Computers & Graphics (2018)

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Distributed monocular visual SLAM as a basis for a collaborative augmented reality framework

Ruwan Egodagamage*, Mihran Tuceryan[1]

723 W Michigan St Indianapolis, IN 46202, USA

ARTICLE INFO

Article history:
Received June 19, 2018

Keywords: Monocular SLAM, Distributed SLAM, Collaborative AR

ABSTRACT

Visual Simultaneous Localization and Mapping (SLAM) has been used for markerless tracking in augmented reality applications. Distributed SLAM helps multiple agents to collaboratively explore and build a global map of the environment while estimating their locations in it. One of the main challenges in distributed SLAM is to identify local map overlaps of these agents, especially when their initial relative positions are not known. We developed a collaborative AR framework with freely moving agents having no knowledge of their initial relative positions. Each agent in our framework uses a camera as the only input device for its SLAM process. Furthermore, the framework identifies map overlaps of agents using an appearance-based method. We also proposed a quality measure to determine the best keypoint detector/descriptor combination for our framework.

1. Introduction

Markerless tracking has been a goal of many augmented reality applications, and the Simultaneous Localization and Mapping (SLAM) has been a robust framework to accomplish this. The robotics community defines the SLAM problem as an agent creating a map of an unknown environment using sensors while localizing itself in it. To localize the agent properly, an accurate map is required. To generate an accurate map, localization has to be done properly. This means that localization and mapping need to be done simultaneously to benefit each other.

Inexpensive, ubiquitous mobile agents with cameras and image processing tools made them a popular choice of a sensor for SLAM. Most Visual SLAM approaches relied on detecting features and generating sparse maps using them. More recent solutions with direct featureless methods [1] generate semi-dense maps of the environment. Dense maps provide many benefits over sparse maps including, better agent interactions with the environment or objects, better scene interaction for augmented reality applications, and better object recognition with enhanced data. However, in practice, direct, featureless methods require significant overlaps between key frames, with narrower baselines. This adds a limit to the movement of the camera. Furthermore, the direct method alone could not handle large loop closures.

Many researchers investigated how to use multiple agents to perform SLAM: called collaborative or distributed SLAM. Distributed SLAM increases the robustness of the SLAM process and makes it less vulnerable to catastrophic failures. Challenges in distributed SLAM are computing map overlaps and sharing information between agents with limited communication bandwidth.

We developed a collaborative augmented reality framework based on distributed SLAM. Agents in our framework do not have any prior knowledge of their relative positions. Each

*Corresponding author
e-mail: janapriya@gmail.com (Ruwan Egodagamage), tuceryan@iu.edu (Mihran Tuceryan)
[1]Professor, Department of Computer and Information Science

This is the author's manuscript of the article published in final edited form as:
Egodagamage, R., & Tuceryan, M. (2018). Distributed monocular visual SLAM as a basis for a collaborative augmented reality framework. Computers & Graphics, 71, 113–123. https://doi.org/10.1016/j.cag.2018.01.002

agent generates a local semi-dense map utilizing direct featureless SLAM approach. The framework uses image features in keyframes to determine map overlaps between agents. We performed a comprehensive analysis on state-of-the-art keypoint detector/descriptor combinations to improve the performance of our system reported in [2] by defining a quality measure to find the optimal combination. We created the publicly available DIST-Mono distributed monocular visual SLAM dataset to evaluate our system. Furthermore we developed a proof-of-concept augmented reality application to demonstrate the potential of our framework.

2. Related Work

In a seminal paper, Smith et al. [3] introduced an Extended Kalman Filter (EKF) based solution for the SLAM problem (EKF-SLAM). The EKF incrementally estimates the posterior distribution over agent pose and landmark positions. The covariance matrix grows with the number of landmarks. Even a single landmark observation leads to an update of the covariance matrix, limiting the number of landmarks EKF-SLAM could handle due to the excessive computational cost. Furthermore, EKF-SLAM has Gaussian noise assumptions. A Monte Carlo Sampling (particle filter) based approach by Montemerlo et al. [4] named FastSLAM, addressed the above limitations and supported non-linear process models and non-Gaussian pose distributions.

Davison et al. [5] introduced Monocular Visual SLAM (MonoSLAM); a method of capturing the path of a freely moving camera while generating a sparse map. The generated sparse map consisted of image patches as features. They combined EKF-SLAM and Particle Filtering (PF) for estimation and feature initialization respectively. Klein et al. in [6] presented, Parallel Tracking and Mapping (PTAM), one of the most significant solutions for visual SLAM. This robust SLAM solution mainly focused on accurate and fast mapping in a similar environment to MonoSLAM. Its implementation decoupled mapping and localization, into two threads. The front-end thread only performs pose estimation and feature tracking while the back-end thread performed mapping and everything else, such as feature initialization and removing unnecessary keyframes. Similar to MonoSLAM, a set of sparse point features represented the map. RANSAC [7] and 5 point algorithm [8] initialized the system. A global Bundle Adjustment (BA) [9] with Levenberg-Marquardt optimization [8] adjusted the pose of all keyframes. Furthermore, a local BA changed the pose of a subset of keyframes allowing a reasonable rate of exploration.

BA worked well for offline Structure from Motion (SfM). Even though BA is relatively computationally expensive, PTAM and other researchers recently adopted BA for many real-time monocular visual SLAM solutions. Strasdat's analysis in [10] showed increasing the number of image features acquired per frame was more beneficial than incorporating information from increased number of closely placed camera frames. They argued that the former increases the accuracy of the motion estimation and a better map estimation for a given computational budget. Their analysis, hence, favored bundle adjustment techniques over incremental methods for accurate monocular visual SLAM. Moreover, BA helps to increase the number of features on the map, leading to denser maps.

The work by DTAM by Newcombe et al. [11] and LSD-SLAM by Engel et al. [1] utilized image pixel intensities directly instead of features for SLAM. Their systems generated dense or semi-dense maps of the environment. Furthermore, these direct methods were more robust to motion blur of images.

2.1. Distributed SLAM

A naïve brute-force method could communicate all sensor observations and map updates between agents in a distributed SLAM system. However, computational resources and communication bandwidth of an agent are limited. Furthermore, the distributed network is subject to failures of nodes and links. Therefore, to overcome these challenges, a proper and intelligent approach is required for a distributed SLAM system.

If agents know either their relative locations or map overlaps they can easily generate a unique, globally consistent map. For example, in [12], relative locations of the agents were provided by global positioning sensors (GPS). It was also relatively easier to determine map overlaps if the relative initial poses of all agents are known. For example, Paull et al. in [13] initialized agents with known GPS location information.

The problem becomes difficult if the relative locations of agents are unknown. In some contributions, agents continued building local sub-maps until they meet each other. Howard et al. [14] proposed a method where each agent could detect other agents. The agents use these coincidental encounters to find their relative locations. Dieter Fox et al. in [15] presented a method where each agent actively sought other agents in the environment to find their relative locations.

In our distributed SLAM framework, each agent adapted LSD-SLAM[1] in itself to generate a local semi-dense map. Compared to earlier approaches, our framework does not expect agents to start from the same location, meet each other, stay in the vicinity of an another agent, etc. The framework computes map overlaps between agents based on an appearance based method.

When we first reported our framework in [2], for our appearance based map overlap detection method, we used SURF keypoint detectors [16] and SIFT descriptors [17]. In this paper, we introduce a quality measure to find the best keypoint detector/descriptor for a distributed monocular visual SLAM system. Based on that, we determined ORB keypoint detectors[18] and BRISK descriptors[19] combination as the best candidate for our work and achieved better results.

We used the experimental framework for distributed SLAM that we introduced in [20], as the architecture of the distributed framework. Compared to earlier approaches, our system does not rely on a centralized server, instead we use two kinds of agents called exploring and monitoring agents in a distributed network.

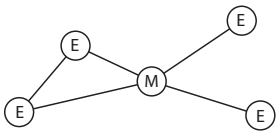


Fig. 1: The network of nodes, exploring nodes (E) are connected to monitoring node (M). Some exploring nodes are connected with each other

3. System Overview

Our framework consists of two types of distributed nodes; *exploring node* and *monitoring node*. These nodes are deployed on different physical machines and given a globally unique identifier. The framework has one monitoring node and multiple exploring nodes at any given time. The nodes use communication channels to pass messages between each other.

