary. Then, we push this keyframe represented by the two types of BoVW vectors into two incremental databases. We query the database for new keyframes, detect the 3D loop candidates, and store the scores between the SHOT BoVW vector of the new keyframe and that of each corresponding 3D loop candidate (Section 3.2). After detecting 2D loop closures using the same strategy as the ORB-SLAM2 system, we accept those loop closures that are also included in 3D loop candidates, and we assign them corresponding weights according to the scores stored previously. In the final graph-based optimization, we create edges with different weights for loop closures and correct the trajectory by solving a nonlinear least-squares optimization problem (Section 3.3).

3.1. Offline Vocabulary Training

3.1.1. Keypoints Detection

There are three types of keypoint detection methods. The first type includes methods such as SIFT3D [17] and Harris3D [18] and detects 3D keypoints from a normalized point cloud. A normalized process for a point cloud with 640×480 resolution requires a substantial amount of time and the resolution reduction will influence the result of keypoint detection; therefore, we reject this method.

The second type includes methods such as NARF [19] (normal aligned radial features) and detects 3D keypoints from a range image, which is generated by a depth image. However, there are approximately less than 100 keypoints in a range image with 640×480 resolution, which will negatively influence the matching result. We also reject this method.

The third type includes methods such as FAST [20, 21] and ORB and detects 2D keypoints from a color image. We select ORB because of its fast speed and good invariance (brightness, rotation, and scale). We have already used ORB tracking for motion estimation between any two frames (left side in **Fig. 3**); therefore, here we do not repeat the detection of keypoints, which saves a substantial amount of time. Another important reason to use ORB is that the ORB-SHOT combination has the best performance in the loop closure detection, which will be discussed in Section 4.2. We adopt this method (ORB) to obtain the 2D keypoints and then compute the corresponding 3D descriptors (SHOT).

3.1.2. 3D Descriptors Computation

There are two types of 3D descriptors: local and global. Local descriptors are computed for individual points that we provide as input. They have no notion of what an object is, and they just describe the local geometry around that point. Usually, the keypoints are the points that we want descriptors to compute for [22]. By contrast, global descriptors encode an object geometry. They do not compute for individual points, but for the entire cluster that represents the object. Because of this computation, a preprocessing step (segmentation) is required to retrieve possible candidates. Global descriptors are used for object recognition and classification, geometric analysis (object type, shape, etc.), and pose estimation [23] which are their great advantage. In this case, however, we do not require recognition, classification, and analysis on an object level. In addition, because we already had the keypoints, the local descriptors are a better choice to judge whether two images are similar or not.

Next, we present three local 3D descriptors, which will be tested in Section 4.2 for loop closure detection.

PFH (point feature histogram) [22] is one of the most important descriptors offered by PCL (point cloud library) and is the basis of other descriptors such as FPFH (fast point feature histogram) [24]. PFH provides accurate results; however, it has a drawback: it is too computationally expensive to perform in real time. For a cloud of n keypoints with k neighbors considered, the complexity is

Table 1. Performance comparison of 3D descriptors (extracted from **Table 3** in Section 4.2).

Detectors	Descriptors	Sensitivity (TPR)	Accuracy (ACC)	Time cost [ms]
NARF	FPFH	0.781	0.884	71
SIFT3D	FPFH	0.925	0.900	160
ORB	SHOT	0.919	0.906	88

$O(nk^2)$. Because of this, we created a derived descriptor named FPFH. FPFH considers only the direct connections between the current keypoint and its neighbors, removing additional links between neighbors. This reduces the complexity to $O(nk)$.

NARF [19] is the descriptor that does not take a point cloud as input. NARF is a keypoint detector as well as a keypoint descriptor. Because we already had keypoints from the corresponding ORB keypoints, here we just use NARF to describe those keypoints. The NARF descriptor encodes the information about changes around a point on the surface. First, a local range patch is created around the point. It is similar to a small range image centered at that point, aligned with the normal (it would appear as if we were looking at the point along the normal). Then, a star pattern with n beams is overlaid onto the patch, also centered at the point. For every beam, a value that reflects how much the surface changes under that condition is computed. The greater the change and the closer that point is to the center, the higher the final value. The n resulting values compose the final output of the descriptor.

SHOT [16] encodes information on the topology (surface) with a spherical support structure. This sphere is divided into 32 bins or volumes, with 8 divisions along the azimuth, 2 along the elevation, and 2 along the radius. For every volume, a one-dimensional local histogram is computed. The chosen variable is the angle between the normal of the keypoint and that of the current point within the volume (to be precise, the cosine, which was found to be more suitable). When all local histograms have been computed, they are stitched together in a final descriptor. SHOT makes use of a local reference frame, making it rotation invariant. It is also robust against noise and clutter.

Table 1 presents the experimental results comparing the three descriptors above with their best keypoint detection method. After the experiments in Section 4.2, we find that the SHOT descriptor has the best performance with ORB keypoint detection. The ORB-SHOT combination can detect the correct loop closure at 91.9% sensitivity (TPR, defined in Section 4.2) and at 90.6% accuracy (ACC, defined in Section 4.2). Details are discussed in Section 4.2. ORB-SHOT not only works extremely fast but also has the highest accuracy among the examined combinations of detector and descriptor.

3.1.3. 3D Vocabulary Training

After computing the 3D descriptors from the point cloud generated by depth images, we use k-means to divide those descriptors into different clusters by Euclidean distance. In this part, we describe how to train a 3D vocabulary.

Algorithm 1 k-means++

```
1:  Given observations $(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)$, cluster[$k$],
    distance[$n$]:
2:  Random $\boldsymbol{x}_c$ be the first center cluster[0], $c \in [0, n]$;
3:  distance[$i$] $\leftarrow |x_i - x_c|$;
4:  $m$: length[cluster]-1;
5:  *while* $m < k$
6:      Update distance[$n$] by cluster[$m$]
7:      dist_sum $\leftarrow \sum_{i=1}^{n} distance[i]$
8:      Random cut_d $\in [0, \text{dist\_sum}]$
9:      *for* $j = 0, 1, 2, \ldots, n$
10:         dist_up_now $+=$ distance[$j$]
11:         *if* dist_up_now $>$cut_d
12:             cluster[$m++$] $\leftarrow \boldsymbol{x}_j$;
13:     *end(for)*;
14: *end(while)*;
```

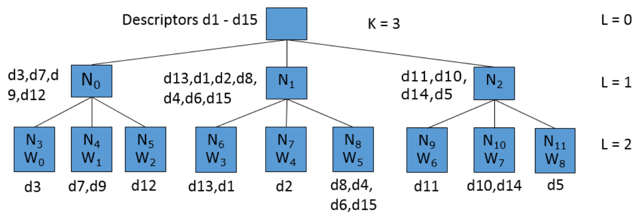

Fig. 4. Simple model of vocabulary tree with $K = 3$ and $L = 2$ (excluding the root).

K-means clustering is an unsupervised learning method when no outputs correspond with inputs. First, we 1) create k cluster centers and 2) compute the distance between each point and each center. Then, we 3) divide those points into different clusters by distance and 4) compute new centers for k clusters. Finally, we repeat steps 2) to 4) until the centers of clusters do not move again. There are two problems: how to determine k (the number of clusters) and how to determine the centers of k clusters. The solution to first problem depends on the number of observations and requires some experience. For our visual vocabulary, we choose $k = 10$ and more details will be discussed later. The second problem is more difficult. Here, we use k-mean++ [25] (an algorithm for choosing the initial value of the k-means clustering algorithm) to create suitable initial centers. The algorithm is as follows (**Algorithm 1**):

We 1) select randomly one point to be the first center at the beginning. Then, we 2) select a random number between 0 and the sum of the distance between each point and the center. Then, we 3) find the new center such that the partial sum is just larger than the random number. We repeat steps 2) and 3) until k cluster centers are found, and finally, we obtain k suitable initial centers.