We use the Robot Operating System (ROS) [21] infrastructure for our framework. ROS includes *nodes* that are responsible for performing computations. We implemented exploring and monitoring nodes as ROS nodes. ROS also provides named communication busses called *topics* to pass messages between ROS nodes. We use ROS topics as our peer-to-peer communication channels between nodes.

As the name suggests, exploring nodes are responsible for generating a local map of the environment. They periodically send their map to the monitoring node. The monitoring node continuously monitors these map updates to determine potential map overlaps. If it finds an overlap between a pair of exploring nodes, it sends a command to connect those nodes and merge their maps. Figure 1 shows a possible configuration of nodes. As illustrated, exploring nodes are always connected to the monitoring node. If there is a map overlap, two exploring nodes can also be connected to each other. Sections 4 and 5 explain the functionality of exploring node and monitoring node respectively.

We developed a multi-user AR application to demonstrate the collaborative AR potential of our framework. We added an AR window to each exploring node, allowing users to interact in the same environment. This is explained in more detail in section 9.

4. Exploring Node

Each exploring node performs semi-dense visual SLAM based on the work by [22]. It uses a single camera as the only input device. It maintains a list of key frames and a pose graph to represent its local map.

4.1. Key Frames

The i^{th} **key frame**, $\mathcal{K}_i$ consists of an **absolute pose** $\boldsymbol{\xi}_{Wi} \in \mathbb{R}^7$, an image I_i, a map containing z coordinate reciprocals corresponding to non-negligible intensity gradient pixels D_i (an inverse depth map), an inverse depth variance map V_i and a list of features F_i. Figure 2 contains a visual representation of $\mathcal{K}_i$ of two key frames. Features of $\mathcal{K}_i$ are computed when we introduce $\mathcal{K}_i$ into the pose graph. In $\mathcal{K}_i$, i corresponds to a 32 bit globally unique identifier. We combine the globally unique node identifier and a locally unique frame identifier to generate a globally unique key frame identifier as shown in Figure 3.

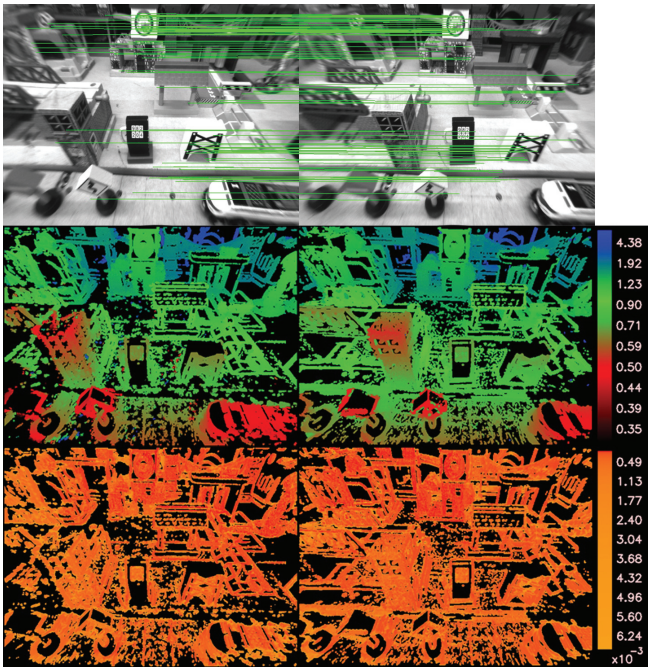


Fig. 2: We show the matched features between key frames $\mathcal{K}_i$ and $\mathcal{K}_j$ superimposed on the images I_i and I_j (top). We also show the pseudo-color encoded D_i and D_j (middle) and pseudo-color encoded V_i and V_j (bottom).

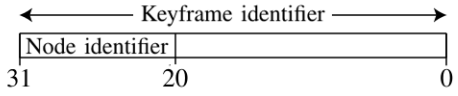


Fig. 3: Globally unique keyframe identifier based on node identifier

4.2. Pose Graph

Pose graph edges ε_{ji} contain similarity transformations $\boldsymbol{\xi}_{ji}$ and $\boldsymbol{\Sigma}_{ji}$ constraints. Here $\boldsymbol{\xi}_{ji} \in \mathbb{R}^7$ are the relative pose transformations and $\boldsymbol{\Sigma}_{ji}$ are the corresponding covariance matrices between i^{th} and j^{th} key frames.

Both absolute pose $\boldsymbol{\xi}_{Wi}$ and similarity transformation $\boldsymbol{\xi}_{ji}$ are encoded with a translation (three components) and orientation with scale using a quaternion (four components).

4.3. SLAM Process

The SLAM process simultaneously tracks the camera against the current key frame $\mathcal{K}_i$ and improves its D_i and V_i based on its new observations. Once the camera deviates significantly from the $\mathcal{K}_i$, either a new key frame is created or, if available, an existing key frame is selected from the map. Next, if a new key frame is created, the previous key frame used for tracking is inserted into the pose graph. The pose graph is continuously optimized in the background. More information on the LSD-SLAM process is found in [1].

4.4. Features

We used ORB [18] features and BRISK [19] descriptors in our framework. For every salient feature in F_i, the corresponding 3D location x_p and the descriptor d_p are computed. Our

choice of features did not adversely affect on the real-time performance, given we only compute features in key frames. We have done a comprehensive analysis on state-of-the-art keypoint detectors and descriptors to select the best combination for our distributed framework as explained in Section 6.

4.5. Communication with the Monitoring node

Between exploring and monitoring nodes, there are three communication channels. An exploring node sends its new key frame $\mathcal{K}_i$ along with features F_i through the *key frames channel*. After every pose graph optimization, the pose graph is sent through the *pose graph channel*. Exploring nodes receive commands through the *commands channel*.

Upon receiving a loop closure command from Monitoring node with $\boldsymbol{\xi}_{ji}$, the exploring node checks whether there is an existing edge ε_{ji} between $\mathcal{K}_i$ and $\mathcal{K}_j$ vertices of the pose graph. If an existing edge is found, it discards the loop closure command. Otherwise, it inserts the new edge and completes the process by performing another iteration of pose graph optimization.

4.6. Communication with other Exploring nodes

As shown in Figure 1, two overlapping exploring nodes can communicate with each other. Map overlap key frame correspondences are provided by the Monitoring node. Once the connection is made, each exploring node sends its map to its counterpart through *map merge channel*. Once the map is received, the key frame correspondences are directly transformed into new constraints between pose graphs of e_i and e_j.

Figure 4 shows how e_i and e_j were generating their own maps before merging. Right hand side map of Figure 5 shows a resulting merged map of two exploring nodes. Once map merging is complete, each exploring node listens to its counterpart for new key frames and the pose graph, to incrementally update its map.

4.7. Modules of the exploring node

Figure 6 shows modules of the distributed framework and the communications between nodes. The Exploring node consists of five main modules: input stream, tracking, mapping, constraint search and optimization modules. Each of these modules runs in its own thread.

The *input stream* module accepts all incoming messages including image frames, key frames, pose graph, map, and commands. All image frames are transferred to the tracking module. Key frames, pose graph and map are transferred to the optimization module so that they can be merged into the map before an optimization iteration. Commands are processed in the input stream module itself.

The *tracking* module accepts the new frame from input stream module and tracks it against the current key frame. If the current key frame can no longer be used to track the current frame, a new key frame is created. The old key frame will be added to the map by the *mapping* module. The *constraint search* module is used to recover from tracking failures. The *optimization* module continuously optimizes the pose graph in the background.

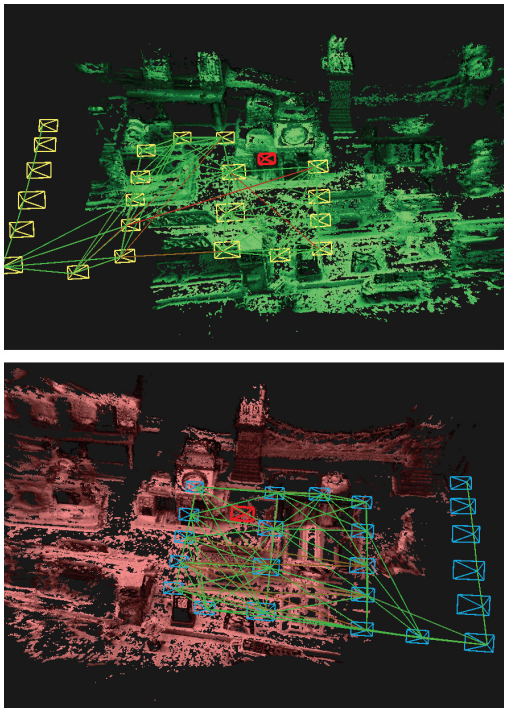


Fig. 4: Map generation process of two exploring nodes. Each exploring node has its own coordinate system. Relative transformations between coordinate systems are initially not known.

5. Monitoring Node

Exploring nodes of our distributed framework do not know their relative poses at the beginning. Monitoring Node's Map overlap detection module is responsible for detecting and computing corresponding relative pose between nodes. It also detects loop closure of each exploring node.

Monitoring node maintains an N number of key frame databases DB_i. Here N equals to the number of exploring nodes in the framework. All incoming key frames $\mathcal{K}_i$, are matched against all these key frame databases. The matching takes place in parallel in M number of threads. The number M ($< N$) is configured based on available system resources.

5.1. Key frame database

Each key frame database consists of key frames of one exploring node. Each incoming key frame $\mathcal{K}_i$ is matched against the entries in the database using FLANN[23] feature matching method. If there are more than 10 matches with another key frame $\mathcal{K}_j$, it is concluded that there is an overlap between key frames $\mathcal{K}_i$ and $\mathcal{K}_j$. If these key frames belong to same exploring node, a loop closure, is found. Otherwise, the result is submitted to the *Fusion Graph*.

5.1.1. Fusion graph

All available exploring nodes are represented as vertices in the fusion graph as shown in Figure 7. Assume there is an overlap between key frames $\mathcal{K}_r$ and $\mathcal{K}_s$ and $\mathcal{K}_r \in e_i^{\mathcal{K}}$ and $\mathcal{K}_s \in e_j^{\mathcal{K}}$, where $e_i^{\mathcal{K}}$ represent key frames in i^{th} exploring node. Then, the fusion graph contains an edge between e_i and e_j. The number of features matched between e_i and e_j are represented using

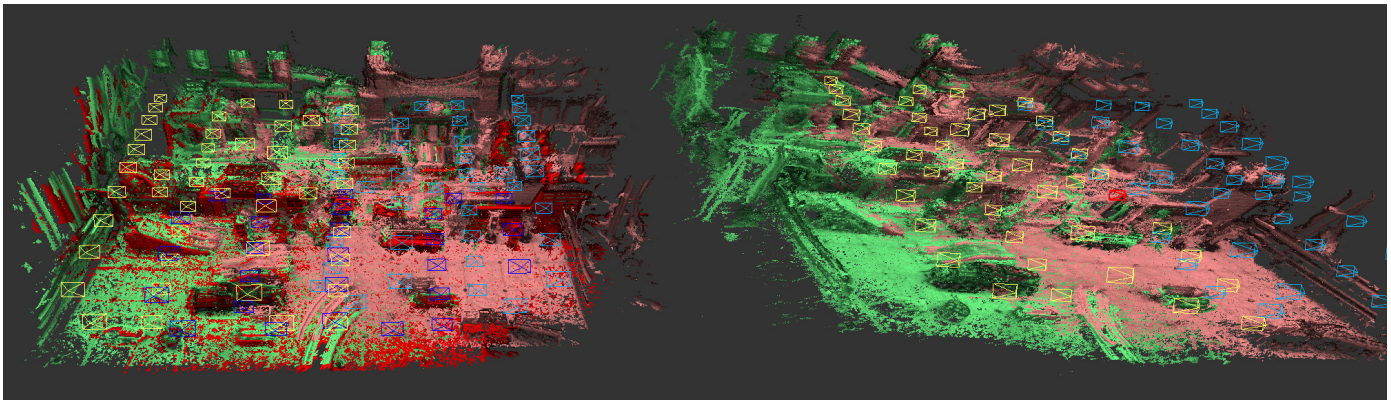


Fig. 5: Resultant maps of two exploration nodes after map merging process. In exploring node on the left, three maps are merged. In exploring node on the right, two maps are merged. It's map and key frames are shown in green and yellow respectively. The map and key frames received from the other node are shown in pink and blue, respectively. Constraints of the pose graph are not shown here to avoid too much clutter in the figure.

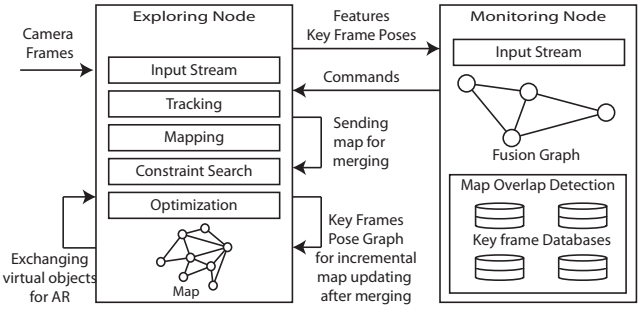


Fig. 6: The distributed framework. In the figure, the arrows looping back to the exploring node rectangle represent communication between two exploring nodes.

c_{ij} as shown in Figure 7. Note that the edge between e_i and e_j could represent matching features between many key frame pairs. The fusion graph is used by the map overlap detectic module as described in more detail in Section 6.

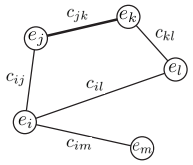


Fig. 7: The fusion graph showing exploring nodes (e_i) and the number of matc ing features (c_{ij}) as the weight of each edge. In this example, c_{jk} is higher th other edges (indicated by the thicker edge), so e_j and e_k is merged first.

5.1.2. Communication with Exploring nodes

When the monitoring node detects a map overlap betwe exploring nodes e_i and e_j, it issues a merge command throug the *commands channel* to both of them. The command contai the relative pose $\boldsymbol{\xi}_{ji}$ between two nodes. The command al contains the map overlap key frame correspondences used to compute the relative pose between e_i and e_j. Similarly, a *loop closure* command is issued to an exploring node e_s, when both overlapping key frames $\mathcal{K}_i$ and $\mathcal{K}_j$ belong to e_s. Fusion graph does not look for map overlaps between nodes that are already found overlapping. This prevents issuing merge command to e_i and e_j again.

5.1.3. Modules of the monitoring node

As shown in Figure 6, the monitoring node has three main modules. The *input stream* module is receiving key frames and pose graphs from exploring nodes. These key frames are submitted to the *map overlap detection* module which processes these key frames against multiple key frame databases in parallel. The *fusion graph* is used to prioritize map merging of exploring nodes.

6. Map Overlap Detection

6.1. Determining overlap between two maps

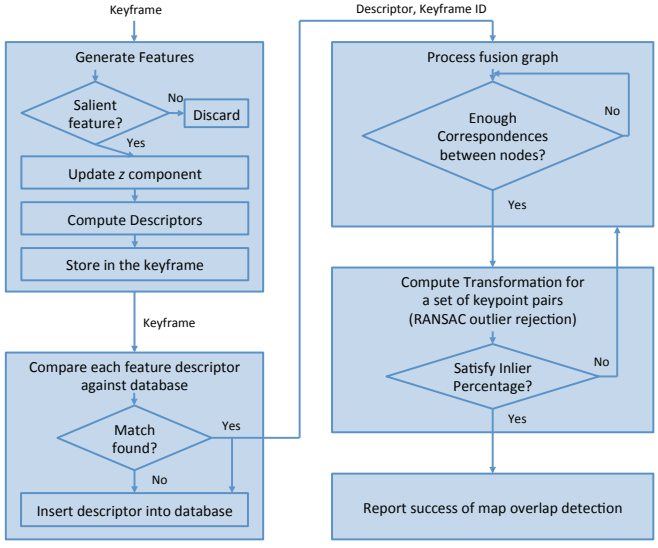


Fig. 8: Determining overlap between two maps

Figure 8 is a flowchart that describes how overlap between two maps are determined. As discussed earlier, the maps used

in our framework is represented using a set of keyframes and a pose graph. The i^{th} keyframe, $\mathcal{K}_i$ consists of an absolute pose $\boldsymbol{\xi}_{Wi}$, an image I_i, an inverse depth map D_i, an inverse depth variance map V_i and a list of features F_i. Each feature in F_i is filtered for its $V_i(x_p)$ to determine its saliency, where x_p is the location of the feature.