After presenting the k-means clustering algorithm, we explain how to build a visual vocabulary tree. **Fig. 4** is a simple model showing the process of vocabulary creation. Suppose we have a total of 15 descriptors and we divide these descriptors into 3 clusters ($k = 3$) on each level in

Algorithm 2 Building visual vocabulary

```
1:  Given total descriptors $N$, clusters on each level $K$,
    total levels $L$;
2:  Func k-means(parent_id, descriptors, current_level)
3:  If descriptors is empty -> return;
4:  clusters[$K$], groups[$K$];
5:  if length[descriptors] <= $K$
6:    for $i = 0, 1, \ldots$ length[descriptors]
7:      groups[$i$] $\leftarrow i$; clusters[$i$] $\leftarrow$ descriptors[$i$];
8:    end(for);
9:  else
10:   first_time $\leftarrow$ true; goon $\leftarrow$ true;
11:     while goon
12:     if first_time
13:       Find initial $K$ centers by $k$-means++;
14:     else
15:       for $c = 0, 1, \ldots, K$
16:         cluster_descriptors[length[groups[$c$]]];
17:         for $i = 0, 1, \ldots$ length[groups[$c$]]
18:           cluster_descriptors[$i$] $\leftarrow$ descriptors[$i$];
19:         end(for);
20:         Compute new centers for cluster[$c$];
21:       end(for);
22:       Find each descriptors has shortest distance
    with which center;
23:       if first_time first_time $\leftarrow$ false;
24:       else goon $\leftarrow$ false;
25:         if there are any descriptor belongs to
    a new center goon $\leftarrow$ true;
26:   end(while);
27:   Push the new centers into new nodes and
    associate to parent node;
28:   if current_level $<L$
29:     for $i = 0, 1, \ldots, K$
30:       $id \leftarrow$ node[parent_id][$i$];
31:       child_descriptors[length[groups[$i$]]];
32:       for $j = 0, 1, \ldots$ length[groups[$i$]]
33:         child_descriptors[$j$] $\leftarrow$ descriptors[$j$];
34:       end(for);
35:       if length[groups[$i$]] $> 1$
36:         Recursively call k-means($id$,
    child_descriptors, current_level+1)
37:     end(for);
```

the vocabulary tree, which has 2 levels excluding the root node. There are $3^2 = 9$ nodes on the bottom level of the tree. These descriptors are clustered by the different Euclidean distances between each other into 3 clusters on level-1. Then, we can observe that there is same clustering for each cluster on level-2, and finally, we obtain 9 clusters in the tree. These nodes on the bottom level are the centers of 9 clusters, and we consider these nodes as words that compose the final vocabulary. The entire algorithm of building the visual vocabulary is described in **Algorithm 2**.

After creating the visual vocabulary tree, we traverse the tree and save the nodes to the vocabulary files, except the root node. If a node is a leaf node, we call it a "word" and set its weight as 1. By contrast, the weights of nodes that are not leaf nodes are 0. This type of weighting is called binary. We also save the words into the vocabulary file.

Here, we select approximately 5,000 color images and their corresponding depth images from different scenes based on the public RGB-D dataset by TUM to train the vocabulary. They provide a large dataset containing the RGB-D data and the ground-truth data with the goal of establishing a novel benchmark for evaluation of the visual odometry systems and the visual SLAM systems. We detect 500 ORB keypoints from a color image and compute the corresponding 3D SHOT descriptors from the point cloud generated by a depth image. The number of descriptors is less than 500 because some keypoints are bad for computing. There are many NaN (not a number in PCL) 3D points around the bad keypoints. We obtain approximately 2 million descriptors from 4,687 images and create a vocabulary with $K = 10$ and $L = 6$. The total number of levels does not include the root level. After the k-means clustering training, we obtain 609,998 words in all and save them into a visual vocabulary.

3.2. Loop Closure Detection

The previous section (Section 3.1) presented the offline visual vocabulary creation process based on 3D SHOT descriptors. In this section, we detect the loop closures by using the 3D visual vocabulary and the incremental database that contains a list composed of all visual words.

3.2.1. BoVW Vector and Score

We convert a new keyframe to a BoVW vector by its SHOT descriptors based on the 3D vocabulary. The BoVW vector has the elements such as $<ID_{word}$, $Value_{word}>$. The ID_{word} is the index of the word in the vocabulary, and the $Value_{word}$ is computed by

$$Value_{word} = \frac{weight_{word}}{\sum weight_{word}} \quad \ldots\ldots\ldots \quad (1)$$

where the *weight* is 1 here. After obtaining the BoVW vectors, we compute the similarity s between two BoVW vectors, which is the score between two keyframes. The score is based on $L1$ norm:

$$\begin{aligned} s(v^a, v^b) &= 1 - \frac{1}{2}\left|\frac{v^a}{|v^a|} - \frac{v^b}{|v^b|}\right| \\ &= 1 - \frac{1}{2}\left(2 + \sum_{i=1}^{n}\left|v_i^a - v_i^b\right| - |v_i^a| - \left|v_i^b\right|\right) \quad \ldots \quad (2) \end{aligned}$$

where n is the number of the same words in two vectors.

In general, the number of descriptors in different frames is approximately 500; however, it is not always completely the same because the descriptors of some keypoints are difficult to compute. Thus, the $Value_{word}$ could be 1/490 or 1/495 sometimes, and the score could be

$$\begin{aligned} & s(v^a, v^b) = \\ & 1 - \frac{1}{2}\left(2 + \left(\left(\frac{1}{490} - \frac{1}{495}\right) - \frac{1}{490} - \frac{1}{495}\right) + \cdots\right) \\ & \in [0, 1]. \quad \ldots\ldots\ldots\ldots\ldots \quad (3) \end{aligned}$$

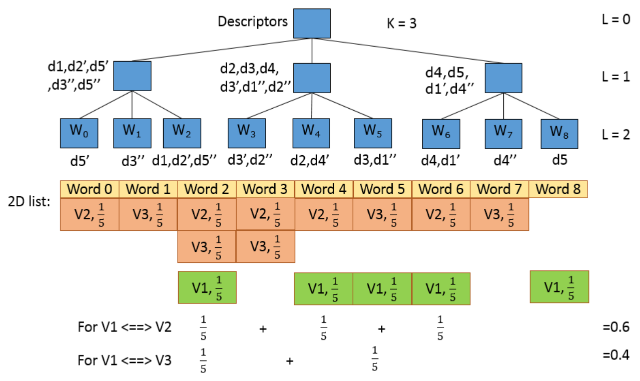
Descriptors
K = 3
L = 0
d1,d2′,d5′ ,d3″,d5″
d2,d3,d4, d3′,d1″,d2″
d4,d5, d1′,d4″
L = 1
W₀ W₁ W₂ W₃ W₄ W₅ W₆ W₇ W₈
L = 2
d5′ d3″ d1,d2′,d5″ d3′,d2″ d2,d4′ d3,d1″ d4,d1′ d4″ d5
2D list:
Word 0 Word 1 Word 2 Word 3 Word 4 Word 5 Word 6 Word 7 Word 8
V2, 1/5 V3, 1/5 V2, 1/5 V2, 1/5 V2, 1/5 V3, 1/5 V2, 1/5 V3, 1/5
V3, 1/5 V3, 1/5
V1, 1/5 V1, 1/5 V1, 1/5 V1, 1/5 V1, 1/5
For V1 <==> V2 1/5 + 1/5 + 1/5 =0.6
For V1 <==> V3 1/5 + 1/5 =0.4

Fig. 5. Query by 2D list in database.

Actually, the same words in two vectors are mostly less than 100, which means that the scores are mostly less than 0.2. For the weight assignment, we set $weight = s(v^a, v^b) \times 10000 \in [0, 10000]$, so that the more similarity the two keyframes have, the larger the weight becomes. More details are explained in Section 3.3

3.2.2. Incremental Database

Next, we present the incremental database with a size of 0 at the beginning. Once we obtain a new keyframe, we convert it to a BoVW vector and push it into the database. Then, we compare the new BoVW vector with all other BoVW vectors in the database, and obtain the scores between them. After sorting the scores, we choose the highest one, which is most similar to the new keyframe. Then, we obtain the loop closure, or we set a threshold and obtain the loop candidates by returning those keyframes with scores that are higher than the threshold.

This type of query in database is usual; however, it is not very efficient. Here, we build a two-dimensional list to improve the query efficiency as shown in **Fig. 5**. The first dimension stores the indexes of words, while the second stores the record on which frame a word appears and the weight of this word in that frame. Here, we provide a simple model based on three BoVW vectors. As we can observe in **Fig. 5**, the previous keyframes, BoVW vector-2 ($V2$, containing 5 descriptors: $d1'$, $d2'$, $d3'$, $d4'$, and $d5'$) and BoVW vector-3 ($V3$, containing 5 descriptors: $d1''$, $d2''$, $d3''$, $d4''$, and $d5''$) have been stored in the 2D list. The values of each word are 1/5 in vector-2 and vector-3. For the new BoVW vector-1 ($V1$, containing 5 descriptors: $d1$, $d2$, $d3$, $d4$, and $d5$), we query in the 2D list by descriptors individually. The first descriptor $d1$ belongs to word-2 (W_2); thus, we determine which previous vectors contain this word-2. Then, we compute the sub score by their weights in their vectors. After the query for all descriptors, we calculate the final score by adding the subscores. Then, we push BoVW vector-1 into this database, obtain a new vector, and continue to query again. Based on this type of operation, each element in the new vectors can be compared with many previous vectors at the same time, which saves a substantial amount of

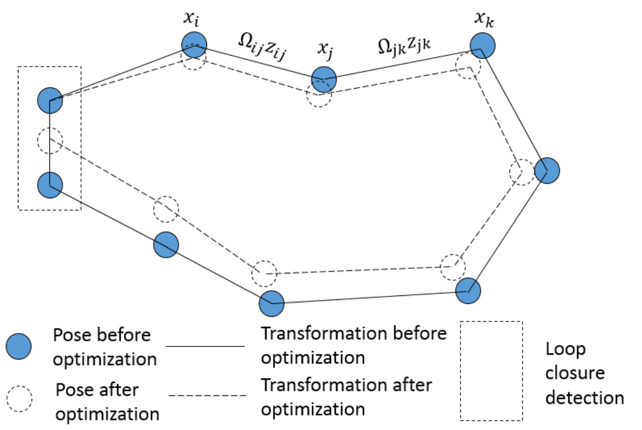

Fig. 6. Graph-based optimization.

time. Each frame has a fixed number of BoVW. Moreover, each BoVW needs to query

$$T = \frac{Number_{\text{words\_on\_a\_frame}} * n}{Number_{\text{total\_words}}} \quad . \; . \; . \; . \; . \; . \; . \; . \quad (4)$$

times from the database. Therefore, the computational complexity of our retrieval algorithm is $O(n)$.