The pth feature in $\mathcal{K}_i$ should satisfy,

$$V_i(x_p) < tD_i(x_p)^2 \tag{1}$$

Where t is a threshold computed empirically. We experimented with different values for t and found 0.001 to be a good value with satisfactory map overlap detections. The z component of the x_p is populated using $D_i(x_p)$. Then we compute the descriptor d_p for each salient feature and store them inside the keyframe.

Next, we look for matches for each descriptor in a descriptor database. The aforementioned database is built incrementally from all salient features the framework has encountered so far. Each entry in the database has information about the descriptor and the identifier of the keyframe to which it belongs. As described earlier the keyframe identifier encodes information about the map to which the keyframe belongs. If a match is found, it is reported to the fusion graph.

An edge of the fusion graph contains all pairs of matching keypoints across the two nodes corresponding to two different maps. If an edge of the fution graph having feature match count c_{ij} satisfies,

$$\max(c_{ij}) > m \tag{2}$$

then the monitoring node concludes that a map overlap exists between exploring nodes e_i and e_j. The value for threshold m was calculated empirically.

These keypoint pairs that belong to an edge of the fusion graph are shown in Figure 9. As shown in the figure, the keypoint pairs may belong to many keyframes.

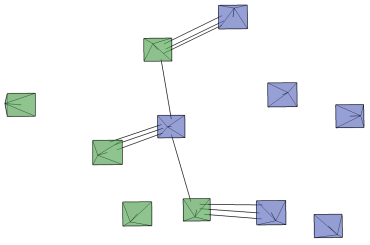


Fig. 9: Matching keypoint pairs across two sets of keyframes belong to two different maps

To calculate the map overlap all 3D points are converted into the map coordinate system using Equation 3.

$$x'_p = \boldsymbol{T}_{Wi} x_p \tag{3}$$

where $\boldsymbol{T}_{Wi}$ is the corresponding similarity transformation matrix of $\boldsymbol{\xi}_{Wi}$.

In our RANSAC implementation, four keypoint pairs are randomly selected and we use Horn's closed form solution [24] to compute rigid body transformation $\boldsymbol{\xi}_{W_r W_s}$ between coordinate systems of r and s maps. Next, we compute the inlier percentage considering all keypoint pairs for the current rigid body transformation. If the inlier percentage is greater than a threshold p, we conclude that there is a map overlap and we refine the transformation by computing the transformation using all inlier keypoint pairs. During map merging, maps exchange keyframes with each other. For example, absolute pose of all the keyframes from the map s will be updated to the coordinate system of r using following equation. Next, these keyframes are inserted into the map of r.

$$\boldsymbol{\xi}_{W_r i} = \boldsymbol{\xi}_{W_r W_s} \boldsymbol{\xi}_{W_s i} \tag{4}$$

Given that we do not limit ourselves to only two keyframes and we use two groups of keyframes, we minimize feature occlusion issues.

With our map overlap detection in place, we wanted to select the best feature-detector/descriptor combination that yields the best results. We have done an experimental analysis on the state-of-the-art feature detectors and descriptors as described in the following sections. We considered several factors including precision of matching, keypoint position accuracy, size of the descriptor, etc. As a result of this analysis we use ORB[18] feature detector and BRISK[19] descriptor in our framework. We computed features only for the keyframes, the added computational cost that resulted did not adversely affect the real-time performance.

6.2. Image Features

In various computer vision applications like image registration, classification, object recognition and 3D reconstruction, feature detection, description and matching play a pivotal role. Especially in many SLAM approaches, it has been used to detect loop closures or to recover from tracking failures by recognizing a place that is revisited. In our distributed framework, we use ORB features to detect map overlaps across multiple agents.

According to Wikipedia[25], an image feature can be described as a piece of information from an image that can be used to solve a computational task in the context of computer vision or image processing. Notably, in different images of the same scene, we would like to detect the same feature repeatedly, irrespective of the difference in viewing angle, zoom level or lighting conditions. Other desired properties of features include localization accuracy, robustness, and efficiency of computing them. The process starts by first detecting and localizing features called keypoints of an image. These may correspond to a corner, blob or region. Once identified, the image properties around that keypoint (feature region) are described using a robust feature vector or descriptor.

We performed a comprehensive analysis of different combinations of state-of-the-art feature detectors and feature descriptors for the task of matching. We propose a quality measure to select a suitable feature descriptor combination to use for the map overlap detection in the distributed framework.

6.3. State-of-the-art detectors and descriptors

We considered eight state-of-the-art feature detectors for our analysis, namely, Harris[26], Shi & Tomasi(GFTT) [27],

FAST[28][29], SIFT [17], SURF [16], BRISK[19], ORB[18], and MSER[30]. The Table 1 summarizes the detectors considered in our study. Even though the original MSER detector does not use scale space as do the other methods, it detects both smaller and larger blobs. Work by Forssen et al.[31] provided a scale-invariant version of the MSER detector and a descriptor that works in scale space.

Table 1: Keypoint detectors

Name	Scale Invariance	Type
HARRIS	No	Corner
GFTT	No	Corner
FAST	No	Corner
SIFT	Yes	Corner
SURF	Yes	Corner
ORB	Yes	Corner
BRISK	Yes	Corner
MSER	Yes	Blob

To create a description of a keypoint that is invariant to transformation and noise is a challenging task. SIFT[17] is one of the most popular descriptor, which performs well in most applications. SURF[16] is another descriptor with comparable performance to SIFT but has much faster computational time. Each of these descriptors contains a sequence of floating point numbers that are matched using Euclidean distance. But, this has a high computational cost.

Alternative descriptors that use bit sequences as descriptors have been proposed primarily to be used on mobile platforms with limited resources. The main advantage of these binary descriptors is the fact that feature matching could be performed faster just by computing the Hamming distance. Recent studies show binary descriptors are comparable in their performance with their floating point counterparts. All descriptors we used in our analysis are given below with a brief introduction to their operation.

The Table 2 provides a summary of descriptors considered in our study. Relatively new binary descriptors like BRISK[19], FREAK[32], and ORB are smaller in size and faster in detecting matches.

Table 2: Keypoint descriptors

Name	Type	Elements	Size
SIFT	Float	128	512 Bytes
SURF	Float	64	256 Bytes
BRISK	Binary	512	64 Bytes
FREAK	Binary	512	64 Bytes
ORB	Binary	256	32 Bytes

6.4. Datasets

For our study and the analysis described below, we used the Oxford affine covariant regions dataset [33]. The dataset contains eight sets of images evaluating five different imaging conditions, namely viewpoint, scale, blur, illumination and JPEG compression. Each image set contains a reference image and five other images that are related by homographies to the reference image. All five homography matrices are provided by the dataset. In our study, we focused mainly on a change in viewpoint, scale, blur, and illumination, hence the JPEG compression dataset was not considered.

6.5. Analysis

We then performed feature detection, description, and matching between the reference image and the other images of the Oxford datasets. Homography matrices are used as the ground truth to compute the precision of matching. A match is considered a good match only when the detected corresponding point is within the 5-pixel radius from the homography-estimated point position.

Figure 10 shows the overall precision result for different keypoint detector and descriptor combinations. The size of the marker corresponds to the number of correct matches found. The color represents the time it took to detect and describe all features.