After the query, we return several loops with highest scores and set them as the loop closure candidates. By combining the 2D constraint and geometric constraint, we can decide the final loop closures, which must satisfy all the 3D, 2D, and geometric constraints.

3.3. Graph-Based Optimization

In this section, we present why the graph-based optimization can reduce the accumulated error during the exploration by loop closures that we detected previously.

Figure 6 shows the pose graph, where the vertices represent the sensor locations at different times and the edges are pose transformations. The optimization objective of the pose graph is to minimize the error function (Eq. (5)) and obtain the optimal solution based on the loop closure constraint:

$$F(\boldsymbol{x}) = \sum_{<i,j>\in C} e(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{z}_{ij})^T \boldsymbol{\Omega}_{ij} e(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{z}_{ij}) \quad . \quad (5)$$

$$\boldsymbol{X}^* = \arg\min_{X} F(\boldsymbol{X})$$

where $\boldsymbol{\Omega}$ is the weight matrix, and e is the error function on the measurement of transformation between two poses.

As we can observe in **Fig. 6**, $\boldsymbol{X} = \left(\boldsymbol{x}_1^T, \ldots, \boldsymbol{x}_n^T\right)^T$ is a vector of sensor poses. The term $\boldsymbol{z}_{ij}$ represents the measurement between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, i.e., the pairwise transformation computed by the visual odometry. Additionally, $e(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{z}_{ij})$ is a vector of the error function that measures how well the poses $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ satisfy the constraint $\boldsymbol{z}_{ij}$. It is 0 when $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ perfectly match the constraint, i.e., the difference of the poses exactly matches the calculated transformation based on the measurements. Finally, $\boldsymbol{\Omega}_{ij}$ represents the weight matrix of a constraint relating the

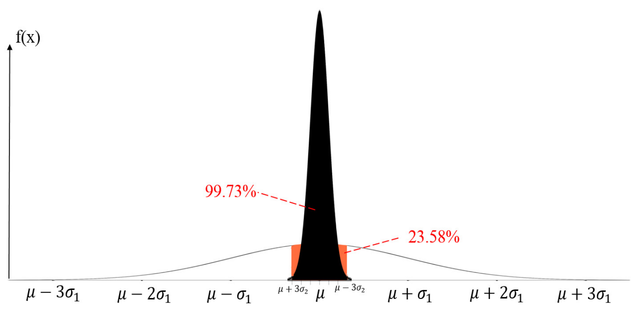

Fig. 7. Examples of Gaussian distribution with different weights and different probabilities.

poses $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. We replace the error function by a simplified version:

$$e(\boldsymbol{x}_i,\boldsymbol{x}_j,z_{ij}) \overset{\text{def}}{=} e_{ij}(\boldsymbol{x}_i,\boldsymbol{x}_j) \overset{\text{def}}{=} e_{ij}(\boldsymbol{x}). \quad . \; . \; . \; . \quad (6)$$

As we mentioned in the last paragraph, the constraint z_{ij} in the error function $e(\boldsymbol{x}_i,\boldsymbol{x}_j,z_{ij})$ is generally the measurement obtained by a camera or inertial measurement unit between two poses $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. The $e(\boldsymbol{x}_i,\boldsymbol{x}_j,z_{ij})$ is error between the measurement and real value, which should be minimized. The simplified error function $e_{ij}(\boldsymbol{x})$ in Eq. (6) is just convenient to write in the later equations.

For Eq. (5), we can also describe it by the maximum likelihood. We assume that the measurement is affected by Gaussian white noise; thus, the distribution of each measurement is the Gaussian distribution (totally n vertices in the pose graph):

$$f_x(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_i,\boldsymbol{x}_j,\boldsymbol{x}_k,\ldots,\boldsymbol{x}_n) = \frac{1}{(2\pi)^{0.5k}|\boldsymbol{\Sigma}|^{0.5}}\exp\left(-\frac{1}{2}(\boldsymbol{x}-\mu)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\mu)\right) \quad (7)$$

where the parameter μ is the expected value and $\boldsymbol{\Sigma}$ is the covariance matrix. For each measurement, we assume that it should appear with the maximum probability, and the maximum likelihood probability is

$$\boldsymbol{l}_{ij} \propto [z_{ij}-\hat{z}_{ij}(\boldsymbol{x}_i,\boldsymbol{x}_j)]^T\,\boldsymbol{\Omega}_{ij}\,[z_{ij}-\hat{z}_{ij}(\boldsymbol{x}_i,\boldsymbol{x}_j)]. \quad (8)$$

From Eqs. (7) and (8), we can find that actually, the weight matrix $\boldsymbol{\Omega}$ is the inverse of the covariance matrix $\boldsymbol{\Sigma}$, and the elements w_{ij} in the weight matrix are also the inverse of the elements in the covariance matrix. In Eq. (9), σ is the standard deviation:

$$w_{ij} = cov_{ij}^{-1} = \sigma_{ij}^{-2}. \quad . \; . \; . \; . \; . \; . \; . \; . \; . \; . \quad (9)$$

Here, we provide two examples of Gaussian distributions that have different weights, and different probabilities (**Fig. 7**). The standard deviation σ_1 of the first Gaussian distribution is equal to 0.1, which means that $weight_1$ is equal to 100. Thus, the probability of the measurement appearing in the interval $[\mu-0.03,\,\mu+0.03]$ is 23.58%. The standard deviation σ_2 of the second Gaussian distribution is equal to 0.01, which means that $weight_2$ is equal to 10,000. Thus, the probability of the measurement appearing in the interval $[\mu-0.03,\,\mu+0.03]$ is 99.73%. Therefore, the larger the weight, the smaller the standard deviation, and the more the trust for loop closures, which means that the measurement is more reliable.

The weight matrix is the most important point in this study. We assign different weights to loop closures based on the scores of their BoVW vectors, because we consider that the loops with larger scores are more trustworthy:

$$weight = score(v^a,v^b)\times 10000 \in [0,10000]. \quad . \quad (10)$$

Because the function of the sensor pose is nonlinear, we use an iterative method to solve the problem. Assuming $\breve{\boldsymbol{x}}$ is the iterative initial solution, we have

$$e_{ij}\left(\breve{\boldsymbol{x}}_i+\Delta\boldsymbol{x}_i,\,\breve{\boldsymbol{x}}_j+\Delta\boldsymbol{x}_j\right) = e_{ij}\left(\breve{\boldsymbol{x}}+\Delta\boldsymbol{x}\right) \simeq e_{ij}+\boldsymbol{J}_{ij}\Delta\boldsymbol{x} \quad (11)$$

where $\boldsymbol{J}_{ij}$ is Jacobian matrix and we use e_{ij} to simplify $e_{ij}(\breve{\boldsymbol{x}})$. Thus, one error of Eq. (5) becomes

$$\begin{aligned} & F_{ij}\left(\breve{\boldsymbol{x}}+\Delta\boldsymbol{x}\right) \\ & = e_{ij}\left(\breve{\boldsymbol{x}}+\Delta\boldsymbol{x}\right)^T\Omega_{ij}e_{ij}\left(\breve{\boldsymbol{x}}+\Delta\boldsymbol{x}\right) \\ & \simeq \left(e_{ij}+\boldsymbol{J}_{ij}\Delta\boldsymbol{x}\right)^T\Omega_{ij}\left(e_{ij}+\boldsymbol{J}_{ij}\Delta\boldsymbol{x}\right) \\ & = e_{ij}^T\Omega_{ij}e_{ij}+2e_{ij}^T\Omega_{ij}\boldsymbol{J}_{ij}\Delta\boldsymbol{x}+\Delta\boldsymbol{x}^T\boldsymbol{J}_{ij}^T\Omega_{ij}\boldsymbol{J}_{ij}\Delta\boldsymbol{x} \\ & = c_{ij}+2\boldsymbol{b}_{ij}^T\Delta\boldsymbol{x}+\Delta\boldsymbol{x}^T\boldsymbol{H}_{ij}\Delta\boldsymbol{x}. \quad . \; . \; . \; . \; . \quad (12) \end{aligned}$$

Substituting Eq. (12) into Eq. (5), we have

$$\begin{aligned} & F\left(\breve{\boldsymbol{x}}+\Delta\boldsymbol{x}\right) = \sum_{<i,j>\in C} F_{ij}\left(\breve{\boldsymbol{x}}+\Delta\boldsymbol{x}\right) \\ & \simeq \sum_{<i,j>\in C} c_{ij}+2\boldsymbol{b}_{ij}^T\Delta\boldsymbol{x}+\Delta\boldsymbol{x}^T\boldsymbol{H}_{ij}\Delta\boldsymbol{x} \\ & = c+2\boldsymbol{b}^T\Delta\boldsymbol{x}+\Delta\boldsymbol{x}^T\boldsymbol{H}\Delta\boldsymbol{x}. \quad . \; . \; . \; . \; . \; . \; . \quad (13) \end{aligned}$$

To obtain the solution, we make the derivation for Eq. (13) while setting the original derivative as 0:

$$\boldsymbol{H}\Delta\boldsymbol{x}^* = -\boldsymbol{b} \quad . \; . \; . \; . \; . \; . \; . \; . \; . \; . \; . \; . \; . \; . \quad (14)$$

where $\boldsymbol{H}$ is the information matrix of the entire pose graph. Then, the final solution $\boldsymbol{x}^*$ is solved by adding $\Delta\boldsymbol{x}^*$ and make the error function Eq. (6) be convergent with Gauss-Newton iteration [26]. We can also use the Levenberg-Marquardt algorithm [27] to solve a nonlinear least-squares problem, which adds a relaxation factor λ to control the gradient descent speed:

$$(\boldsymbol{H}+\lambda\boldsymbol{I})\Delta\boldsymbol{x}^* = -\boldsymbol{b}. \quad . \; . \; . \; . \; . \; . \; . \; . \; . \; . \quad (15)$$

Here, we have a problem that $\boldsymbol{x}$ is a pose represented by a rotation matrix and a translation matrix. For the rotation matrix, we do not have any additional operation. Therefore, we need to define an additional operation for the equations above using Lie algebra (a vector space together with a non-associative multiplication), and convert the transformation from Lie group SE(3) to Lie algebra se(3).

In the actual programming, we use the G2O [28] library to perform the optimization. **Algorithm 3** shows the process of the graph-based optimization.

Algorithm 3 Process of graph-based optimization

```
1:  Require: $\breve{\boldsymbol{X}} = \breve{\boldsymbol{X}}_{1:T}$ : initial guess,
    $C = \{< e_{ij}, \Omega_{ij} >\}$ : constraint
2:  Ensure: $\boldsymbol{X}^*$ : new solution, $\boldsymbol{H}^*$ : new
    information matrix
3:    // find the maximum likelihood solution
4:    While ($\breve{\boldsymbol{X}}$ is not converged) do
5:      $\boldsymbol{b} \leftarrow 0 \quad \boldsymbol{H} \leftarrow 0$
6:      For all $< e_{ij}, \Omega_{ij} > \in C$ do
7:        // compute the Jacobians $\boldsymbol{A}_{ij}$ and $\boldsymbol{B}_{ij}$ of the
    error function
8:        $\boldsymbol{A}_{ij} \leftarrow \frac{\partial e_{ij}(\boldsymbol{X})}{\partial x_i}|_{\boldsymbol{X}=\breve{\boldsymbol{X}}} \quad \boldsymbol{B}_{ij} \leftarrow \frac{\partial e_{ij}(\boldsymbol{X})}{\partial x_j}|_{\boldsymbol{X}=\breve{\boldsymbol{X}}}$
9:        // compute the contribution of this
    constraint to the linear system
10:       $\boldsymbol{H}_{[ii]} += \boldsymbol{A}_{ij}^T \Omega_{ij} \boldsymbol{A}_{ij} \quad \boldsymbol{H}_{[ij]} += \boldsymbol{A}_{ij}^T \Omega_{ij} \boldsymbol{B}_{ij}$
11:       $\boldsymbol{H}_{[ji]} += \boldsymbol{B}_{ij}^T \Omega_{ij} \boldsymbol{A}_{ij} \quad \boldsymbol{H}_{[jj]} += \boldsymbol{B}_{ij}^T \Omega_{ij} \boldsymbol{B}_{ij}$
12:       // compute the coefficient vector
13:       $\boldsymbol{b}_{[i]} += \boldsymbol{A}_{ij}^T \Omega_{ij} \boldsymbol{e}_{ij} \quad \boldsymbol{b}_{[j]} += \boldsymbol{B}_{ij}^T \Omega_{ij} \boldsymbol{e}_{ij}$
14:     end for
15:     // keep the first node fixed
16:     $\boldsymbol{H}_{[00]} += \boldsymbol{I}$
17:     // solve the linear system using sparse
    Cholesky factorization
18:     $\Delta\boldsymbol{X} \leftarrow$ solve ($\boldsymbol{H}\Delta\boldsymbol{X} = -\boldsymbol{b}$)
19:     // update the parameters
20:     $\breve{\boldsymbol{X}} += \Delta\boldsymbol{X}$
21:   end while
22:   $\boldsymbol{X}^* \leftarrow \breve{\boldsymbol{X}}$
23:   $\boldsymbol{H}^* \leftarrow \boldsymbol{H}$
24:   // release the first node
25:   $\boldsymbol{H}^*_{[00]} -= \boldsymbol{I}$
26:   Return $< \boldsymbol{X}^*, \boldsymbol{H}^* >$
```

4. Experiment

We first present the TUM public RGB-D dataset in Section 4.1. Then, we describe the two experiments. Section 4.2 explains how to select the best combination of the detectors and 3D descriptors to detect an accurate loop closure. In Section 4.3, we show that the proposed ORB-SHOT SLAM (or OS-SLAM) system has the best performance for trajectory correction among the state-of-the-art systems.

4.1. Dataset Introduction

This dataset [14] is provided by the famous Computer Vision Group of Technical University of Munich (TUM). They provide a large dataset containing the RGB-D data and ground-truth data with the goal of establishing a novel benchmark for evaluation of visual odometry systems and visual SLAM systems. The dataset contains the color and depth images of a Microsoft Kinect (or ASUS Xtion) sensor along with the ground-truth trajectory by an external sensor. The data was recorded at full frame rate (30 Hz) and sensor resolution (640×480). The ground-truth trajectory was obtained from a high-accuracy motion-capture system with eight high-speed tracking cameras (100 Hz). Finally, they propose an evaluation criterion for measuring the quality of the estimated camera trajectory of visual SLAM systems. Many researchers have used this RGB-D dataset to evaluate their studies in recent years.

Table 2. Number of loops in four scenes of dataset.

Scenes	Number of loops	Number of non-loops	Total keyframes
fr1_desk2	25	37	62
fr1_room	26	110	136
fr2_desk	54	169	223
fr3_long_office	55	194	249
Total	160	510	670

4.2. Detector and Descriptor Selection

4.2.1. Purpose

In Section 3.1.1 (keypoints detection) and Section 3.1.2 (3D descriptors computation), we have presented several types of keypoints and descriptors. In this section, we select the best combination of keypoint detectors and descriptors based on precision-recall (PR) [29] and receiver operating characteristic (ROC) curve by conducting several experiments. Precision is a function of retrieved instances that are relevant, while recall is a fraction of relevant instances that are retrieved. The ROC curve, which illustrates the performance of a binary classifier system when its discrimination threshold is varied, is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

4.2.2. Procedure

First, we select several scenes containing loop closures from the TUM public RGB-D dataset: *fr1_desk2, fr1_room, fr2_desk*, and *fr3_long_office_household*. Each of these scenes contains more than one loop closure. We check the ground truth, which includes 6-DOF poses for each image frame, and we determine whether two images were taken at the same place by considering the Euclidean distance of the translation and the norm of rotation between the 6-DOF poses: two images are counted as a loop when their translation is less than 0.5 m and their norm of rotation is less than 0.3 rad (17°). Because we define that the distance between two images of a loop must be less than 0.5 m, we must ensure that these two images were not taken at a short time. Here, we simply define that there are 10 image frames between two keyframes. Therefore, there must be at least 10 keyframes, or 100 frames between two images in order for them to be considered as a loop. Then, we consider that a true loop has the shortest distance and norm of rotation. After computing, we present the number of true loops in the four scenes above in **Table 2**.