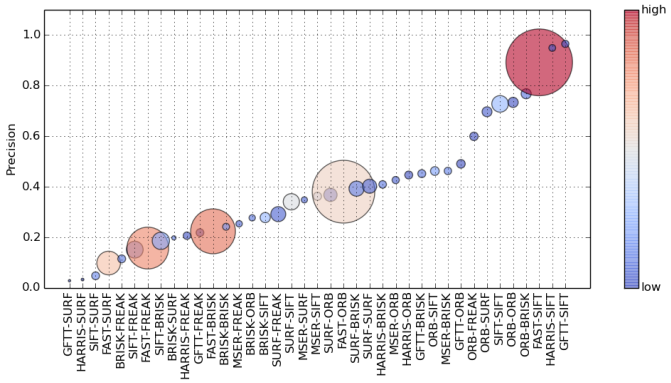
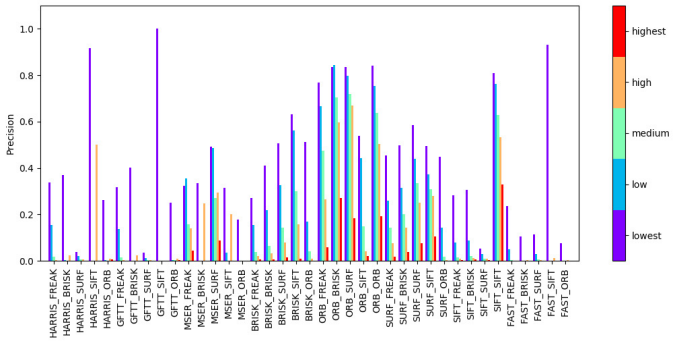
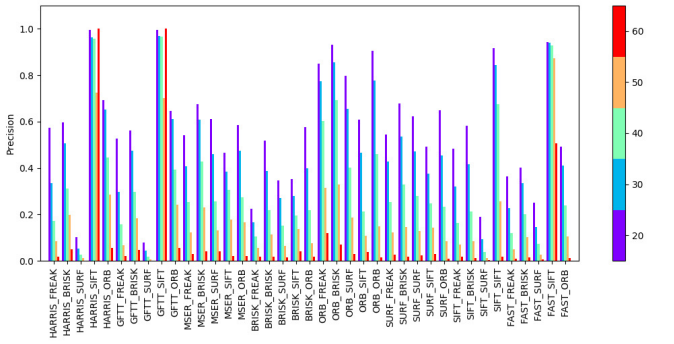


Fig. 10: Precision vs Detector-Descriptor Combinations. Number of keypoints generated are encoded in size of the marker and time to generate keypoints and descriptors are encoded in color.

To investigate how each keypoint detector and descriptor combination behaves in different imaging conditions, we computed the precision for all sets of images. We report the result of *bark* and *boat* image sets that contain varying scale in Figure 11a. The scale changes are categorized as lowest, low, medium, high and highest scale groups. From the chart, we can see the combinations (ORB, BRISK), (ORB, SURF), (SIFT, SIFT) and (ORB, ORB) perform better than the rest, regarding the overall precision of all different scale values. Similarly, Figure 11b contains the result of image sets, *graf* and *wall* which correspond to viewpoint changes. The result for imaging condition blur, the image sets *bikes* and *trees* are shown in Figure 12a. Similarly, Figure 12b shows the result of *leuven* image set, which has images with changing illumination. As expected, the precision decreases when we increasingly vary the imaging condition.

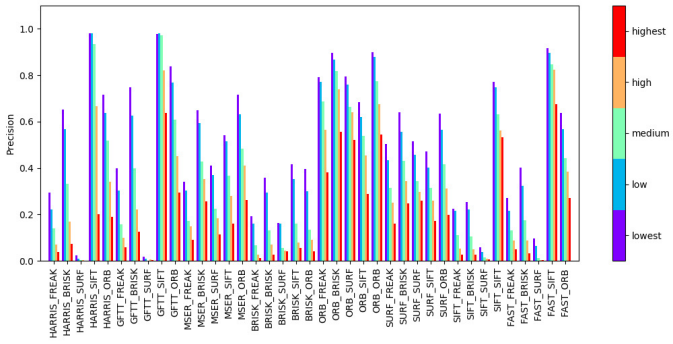


(a) Changing scale using camera zoom

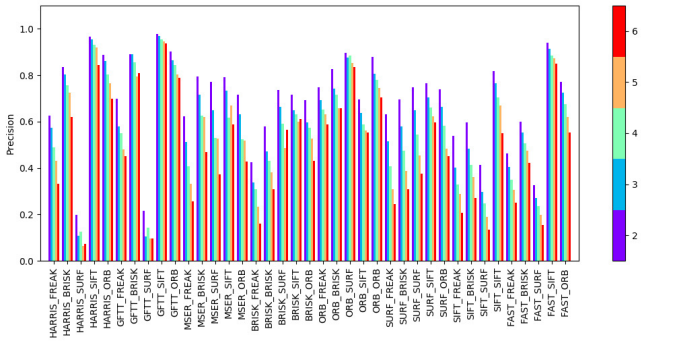


(b) Changing viewing angle of camera

Fig. 11: Precision vs Detector-Descriptor Combinations.



(a) Changing blur using camera focus



(b) Changing illumination using camera aperture.

Fig. 12: Precision vs Detector-Descriptor Combinations.

6.5.1. *Factors for finding the best keypoint detector and descriptor combination*

In our distributed monocular SLAM framework, we want to compare a given descriptor with already existing descriptors in a database. Our goal is to find the best keypoint detector and descriptor combination for our framework. We considered the following factors.

- Precision change with the changes in zoom, viewpoint angle, blur and illumination. A good combination would roughly maintain its precision with changing imaging conditions. For example, in Figure 11a the (SIFT, SIFT) combination dropped its precision from 0.8 to 0.75 as zoom changed from lowest to low. However, in the case of (FAST, SIFT) the precision is dropped from 0.9 to 0.0. The latter case has a drastic change in precision.

- Higher precision numbers. Apart from maintaining its precision under varying imaging conditions, we would like to have the precision numbers as close to 1 as possible.

- Size of the descriptor. Our framework transfers descriptor data across the network. For a given set of keypoints, a smaller descriptor is favorable than a larger one, as it reduces the amount of data the framework has to transfer.

- Location accuracy. The distance distribution between the estimated and real location of matching keypoints should have its peak located closer to the distance of 0.

- The shape of the distance distribution. We would like the distribution to have lighter tails, meaning it should have a very little number of outliers.

6.5.2. *The quality measure for keypoint detector and descriptor combination*

To come up with a quality measure for keypoint detector and descriptor combination, we used the following parameters.

- Precision average (p). For imaging conditions, zoom, viewpoint angle, blur and illumination, we separately computed the average precision. This relates to the first two precision related factors mentioned in Section 6.5.1. We would like p to be higher.

- Normalized size of the descriptor (d). We normalize the descriptor size by dividing the descriptor size by 512. As shown in the Table 2 descriptor sizes vary from 32 Bytes to 512 Bytes. Thus descriptors ORB, BRISK, FREAK, SURF and SIFT get 0.0625, 0.125, 0.125, 0.5 and 1 respectively. We would like d to be lower.

- Mode of the distance distribution (m). As we want our location accuracy to be higher, we expect the mode of the distribution to be closer to zero.

- Kurtosis of the distance distribution (k). Kurtosis of a distribution indicates whether data has a higher or a lower number of outliers. If the Kurtosis is high, the distribution is heavy-tailed and has more outliers. Hence, we would like k to be smaller.

Considering the above parameters, the quality measure q_i for imaging condition i is given by,

$$q_i = \frac{pe^{1-m}}{1 + f(\alpha + k)} \tag{5}$$

here α is a constant bias so that $\alpha = 3$ and $f(x)$ is the sigmoid function given by,

$$f(x) = \frac{1}{1 + e^{-x/10}} \tag{6}$$

The overall quality is given by,

$$q = \frac{\sum_{i=1}^{n} \gamma_i q_i}{\beta + d} \tag{7}$$

Here γ_i are constants computed empirically to represent the importance of supporting corresponding imaging condition. As we are considering only four different imaging conditions n has the value four. The maximum possible value for the quality measure is 27.33. The $\sum_{i=1}^{4} \gamma_i$ and β are 32.8 and 3.2 respectively.

Figure 13 shows how each imaging condition contributes to the overall quality measure. The proposed measure ranks (ORB, BRISK) combination as the best candidate for our framework. The rest of the leading combinations, (ORB, ORB), (SIFT, SIFT), (ORB, SURF), (ORB, FREAK) appear fourth, fifth, sixth, seventh and eighth positions in the Figure 10 as well. The first three combinations in the Figure 10, (GFTT, SIFT), (HARRIS, SIFT) and (FAST, SIFT), have higher m values, thus are demoted to be below leading candidates in Figure 13. Interestingly, (SURF, SURF) combination received a lower quality measure value due to its relatively low precision values across all imaging conditions evaluated (see Figures 12 and 11).

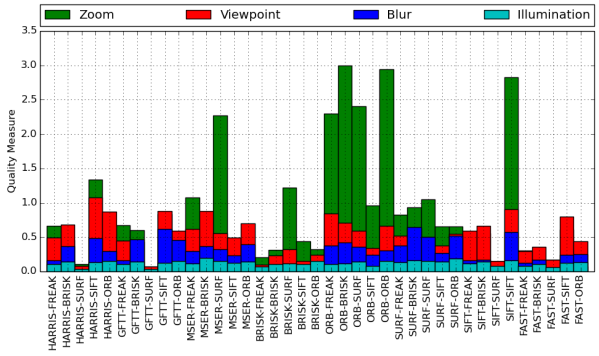


Fig. 13: Quality measure vs Detector-Descriptor combination

7. Evaluation

7.1. Public datasets

To evaluate our system, we need a monocular visual SLAM dataset, with multiple trajectories covering a single scene. We considered publicly available datasets, and they did not satisfy our requirements. For example, the dataset EuRoC [34] contains pure rotations which did not work well with the monocular SLAM approach we used. The Kitti [35] is mainly a stereo dataset, even when we considered a single camera, the direct monocular SLAM process failed since the camera motion is along the optical axis. The TUM-Mono [36] dataset does not provide ground truth for all frames and is primarily suitable for evaluating single agent SLAM. Therefore, we created the DIST-Mono dataset to evaluate our system. We also made it publicly available[2].

7.2. Experimental setup

Our experimental setup is designed to define the ground truth of a camera motion precisely. As shown in Figure 14 we mounted a Point Grey Firefly MV global shutter camera on a Computer Numeric Controller (CNC) machine. We also prepared a $1m \times 1.5m$ scene containing wooden objects. We then moved the camera along a path roughly four minutes each time, while capturing its location ground truth periodically. We captured 640×480 resolution camera frames at 60Hz and ground truth at 40Hz. The CNC machine has 0.2mm accuracy in all three axes. We developed an open-source ROS node [3] to capture the ground truth from the TinyG CNC controller.

Fig. 14: Experimental setup showing a camera mounted on a CNC machine allowing us to capture ground truth information.Camera mounted on a CNC machine

7.3. DIST-Mono dataset

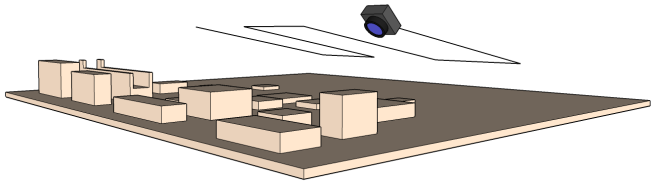
The dataset consists of five sub-datasets. We defined three camera motion paths, Path A, Path B and Path C. All these paths are on a plane slanted above the scene as shown in Figure 15a. These paths have roughly 10% overlap and three different starting points. We generated two datasets using PathA by rotating the camera around its z axis. In S01-A-0, the camera optical

[2] http://slam.cs.iupui.edu
[3] http://github.com/japzi/rostinyg

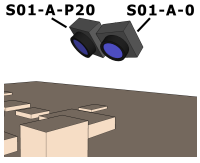
Dataset	Path	Initial camera rotation
S01-A-0	Path A	0
S01-A-P20	Path A	20 CW
S01-B-0	Path B	0
S01-B-N20	Path B	20 CCW
S01-C-0	Path C	0

Table 3: DIST-Mono dataset

axis and scene Y axis is on a vertical plane. In S01-A-P20, we rotated the camera around its y axis by 20°. This is illustrated in Figure 15b. Similarly, we created datasets S01-B-0, S01-B-N20, and S01-C-0 as shown in Table 3.



(a) Motion paths are in a plane slanted above the scene



(b) 20° clockwise rotation

Fig. 15: Camera motion and its initial rotation for datasets

7.4. Experiments

7.4.1. Experiments I

Two of these datasets were then used to deploy two exploring nodes on two separate physical computers. The monitoring node is deployed on a third computer. All these computers run on Ubuntu 14.04 operating system. They are connected via a wired router. This experiment is repeated 100 times, and the resultant transformation between merged two maps is compared against the ground truth.

The resulting relative transformation between dataset S01-A-P20 and dataset S01-B-0 was recorded as shown in Table 4 (in the table, μ is the average over 96 trials, and σ is the standard deviation). The average error in translation and average error in the rotation were 2.7cm and 5.3°, respectively. Furthermore, it merged maps successfully in 96 out of the 100 attempts. The framework failed to detect map overlaps in the remaining four attempts. Once the framework merged two maps, one exploring node displayed its map as in the right hand side map of Figure 5.

7.4.2. Experiments II

Similar to the Experiments I, we used dataset SCENE-A-0 and dataset SCENE-B-N20 in two different exploring nodes. After map merging, each exploring node exported its key frame poses in TUM dataset [37] pose format. Most importantly,

Table 4: Relative transformation with rotation (q) and translation (t)

ξ_{ji}	Ground truth	Results (96 attempts)		Average error
		μ	σ	
q_x	0.00	0.00	0.01	5.33°
q_y	0.38	0.41	0.01	
q_z	0.05	0.08	0.01	
q_w	0.93	0.91	0.01	
t_x(mm)	-680.0	-706.5	6.1	27.4
t_y(mm)	-70.0	-74.6	17.0	
t_z(mm)	350.0	355.8	15.0	

these poses contain key frames from both exploring nodes. We then computed the Absolute Translation RMSE [37] against the ground truth. To support the non-deterministic nature of the distributed system, we ran the experiment five times, and the median result is recorded. Similarly, we performed three more experiments with other combinations of datasets as shown in Table 5. Given monocular visual SLAM, systems do not capture the scale, we manually calculated that in all experiments to minimize the RMSE error.

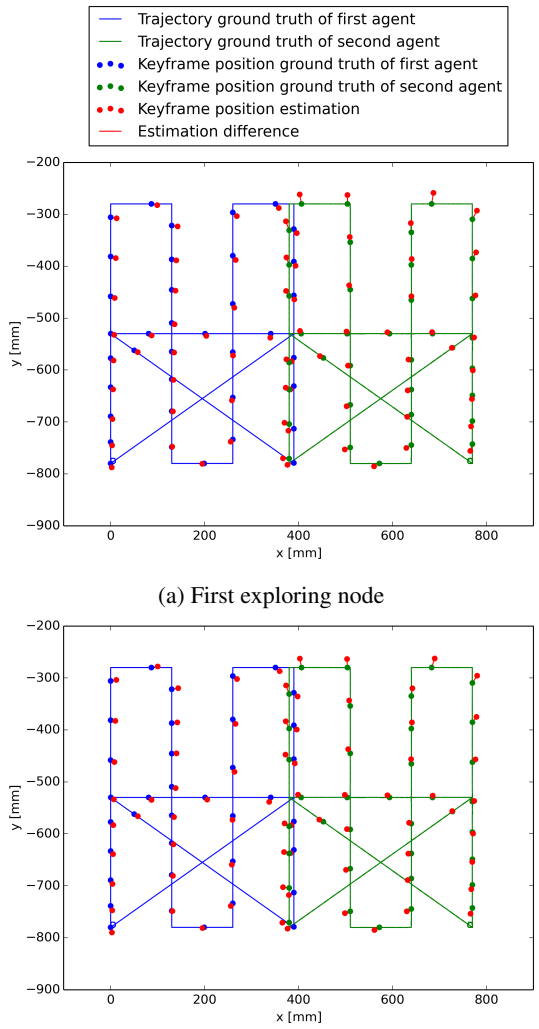
Table 5: RMSE, median, mean and standard deviation against the ground truth of the absolute translation. Number of pose pairs compared are also shown.