At first glance, the loop closure detection is a binary classification problem: given a place described by an image and a database of images and decide whether the place has been visited before. Therefore, it appears natural to evaluate the performance in terms of *TPR*, which we can

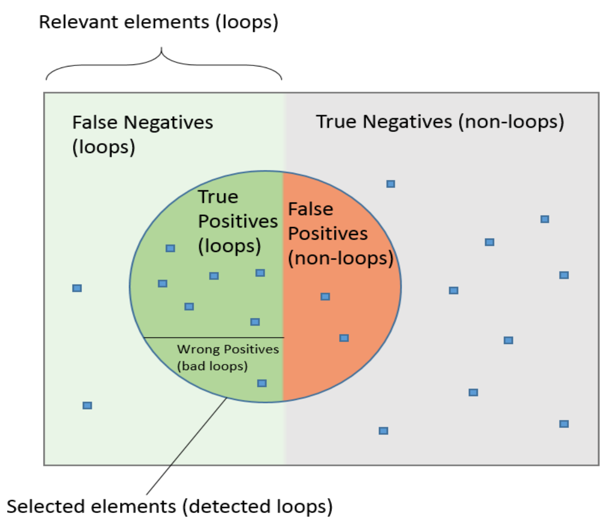
Relevant elements (loops)
False Negatives (loops)
True Negatives (non-loops)
True Positives (loops)
False Positives (non-loops)
Wrong Positives (bad loops)
Selected elements (detected loops)

Fig. 8. Precision and recall (PR).

call sensitivity or recall (**Fig. 8**). If a loop is detected, it is a *true positive* (*TP*). If a loop is not detected, it is a *false negative* (*FN*). If a non-loop is detected, it is a *false positive* (*FP*). If a non-loop is not detected, it is a *true negative* (*TN*). There is a problem however, if a loop is detected but the corresponding frame is not contained in the loop candidates; we call this *wrong positive* (*WP*). For example, frame-A has a corresponding frame-B; however, we detect that frame-A has no loops with any frames, which is called *FP*. Frame-A has a corresponding frame-B; however, we detect that frame-A has a corresponding frame-C, not frame-B to be the loop; this is called *WP*. We can compute the *TPR*, *FPR*, and accuracy (*ACC*) by using the following equations:

$$TPR = \frac{TP}{TP+WP+FN} \quad \text{(16)}$$

$$FPR = \frac{FP}{FP+TN} \quad \text{(17)}$$

$$ACC = \frac{TP+TN}{TP+FP+WP+TN+FN} \quad \text{(18)}$$

4.2.3. Result

In Section 3.2 (loop closure detection), we have presented how to detect loop closures by using BoVW vectors based on the incremental database. We return those loops with scores that are higher than a threshold S_t. The number of loop closures depends on the threshold S_t. Here, we provide some values (**Table 3**) with the most suitable threshold, which is $S_t = 0.2$. This threshold is decided based on some preliminary experiments, where different thresholds have different values but the same result.

4.2.4. Analysis

Good data are listed in bold while the best data are in bold and italic in **Table 3**. We can observe that for the detectors, ORB (7, 8, and 9) and SIFT3D (4, 5, and 6) show a better performance. Though NARF (1, 2, and 3) has a better *FPR*, its *TPR* is bad. For descriptors, SHOT (3, 6, and 9) demonstrates the best performance for both *TPR* and *ACC*.

Because we considered *TPR* as well as *FPR*, putting the values into the ROC space is an effective way to evaluate which combination is the best. From **Fig. 9**, we can observe that the point (*FPR*, *TPR*) is closer to the upper left in the chart, which means that the *ACC* is higher. Thus, SIFT3D-FPFH (5), SIFT3D-SHOT (6), and ORB-SHOT (9) show good results (actually, we can observe that the last one is the best). SIFT3D detection requires too much time, approximately more than 100 ms; however, the ORB detection requires approximately 6 ms and detection is already performed in the pose estimation part (left side in **Fig. 3**). Considering the time cost constraint, we consider that ORB-SHOT (9) is the best combination for loop closure detection.

4.3. Trajectory Accuracy Comparison

4.3.1. Purpose

In this section, we show some results (speed and precision) obtained by the proposed ORB-SHOT SLAM (or OS-SLAM) system compared with other related systems such as ORB-SLAM2 and RGB-D SLAM to prove that the OS-SLAM system has the best performance among the systems for trajectory correction.

4.3.2. Procedure

In the testing and debugging category, sequences such as *fr1_xyz* contain only translation motions along the principal axes of Kinect, while the orientation was maintained and (mostly) fixed. The sequence *frg2_xyz* contains very clean data for debugging translations. Kinect2 was moved along the principal axes in *x*-, *y*- and *z*-directions very slowly. The slow camera movement basically ensures that there is (almost) no motion blur and there are no rolling shutter effects in the data.

In the handheld SLAM category, sequences such as *fr1_room* start with the four desks but continue to the surrounding wall of the room until the loop is closed. This sequence is well suited for evaluating how satisfactory a SLAM system can cope with loop closures. In the sequence such as *fr3_long_office_household*, the Asus Xtion sensor was moved along a large circle through household scenes and office scenes with much texture and structure. Because the end of the trajectory overlaps with its beginning, there is a large loop closure. It is also well suited for evaluating how satisfactorily a SLAM system can cope with large loop closures.

In the robot SLAM category, sequences such as *fr2 pioneer slam* are recorded from Kinect2 mounted on top of a pioneer robot (a two-wheel drive mobile robot). The pioneer robot is controlled by a joystick through a maze with tables, containers and other walls, so that several loops have been closed for map building.

Table 3. True positive rate for different combinations with $S_t = 0.2$.

Combination	Detectors	Descriptors	Detected loops	*TP*	*WP*	*FP*	*TPR*	*FPR*	*ACC*	Time cost [ms]
1. N-N	NARF	NARF	172	122	8	42	0.763	***0.082***	0.881	19
2. N-F	NARF	FPFH	173	125	5	43	0.781	**0.084**	0.884	71
3. N-S	NARF	SHOT	177	126	6	45	0.788	0.088	0.882	85
4. S-N	SIFT3D	NARF	197	136	7	54	0.850	0.106	0.884	100
5. S-F	SIFT3D	FPFH	208	148	5	55	**0.925**	0.108	**0.900**	160
6. S-S	SIFT3D	SHOT	211	149	5	57	***0.931***	0.112	**0.899**	175
7. O-N	ORB	NARF	180	129	9	43	0.806	**0.084**	0.890	21
8. O-F	ORB	FPFH	196	141	6	49	0.888	0.096	0.898	73
9. O-S	ORB	SHOT	205	147	8	50	**0.919**	0.098	***0.906***	88

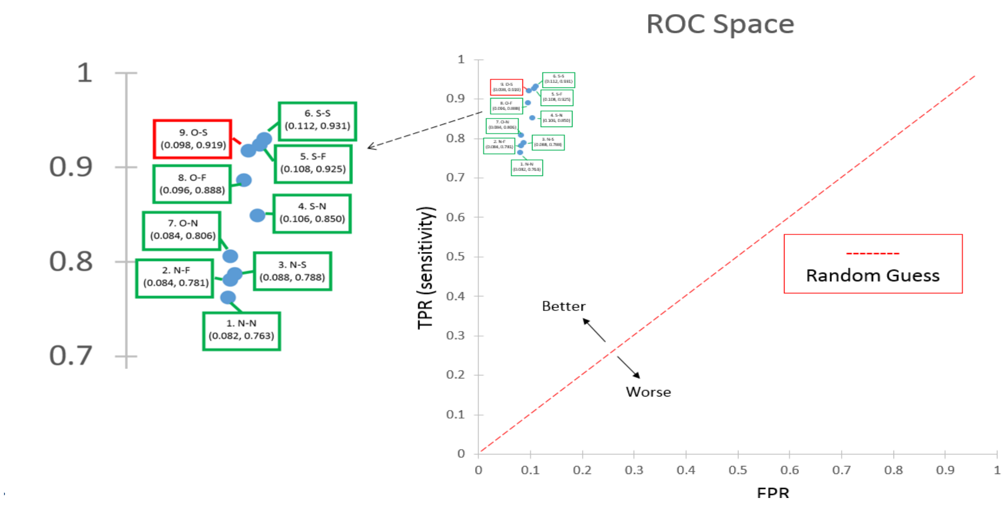

Fig. 9. Examination of the best combination of detector and descriptor based on ROC space.

For the structure category, in sequences such as *fr3 nostructure texture near withloop*, the Asus Xtion was moved by 1 m height in a circle on a textured planar surface. The texture is highly discriminative as it consists of conference posters. Loop closures exist because the beginning and end of the trajectory overlap.

We select 12 scenes from the abovementioned four categories and run the OS-SLAM system. After estimating the camera trajectory of the RGB-D sensor, we evaluate the error in the estimated trajectory by comparing it with the ground truth. There are two different error metrics: the absolute trajectory error (ATE) and the relative pose error (RPE). The ATE is well suited for measuring the performance of visual SLAM systems, while the RPE is well suited for measuring the drift of visual odometry systems, such as the drift per second.

Because our purpose is to correct the trajectory for visual SLAM, we select the ATE to evaluate its effect. The absolute trajectory error is directly measured as the difference between the points of the ground truth and those along the estimated trajectory. As a preprocessing step, we associate the estimated poses with the ground-truth poses using the timestamps. Based on this association, we align the true and the estimated trajectory using the singular value decomposition. Finally, we compute the difference between each pair of poses, and output the root mean square error (RMSE) of these differences.

4.3.3. Result

After obtaining the RMSE of the ATE, we compare our results with the results obtained using several state-of-the-art systems such as ORB-SLAM2 and RGB-D SLAM, which were presented in Section 2 (related works).

4.3.4. Analysis

From **Table 4**, we can observe that the overall effect of the OS-SLAM system is better than that of other state-of-the-art systems. The data of last 2 columns are from [3] and [2] respectively, and “x” means that the data are not given by these references. We obtained the data in the ORB-SLAM2$*$ column by ourselves. The algorithm is programmed by adding the weighting process for 2D loop closures and it is applied to the dataset. We can replace the “x” in ORB-SLAM2 with the value in ORB-SLAM2$*$, which is convenient for us to compute the average error of ORB-SLAM2. Our system is based on ORB-SLAM2, and ORB-SLAM2 has a better performance than RGB-D SLAM; therefore, our system has a better performance than RGB-D SLAM at least, and we just need to compare our result with ORB-SLAM2.

First, we provide larger weights for 2D loop closures detected by ORB-SLAM2 and obtain ORB-SLAM2*, the results of which are shown in the fifth column. The normal edges in the pose graph have unit weights and

Table 4. Comparison of time required and absolute trajectory error.

Scene(from TUM public RGB-D dataset)	Time Cost (mean) [millisecond]		ATE (RMSE) [centimeter]			
	OS-SLAM (Proposed)	*ORB-SLAM2**	***OS-SLAM (Proposed)***	*ORB-SLAM2**	*ORB-SLAM 2* [3]	*RGBD-SLAM* [2]
fr1_xyz	52.9	36.7	0.91	0.91	**0.9**	1.34
fr1_room	55.6	35.7	**3.97**	6.47	x	8.4
fr1_desk	64.8	33.1	**1.47**	1.62	1.69	2.52
fr1_desk2	66.6	36.7	**2.5**	3.95	x	4.3
fr1_floor	47.6	31.2	**1.34**	1.36	2.99	3.51
fr2_pioneerslam2	24.9	23.6	**4.83**	6.35	x	x
fr2_desk	43.6	40.6	**0.86**	0.88	0.88	3.94
fr2_xyz	33.7	32.0	0.41	0.42	**0.3**	1.43
fr3_long_office	51.0	43.7	**0.96**	1.2	3.45	x
fr3_nostructure_tex_near_withloop	38.3	36.7	**1.24**	1.82	1.39	x
fr3_structure_texture_far	39.3	38.6	0.95	0.95	**0.77**	x
fr3_ structure_texture_near	43.1	35.6	1.03	**0.93**	1.58	x
Average	46.8	35.4	**1.71**	2.24	2.56	3.25

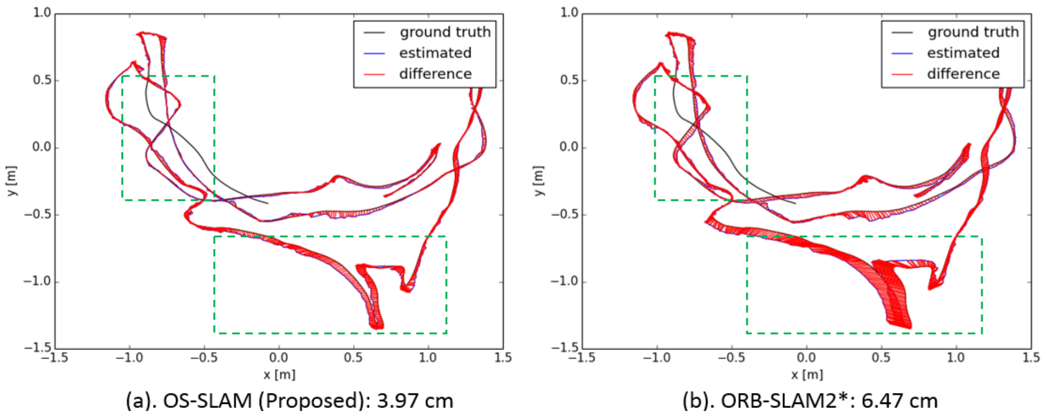

Fig. 10. Comparison of ATE (RMSE) between OS-SLAM and ORB-SLAM2∗ for the sequence *fr1_room*.

the loop edges have larger weights, for example, 100 in ORB-SLAM2*. We can observe from **Table 4** that ORB-SLAM2* has a better performance than ORB-SLAM2 in most cases. If the RMSE of the ATE is already small (for example, smaller than 1 cm), there is almost no effect of larger weights, such as in the cases of *fr1_xyz* and *fr2_desk*, or even a slightly larger error similar to the cases of *fr2_xyz* and *fr3_structure_texture_far*. If there are bad loops being detected, ORB-SLAM2* yields a bad result similar to that of *fr3_nostructure_tex_near_withloop*.

Then, we add the 3D loop closing constraint in ORB-SLAM2*. The 2D loops detected by ORB-SLAM2 must be the 3D loop candidates; thus, the proposed ORB-SHOT SLAM (or OS-SLAM) system obtains a better result by removing the bad loops similar to that of *fr3_nostr_tex_near_withloop*. We also assign loops that are both the 2D loops detected by ORB-SLAM2 and the 3D loops detected by the 3D vocabulary and the database with different weights base on their BoVW scores. After that, we obtain the OS-SLAM, the results of which are shown in the fourth column. The case of the sequences *fr1_room*, *fr2_pioneerslam2,* and *fr3_long_office* shows that the OS-SLAM system can correct the trajectories effectively. **Fig. 10** shows the comparison of ATE between OS-SLAM and ORB-SLAM2* for the sequence *fr1_room*. We can observe that the OS-SLAM system has a better correction especially at the dotted squares.

From the "time required" columns in **Table 4**, we find that the OS-SLAM system works slightly slower than the ORB-SLAM2 system. However, the OS-SLAM system runs with 21 fps (frames per second) so that it also works in real-time. Although the calculation of the 3D descriptors requires approximately 80 ms, the calculation process works after a keyframe has been detected. There are at least 10 keyframes between the new keyframe and the previous keyframe.

Furthermore, the OS-SLAM system is a multithreading program, so that the average calculation time for 3D descriptors is reduced greatly.

Except for the sequence *fr2_pioneerslam2* (owing to its lost tracking for a part of trajectory), we can observe that the shortest time required is for sequence *fr2_xyz*, which is 33.7 ms. The longest time required is for sequence *fr1_desk2*, which is 66.6 ms. The time required depends on the point cloud when we calculate the 3D descriptors. The smoother the point cloud, the higher the time

Table 5. Details of application configuration.

CPU	Intel(R) Core(TM) i7-4720HQ
Memory	16.00 GB (1600 MHz)
GPU	NVIDIA GeForce GTX 970M
Sensor	ASUS Xtion Pro Live
OS	Ubuntu Kylin 14.04 LTS
OpenCV	2.4.12 version
PCL	1.8.0 version
Qt	4.8 version

required. Therefore, we should control the RGB-D sensor at a suitable distance, e.g., 1.5 m from the indoor objects in practical application.

4.3.5. Summary

By adding the 3D loop closure constraint, the proposed OS-SLAM system improves the accuracy of loop closure detections. Moreover, the different weights for loop closures in the graph-based optimization reduce the accumulated error created by continuous motions more efficiently than those in state-of-the-art systems such as RGB-D SLAM and ORB-SLAM2. From **Table 4**, we can observe that our OS-SLAM system reduces the accumulated error by approximately 33.2% (from 2.56 cm to 1.71 cm) compared with the ORB-SLAM2 system. Although the time required increases by approximately 11.4 ms (from 35.4 ms to 46.8 ms) compared with ORB-SLAM2, our OS-SLAM system still works in real-time at 21 fps.

5. Application

In Section 4, we showed that the proposed ORB-SHOT SLAM (or OS-SLAM) has a better performance on reducing the accumulated error for continuous motions. In this section, we design an application that reconstructs the surroundings around the sensor based on the estimated trajectory.

5.1. Description

The task of this application is to reconstruct the surroundings around the sensor based on the estimated trajectory. We use OS-SLAM to obtain the estimated transformation of sensor in real time, while we make a data fusion for the point cloud by TSDF (truncated signed distance function) [30] method based on the estimated transformation. Then, we obtain the TSDF cloud and save the optimized trajectory. Finally, we make a surface reconstruction based on the TSDF cloud.

Table 5 lists the details of the application configuration, and **Fig. 11** shows the operations of our application: holding the Xtion and walking around. The interface of the application is also shown in **Fig. 11**: 1) the upper right is the window showing the trajectory correction by the proposed OS-SLAM system; 2) the lower left is the window showing the point cloud fusion; and 3) the upper left is the window showing the surface reconstruction. Notice

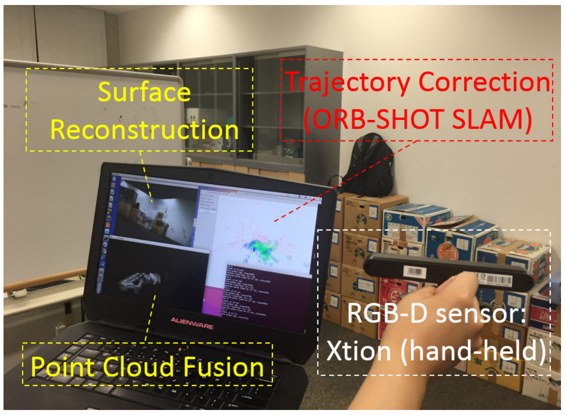
Surface Reconstruction
Trajectory Correction (ORB-SHOT SLAM)
RGB-D sensor: Xtion (hand-held)
Point Cloud Fusion

Fig. 11. Interface and operation of application.

that our software can only work at indoor environments owing to the limitations of Xtion (its valid range is from 0.8 m to 3.5 m).