Experiment	Node	RMSE (m)	Median (m)	Poses
S01-A-0, S01-B-0	1	0.0105	0.0087	74
	2	0.0106	0.0095	74
S01-A-0, S01-B-N20	1	0.0143	0.0120	74
	2	0.0136	0.0106	74
S01-A-0, S01-C-0	1	0.0046	0.0034	60
	2	0.0089	0.0067	51
S01-B-0, S01-C-0	1	0.0032	0.0024	63
	2	0.0046	0.0041	52

Figure 16 shows how estimated keyframe positions are compared against ground truth in experiment II using datasets S01-A-0 and S01-B-0. After map merging each exploring node has a map with keyframes generated on its own and originated from the other exploring node. In the Figure, ground truth keyframe positions from first and second exploring nodes are shown in blue and green circles respectively. The red circles in the figure display the estimated positions of corresponding keyframe positions. Red lines show the difference between the estimated and the ground truth positions of the keyframe. Similarly, Figure 17 shows keyframe positions for S01-B-0 and S01-C-0 datasets. It also shows the relatively smaller coverage of the S01-C-0 dataset.

8. System Implementation

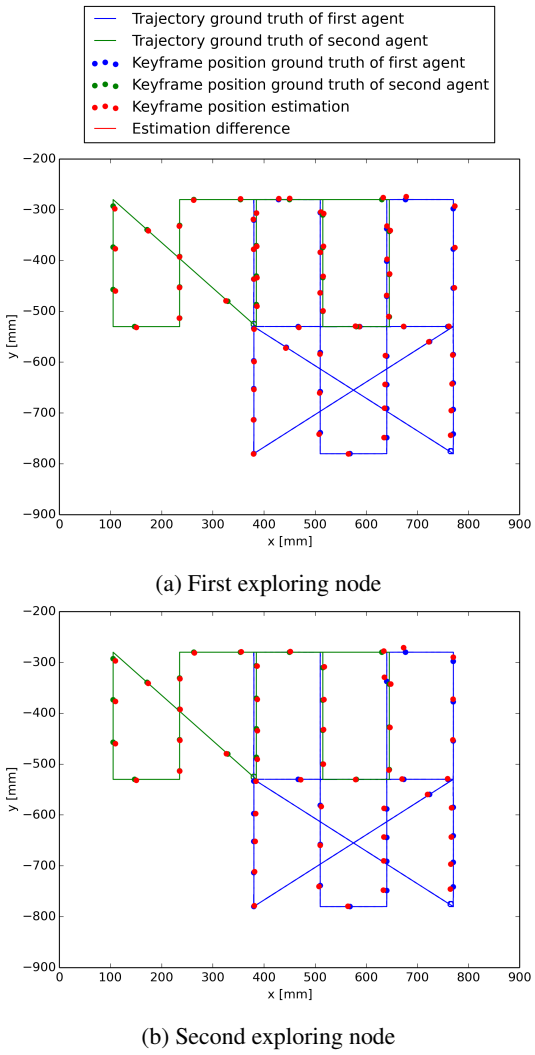
We developed exploring nodes and monitoring nodes as ROS nodes. We used ROS Indigo Igloo infrastructure on Ubuntu 14.04 LTS (Trusty) operating system. Both framework implementation and the comprehensive analysis on state-of-the-art



(a) First exploring node

(b) Second exploring node

Fig. 16: Keyframe position estimation against ground truth for experiment with datasets S01-A-0 and S01-B-0.



(a) First exploring node

(b) Second exploring node

Fig. 17: Keyframe position estimation against ground truth for experiment with datasets S01-B-0 and S01-C-0

feature detector and descriptor combinations we used version 2.4.8 of the OpenCV library.

Nodes in the framework communicate with each other using ROS topics. We used ROS statistics to measure bandwidth utilization in those communication channels. In addition to that ROS statistics could be used to measure, number of dropped messages, mean & standard deviation of the age of messages, and period of messages by all providers.

Our distributed setup contained multiple personal computers running above Ubuntu operating system. Exploring nodes were deployed in computers having Intel Core i5 processors and 16GB RAM. A monitoring node was deployed in Intel Core i7 processor and 16GB RAM. As mentioned in Experiments sections the framework merged maps 96 out of 100 times attempted. Furthermore, to investigate the computational demand, we deployed three nodes of the framework in a single machine and observed successful map merging repeatedly.

Furthermore, exploring nodes merge their maps without interrupting their regular SLAM operation, largely due to the parallelized nature of their architecture.

9. AR Application

We added an AR window to each exploring node to test our framework. The AR window, allows users to add a virtual object (a simple cube, in our example) into its map. This allows us to demonstrate the collaborative AR potential of the distributed SLAM framework. Each exploring node has its local map so that it can render the augmented scene from its viewpoint. It also knows its pose on the global map. This allows it to render objects added by the other exploring nodes as well. Furthermore, exploring nodes can interact with one another using peer-to-peer communication channels of the framework.

Figure 18 shows AR windows of two exploring nodes and two interactively added cubes.

10. Conclusion

In this paper, we introduced a distributed SLAM framework that identifies map overlaps based on an appearance-based

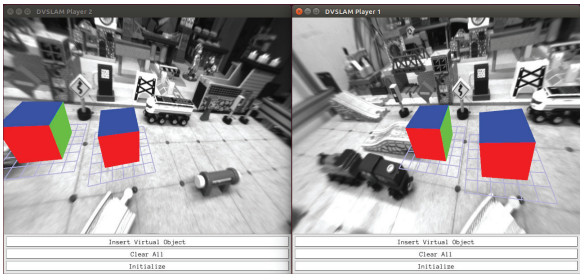


Fig. 18: Same set of virtual objects is viewed from two different exploring nodes

method. For the appearance based method, we have done a comprehensive analysis on the state-of-the-art keypoint detector and descriptors and introduced a quality measure to select the best combination for a distribution visual SLAM framework. The framework operates with no prior knowledge of relative starting poses of its nodes. Using an AR application we have shown that our framework can support collaborative Augmented Reality applications. We also developed a new publicly available dataset and used that for an extensive evaluation of the system.

We developed a quality measure to find the best keypoint detector/descriptor candidate to be used in our proposed map overlap detection method. We performed a comprehensive analysis on state-of-the-art detector/descriptor combinations and achieved better results with ORB and BRISK combination.

Our next step would be improving the exploring node's SLAM process by incorporating features in pose graph optimization. That would help greatly in supporting public datasets as well. We would also like to develop more distributed monocular visual SLAM datasets to evaluate our system at a larger scale. Furthermore, we will evaluate the possibility of using a BoW[38] based method instead of the FLANN[23] method we used to detect map overlaps, mainly to improve the performance of the system. The ultimate goal of this framework is to be ported to truly mobile, resource limited platforms and for the computational nodes to run on such mobile devices.

References

[1] Engel, J, Schps, T, Cremers, D. LSD-SLAM: Large-scale direct monocular SLAM. In: Computer Vision ECCV 2014; vol. 8690 of *Lecture Notes in Computer Science*. Springer International Publishing. ISBN 978-3-319-10604-5; 2014, p. 834–849. URL: http://dx.doi.org/10.1007/978-3-319-10605-2_54. doi:10.1007/978-3-319-10605-2_54.

[2] Egodagamage, R, Tuceryan, M. A collaborative augmented reality framework based on distributed visual SLAM. In: 2017 International Conference on Cyberworlds (CW). 2017, p. 25–32. doi:10.1109/CW.2017.12.

[3] Smith, R, Self, M, Cheeseman, P. Estimating Uncertain Spatial Relationships in Robotics. In: Cox, I, Wilfong, G, editors. Autonomous Robot Vehicles. Springer New York. ISBN 978-1-4613-8999-6; 1990, p. 167–193. URL: http://dx.doi.org/10.1007/978-1-4613-8997-2_14. doi:10.1007/978-1-4613-8997-2_14.

[4] Montemerlo, M, Thrun, S, Koller, D, Wegbreit, B. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In: In Proceedings of the AAAI National Conference on Artificial Intelligence. AAAI; 2002, p. 593–598.

[5] Davison, A, Reid, I, Molton, N, Stasse, O. MonoSLAM: Real-time single camera SLAM. Pattern Analysis and Machine Intelligence, IEEE Transactions on 2007;29(6):1052–1067. doi:10.1109/TPAMI.2007.1049.

[6] Klein, G, Murray, D. Parallel tracking and mapping for small AR workspaces. In: Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on. 2007, p. 225–234. doi:10.1109/ISMAR.2007.4538852.

[7] Fischler, MA, Bolles, RC. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Commun ACM 1981;24(6):381–395. URL: http://doi.acm.org/10.1145/358669.358692. doi:10.1145/358669.358692.

[8] Hartley, RI, Zisserman, A. Multiple View Geometry in Computer Vision. Second ed.; Cambridge University Press, ISBN: 0521540518; 2004.

[9] Triggs, B, McLauchlan, PF, Hartley, RI, Fitzgibbon, AW. Bundle adjustmenta modern synthesis. In: Vision algorithms: theory and practice. Springer; 2000, p. 298–372.

[10] Strasdat, H, Montiel, J, Davison, A. Real-time monocular SLAM: Why filter? In: Robotics and Automation (ICRA), 2010 IEEE International Conference on. 2010, p. 2657–2664. doi:10.1109/ROBOT.2010.5509636.

[11] Newcombe, RA, Lovegrove, S, Davison, A. DTAM: Dense tracking and mapping in real-time. In: Computer Vision (ICCV), 2011 IEEE International Conference on. 2011, p. 2320–2327. doi:10.1109/ICCV.2011.6126513.

[12] Nettleton, E, Thrun, S, Durrant-Whyte, H, Sukkarieh, S. Decentralised SLAM with low-bandwidth communication for teams of vehicles. In: Field and Service Robotics. Springer; 2006, p. 179–188.

[13] Paull, L, Huang, G, Seto, M, Leonard, J. Communication-constrained multi-auv cooperative SLAM. In: Robotics and Automation (ICRA), 2015 IEEE International Conference on. 2015, p. 509–516. doi:10.1109/ICRA.2015.7139227.

[14] Howard, A, Parker, L, Sukhatme, G. The SDR experience: Experiments with a large-scale heterogeneous mobile robot team. In: Ang MarceloH., J, Khatib, O, editors. Experimental Robotics IX; vol. 21 of *Springer Tracts in Advanced Robotics*. Springer Berlin Heidelberg. ISBN 978-3-540-28816-9; 2006, p. 121–130. URL: http://dx.doi.org/10.1007/11552246_12. doi:10.1007/11552246_12.

[15] Fox, D, Ko, J, Konolige, K, Limketkai, B, Schulz, D, Stewart, B. Distributed multirobot exploration and mapping. Proceedings of the IEEE 2006;94(7):1325–1339. doi:10.1109/JPROC.2006.876927.

[16] Bay, H, Ess, A, Tuytelaars, T, Van Gool, L. Speeded-up robust features (surf). Comput Vis Image Underst 2008;110(3):346–359. URL: http://dx.doi.org/10.1016/j.cviu.2007.09.014. doi:10.1016/j.cviu.2007.09.014.

[17] Lowe, DG. Distinctive image features from scale-invariant keypoints. Int J Comput Vision 2004;60(2):91–110. URL: http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94. doi:10.1023/B:VISI.0000029664.99615.94.

[18] Rublee, E, Rabaud, V, Konolige, K, Bradski, G. ORB: An efficient alternative to sift or surf. In: Computer Vision (ICCV), 2011 IEEE international conference on. IEEE; 2011, p. 2564–2571.

[19] Leutenegger, S, Chli, M, Siegwart, RY. BRISK: Binary robust invariant scalable keypoints. In: Computer Vision (ICCV), 2011 IEEE International Conference on. IEEE; 2011, p. 2548–2555.

[20] Gamage, R, Tuceryan, M. An experimental distributed framework for distributed simultaneous localization and mapping. In: 2016 IEEE International Conference on Electro Information Technology (EIT). 2016, p. 0665–0667. doi:10.1109/EIT.2016.7535318.

[21] Quigley, M, Conley, K, Gerkey, BP, Faust, J, Foote, T, Leibs, J, et al. ROS: an open-source robot operating system. In: ICRA Workshop on Open Source Software. 2009,.

[22] Engel, J, Sturm, J, Cremers, D. Semi-dense visual odometry for a monocular camera. In: Computer Vision (ICCV), 2013 IEEE International Conference on. 2013, p. 1449–1456. doi:10.1109/ICCV.2013.183.

[23] Muja, M, Lowe, DG. Fast approximate nearest neighbors with automatic algorithm configuration. In: International Conference on Computer Vision Theory and Application VISSAPP'09). INSTICC Press; 2009, p. 331–340.

[24] Horn, BK. Closed-form solution of absolute orientation using unit quaternions. JOSA A 1987;4(4):629–642.

[25] Feature (computer vision). https://en.wikipedia.org/wiki/Feature_(computer_vision); 2017. [Online; accessed 15-October-2017].

[26] Harris, C, Stephens, M. A combined corner and edge detector. In: In Proc. of Fourth Alvey Vision Conference. 1988, p. 147–151.

[27] Shi, J, et al. Good features to track. In: Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on. IEEE; 1994, p. 593–600.

[28] Rosten, E, Drummond, T. Fusing points and lines for high performance tracking. In: Proceedings of the Tenth IEEE International Conference on Computer Vision - Volume 2. ICCV '05; Washington, DC, USA: IEEE Computer Society. ISBN 0-7695-2334-X-02; 2005, p. 1508–1515. URL: https://doi.org/10.1109/ICCV.2005.104. doi:10.1109/ICCV.2005.104.

[29] Rosten, E, Porter, R, Drummond, T. Faster and better: a machine learning approach to corner detection. CoRR 2008;abs/0810.2434. URL: http://arxiv.org/abs/0810.2434.

[30] Matas, J, Chum, O, Urban, M, Pajdla, T. Robust wide-baseline stereo from maximally stable extremal regions. Image and vision computing 2004;22(10):761–767.

[31] Forssen, PE, Lowe, DG. Shape descriptors for maximally stable extremal regions. In: 2007 IEEE 11th International Conference on Computer Vision. 2007, p. 1–8. doi:10.1109/ICCV.2007.4409025.

[32] Alahi, A, Ortiz, R, Vandergheynst, P. FREAK: Fast retina keypoint. In: Computer vision and pattern recognition (CVPR), 2012 IEEE conference on. Ieee; 2012, p. 510–517.

[33] Mikolajczyk, K, Tuytelaars, T, Schmid, C, Zisserman, A, Matas, J, Schaffalitzky, F, et al. A comparison of affine region detectors. International journal of computer vision 2005;65(1-2):43–72.

[34] Burri, M, Nikolic, J, Gohl, P, Schneider, T, Rehder, J, Omari, S, et al. The EuRoC micro aerial vehicle datasets. The International Journal of Robotics Research 2016;doi:10.1177/0278364915620033.

[35] Geiger, A, Lenz, P, Urtasun, R. Are we ready for autonomous driving? the KITTI vision benchmark suite. In: Conference on Computer Vision and PatternRecognition (CVPR). 2012,.

[36] Engel, J, Usenko, V, Cremers, D. A photometrically calibrated benchmark for monocular visual odometry. In: arXiv:1607.02555. 2016,.

[37] Sturm, J, Engelhard, N, Endres, F, Burgard, W, Cremers, D. A benchmark for the evaluation of rgb-d SLAM systems. In: Proc. of the International Conference on Intelligent Robot Systems (IROS). 2012,.

[38] Gálvez-López, D, Tardós, JD. Bags of binary words for fast place recognition in image sequences. IEEE Transactions on Robotics 2012;28(5):1188–1197. doi:10.1109/TRO.2012.2197158.