5.2. Demonstrations

We demonstrate some results of point cloud fusion and surface reconstruction, which are shown in **Figs. 12–15**. The first scene (scene-1, **Fig. 12**) is our laboratory room *N253* (approximately 7×7 m^2) at Kitakyushu campus, Waseda University. The second scene (scene-2, **Fig. 13**) is the laboratory room *N253*, the laboratory room *N256*, and the corridor from *N253* and *N256* (approximately 50 m). The last scene (scene-3, **Fig. 14**) is *fr1_room* from the public TUM RGB-D dataset.

5.2.1. Data Fusion

Figures 12, 13, and **14** show the result of the data fusion. In **Fig. 12** (scene-1), from the local perspectives (a), (b), and (c), we can observe that the point cloud based on the optimized trajectory has a slightly overlapping error. In **Fig. 13** (scene-2), from two local perspectives (a) and (d), a straight corridor is illustrated. From (c) and (f), we can see a straight corridor as well as two laboratories connected with the corridor at right angles from two global perspectives. From (e), we can see the neat desks in the laboratory *N256* room. **Fig. 14** is the room scene from the TUM public dataset, and we can see that there is a slight overlapping error between objects.

5.2.2. Surface Reconstruction

Figure 15 shows the result of the surface reconstruction. **Fig. 15(a)** corresponds with **Fig. 12(c)**, and we can see the scene more clearly after the reconstruction than that of only using point clouds. **Fig. 15(b)** shows the details around the desk. Owing to the resolution limitation of the global point cloud, the local point cloud around the desk is not very clear as shown in **Fig. 14(c)**; however, we can see more details clearly after the reconstruction, particularly when we look closer to the desk as in **Fig. 15(c)**.

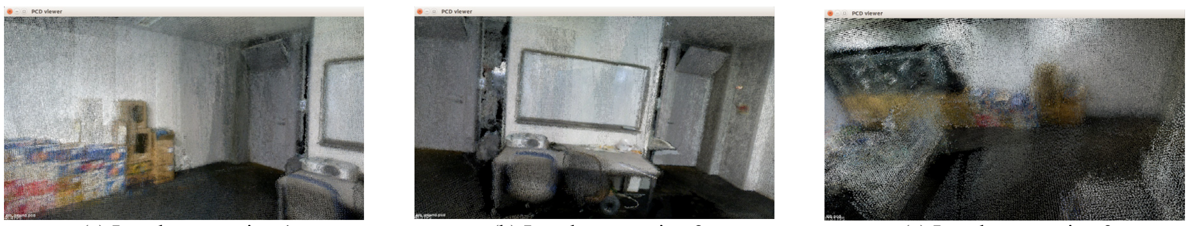
(a) Local perspective-1 (b) Local perspective-2 (c) Local perspective-3

Fig. 12. Data fusion for scene-1 (small scale, room N253).

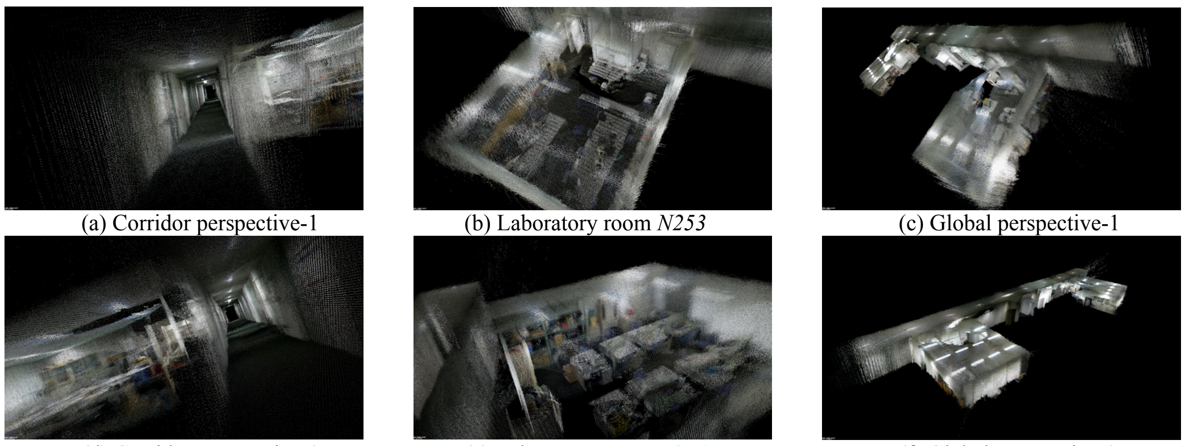
(a) Corridor perspective-1 (b) Laboratory room *N253* (c) Global perspective-1
(d) Corridor perspective-2 (e) Laboratory room *N256* (f) Global perspective-2

Fig. 13. Data fusion for scene-2 (large scale).

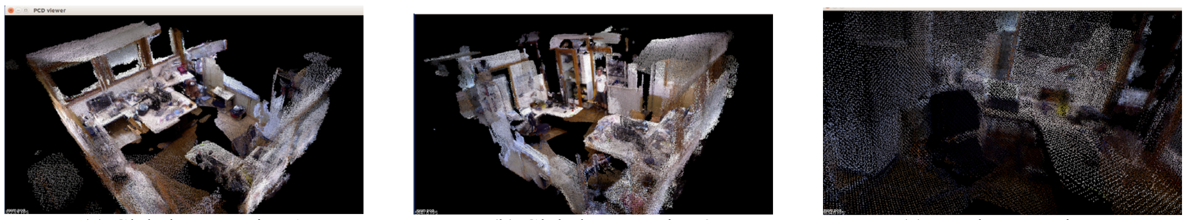
(a) Global perspective-1 (b) Global perspective-2 (c) Local perspective

Fig. 14. Data fusion for scene-3.

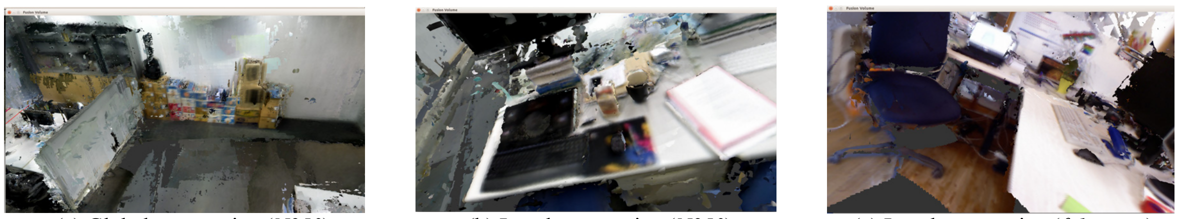
(a) Global perspective (*N253*) (b) Local perspective (*N253*) (c) Local perspective (*fr1_room*)

Fig. 15. Surface reconstruction.

Scene-2 has an extremely wide range so that we do not make the surface reconstruction for that scene because of the prescribed voxel number and the consequent low precision.

5.2.3. Summary

We implement the estimated trajectory by the proposed OS-SLAM system into the application that reconstructs the surroundings around the sensor. We also obtained a fusion for the point clouds and made a surface reconstruction for objects. The demonstrations show that our application has a good visual effect in both large-scale point cloud fusion (as shown in Section 5.2.1 with **Fig. 13**) and small-scale detailed surface reconstruction (as shown in Section 5.2.2 and comparing **Fig. 15(c)** with **Fig. 14(c)**).

6. Conclusion

In this section, we summarize the proposed method of correcting trajectory, and discuss the contribution of the proposed OS-SLAM system and future research.

6.1. Solved Problem

Here, we summarize the proposed solution to the problem, which is trajectory correction using visual SLAM.

6.1.1. 3D Vocabulary Training

The 3D vocabulary training is an offline process and the vocabulary supports the conversion from depth images to BoVW vectors. We first detect the keypoints and compute the corresponding SHOT descriptors from approximately 5,000 images provided by the TUM public RGB-D dataset. Then, we use k-means algorithm to cluster these descriptors ($k = 10$) into different visual words and generate a 3D vocabulary tree with 6 levels excluding the root (Section 3.1).

6.1.2. Loop Closing

With passage of time and extending the range of map, the error due to the movement of the sensor becomes larger and larger. To solve this problem, we detect the loop closures and use them to reduce the error based on the graph-based optimization approach. The proposed OS-SLAM method is used to detect the 3D loop closures from an incremental database and assign them different weights based on the scores of BoVW vectors (Section 3.2). For a newly detected frame, we query it in the database and obtain the loop candidates. The real loop closure satisfies the 2D constraint (adequately matched features of keypoints and 2D descriptors) as well as the 3D constraint (adequately matched features of keypoints and 3D descriptors). The higher scores mean that the loop closures are more reliable; therefore, we put more weights on these edges (loop closure) in the pose graph. After the graph-based optimization (Section 3.3), we obtain a better trajectory, which has a smaller error from the ground truth.

6.1.3. Distinctive Features

Compared with other state-of-the-art systems, the main improvement is the weight assignment for different loop closures. The results in **Table 4** in Section 4.3 indicate that overall, the proposed OS-SLAM system has a better effect for trajectory correction. If the loop closures are few or the movement of the sensor is small, our OS-SLAM can just improve the correction slightly or none at all. Otherwise, if the loop closure is after a large loop, which means that there are many keyframes between both sides of this loop – in other words, the movement of the sensor is not limited in a small space – our OS-SLAM can improve the trajectory correction effectively by suitable weights of loop closures. From the data in **Table 4**, the OS-SLAM system reduces the accumulated error by 33.2% (from 2.56 cm to 1.71 cm) compared with the ORB-SLAM2 system. Although the time required increases by 11.4 ms (from 35.4 ms to 46.8 ms) compared with ORB-SLAM2 system, the OS-SLAM system still works in real time at 21 (1000/46.8) fps. Section 5.2 also provides the intuitive results by the 3D reconstruction application on both the large-scale point cloud fusion (Section 5.2.1) and the small-scale detailed surface reconstruction (Section 5.2.2).

6.2. Contribution

Based on Section 3 (methodology part), Section 4 (experiment), and Section 6.1 (solved problem), the contributions of this study (new proposal) are the following:

1) **3D SHOT vocabulary** that supports the conversion from depth images to BoVW vectors and the query for vectors in the incremental database.
2) **Testimony** that shows that 3D descriptors work satisfactorily for loop closure detection, and the ORB-SHOT combination has the best performance for loop closure detection.
3) **ORB-SHOT SLAM** that provides a more accurate trajectory for the movement of sensor.

6.3. Future Research

There are mainly two limitations of our OS-SLAM system, which would be considered in future research.

The first limitation is on the size of the visual vocabulary. We select the images from indoor scenes to train the vocabulary, which may incur some error when the features on images from completely different scenes are gathered into different clusters (words). It means that the performance of the system would be affected by the data used for its training. To avoid this case, we suggest adding new words into the vocabulary in the online system.

The second limitation concerns the features on the images. In our system, we use the 2D corners as the feature and use the 2D-3D descriptor combination as the words in BoVW model for the loop closure detection, which means that our system has the same limitations with corners (keypoints). As we know, people recognize scenes based on an object level and not on a point level. Based on our knowledge, we know which types of objects are in the scenes when we see them. Thus, a better approach to detect a loop closure is training the machine to learn different types of objects after segmentation. Therefore, we should try deep learning approach to detect loop closures in future research.

Acknowledgements

This research is supported partially by Waseda University (2014K-6191, 2014B-352, 2015B-346, and 2016B-203), Kayamori Foundation of Informational Science Advancement (K26kenXIX-453), and SUZUKI Foundation (26-Zyo-I29), to which we would like to express our sincere gratitude.

References:

[1] A. J. Davison and I. D. Reid, "MonoSLAM: Real-time single camera SLAM," IEEE Trans. on pattern analysis and machine intelligence, Vol.29, No.6, pp. 1052-1067, 2007.

[2] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3-D mapping with an RGB-D camera," IEEE Trans. on Robotics, Vol.30, No.1, pp. 177-187, 2014.

[3] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "Orb-slam: a versatile and accurate monocular slam system," IEEE Trans. on Robotics, Vol.31, No.5, pp. 1147-1163, 2015.

[4] H. Hagiwara, Y. Touma, K. Asami, and M. Komori, "FPGA-Based Stereo Vision System Using Gradient Feature Correspondence," J. of Robotics and Mechatronics, Vol.27, No.6, pp. 681-690, 2015.

[5] T. Suzuki, Y. Amano, T. Hashizume, and S. Suzuki, "3D terrain reconstruction by small Unmanned Aerial Vehicle using SIFT-based monocular SLAM," J. of Robotics and Mechatronics, Vol.23, No.2, pp. 292-301, 2011.

[6] A. Sujiwo, T. Ando, E. Takeuchi, Y. Ninomiya, and M. Edahiro, "Monocular Vision-Based Localization Using ORB-SLAM with LIDAR-Aided Mapping in Real-World Robot Challenge," J. of Robotics and Mechatronics, Vol.28, No.4, pp. 479-490, 2016.

[7] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," 2011 IEEE Int. Conf. on computer vision, 2011.

[8] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Epnp: An accurate $o(n)$ solution to the pnp problem," Int. J. of computer vision, Vol.81, No.2, pp. 155-166, 2009.

[9] D. Galvez-Lopez, and J. D. Tardos, "Real-time loop detection with bags of binary words," 2011 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 2011.

[10] S. Aoyagi, A. Kohama, Y. Inaura, M. Suzuki, and T. Takahashi, "Image-Searching for Office Equipment Using Bag-of-Keypoints and AdaBoost," J. of Robotics and Mechatronics, Vol.23, No.6, pp. 1080-1090, 2011.

[11] R. Mur-Artal and J. D. Tardós, "Fast relocalisation and loop closing in keyframe-based SLAM," 2014 IEEE Int. Conf. on Robotics and Automation (ICRA), 2014.

[12] D. G. Lowe, "Object recognition from local scale-invariant features," The Proc. of the seventh IEEE Int. Conf. on Computer Vision 1999, Vol.2, 1999.

[13] H. Bay, T. Tuytelaars, and L. V. Gool, "Surf: Speeded up robust features," European Conf. on computer vision, Springer Berlin Heidelberg, 2006.

[14] F. Endres and N. Engelhard, "An evaluation of the RGB-D SLAM system," 2012 IEEE Int. Conf. on Robotics and Automation (ICRA), 2012.

[15] S. A. Scherer, A. Kloss, and A. Zell, "Loop closure detection using depth images," 2013 European Conf. on Mobile Robots (ECMR), 2013.

[16] F. Tombari, S. Salti, and L. D. Stefano, "Unique signatures of histograms for local surface description," European Conf. on computer vision, Springer Berlin Heidelberg, 2010.

[17] G. T. Flitton, T. P. Breckon, and N. M. Bouallagu, "Object Recognition using 3D SIFT in Complex CT Volumes," British Machine Vision Conference (BMVC), 2010.

[18] I. Sipiran and B. Bustos, "Harris 3D: a robust extension of the Harris operator for interest point detection on 3D meshes," The Visual Computer, Vol.27, No.11, pp. 963-976, 2011.

[19] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard, "NARF: 3D range image features for object recognition," Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), Vol.44, 2010.

[20] E. Rosten and T. Drummond, "Machine learning for high speed corner detection" 9th European Conf. on Computer Vision, Vol.1, pp. 430-443, 2006.

[21] E. Rosten, R. Porter, and T. Drummond, "Faster and better: a machine learning approach to corner detection" IEEE Trans. Pattern Analysis and Machine Intelligence, Vol.32, pp. 105-119, 2010.

[22] R. B. Rusu, Z. C. Marton, N. Blodow, and M. Beetz, "Persistent point feature histograms for 3D point clouds," Proc. 10th Int. Conf. Intel. Autonomous Syst. (IAS-10), Baden-Baden, Germany, 2008.

[23] R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu, "Fast 3d recognition and pose using the viewpoint feature histogram," 2010 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), 2010.

[24] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (FPFH) for 3D registration," IEEE Int. Conf. on Robotics and Automation (ICRA'09), 2009.

[25] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," Proc. of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2007.

[26] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," J. of the society for Industrial and Applied Mathematics, Vol.11, No.2, pp. 431-441, 1963.

[27] J. J. Moré, "The Levenberg-Marquardt algorithm: implementation and theory," Numerical analysis, Springer Berlin Heidelberg, pp. 105-116, 1978.

[28] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A general framework for graph optimization," 2011 IEEE Int. Conf. on Robotics and Automation (ICRA), 2011.

[29] D. M. W. Powers, "Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness and Correlation," Int. J. of Machine Learning Technology, Vol.2, No.1, pp. 37-63, 2011.

[30] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," 2011 10th IEEE Int. symposium on Mixed and augmented reality (ISMAR), 2011.

Name:
Zheng Chai

Affiliation:
Graduate School of Information, Production and Systems (IPS), Waseda University

Address:
2-7 Hibikino, Wakamatsu-ku, Kitakyushu 808-0135, Japan

Brief Biographical History:
2011-2014 Sichuan University
2014-2016 Waseda University
2016- IDEALSEE (Chengdu, Sichuan)

Main Works:
• augmented reality
• visual SLAM

Name:
Takafumi Matsumaru

Affiliation:
Professor, Graduate School of Information, Production and Systems (IPS), Waseda University

Address:
2-7 Hibikino, Wakamatsu-ku, Kitakyushu 808-0135, Japan

Brief Biographical History:
1987-1999 Toshiba Corporation
1999-2010 Shizuoka University
2010- Waseda University

Main Works:
• bio-robotics
• human-mechatronics

Membership in Academic Societies:
• Society of Biomechanisms Japan (SOBIM)
• The Robotics Society of Japan (RSJ)
• The Society of Instrument and Control Engineers (SICE)
• The Japanese Society of Mechanical Engineers (JSME) etc.