Computer Vision and Image Understanding 118 (2014) 97–110

Contents lists available at ScienceDirect

Computer Vision and Image Understanding

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

Efficient keyframe-based real-time camera tracking ☆

Zilong Dong [a], Guofeng Zhang [a,*], Jiaya Jia [b], Hujun Bao [a]

[a] State Key Lab of CAD&CG, Zhejiang University, Hangzhou, PR China
[b] Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, NT, Hong Kong

ARTICLE INFO

Article history:
Received 22 September 2011
Accepted 21 August 2013
Available online 30 August 2013

Keywords:
Keyframe selection
Real-time camera tracking
Global localization
Online map extension

ABSTRACT

We present a novel keyframe-based global localization method for markerless real-time camera tracking. Our system contains an offline module to select features from a group of reference images and an online module to match them to the input live video for quickly estimating the camera pose. The main contribution lies in constructing an optimal set of keyframes from the input reference images, which are required to approximately cover the entire space and at the same time to minimize the content redundancy among the selected frames. This strategy not only greatly saves computation, but also helps significantly reduce the number of repeated features. For a large-scale scene, it requires a significant effort to capture sufficient reference images and reconstruct the 3D environment. In order to alleviate the effort of offline preprocessing and enhance the tracking ability in a larger scale scene, we also propose an online reference map extension module, which can real-time reconstruct new 3D features and select online keyframes to extend the keyframe set. In addition, we develop a parallel-computing framework that employs both GPUs and multi-threading for speedup. Experimental results show that our method dramatically enhances the computing efficiency and eliminates the jittering artifacts in real-time camera tracking.

1. Introduction

Vision-based camera tracking aims to estimate the camera poses from input images or videos. It is the foundation for solving a wide spectrum of computer vision problems, e.g., 3D reconstruction, video registration and enhancement. Offline camera tracking has been well studied, with several state-of-the-art softwares (e.g., bundler,[1] ACTS[2]), as well as extensive research findings [24,40,46]. Real-time camera tracking [14,31,30] has recently attracted much attention, as it has found many applications for mobile robotics and augmented reality.

In this paper, we propose a practical real-time camera tracking system by combining global localization (GL) and parallel tracking and mapping schemes, which involves an incomplete offline preprocessing for representing the 3D environment using features and an online step for real-time feature matching and reference map (i.e. 3D features and keyframes) extension. Specially, the offline step extracts sparse invariant features from the captured reference images and uses them to represent the scene. The 3D locations of these invariant features can be estimated by the offline structure-from-motion (SfM). Afterwards, taking these features as the reference, for each online image, we can extract the features and establish the correspondences with the reference ones, so that the camera pose can be quickly estimated.

We generally call the above scheme as GL scheme because it matches features of the input frame to the whole model directly. It is robust to large camera motion and also precludes the possibility of error accumulation. It, however, has the following common problems with previous work. First, it relies excessively on the feature distinctiveness, which cannot be guaranteed when the space scale is large or the scene contains repeated structures. It was observed that the matching reliability decreases quickly when the number of features increases, which greatly affects the robustness and practicability of this system in camera tracking. Second, since GL scheme relies on the offline reconstruction result, it will fail if the camera moves to a new place which is not covered by reference images. PTAM [28] and MonoSLAM [15] are two state-of-the-art techniques proposed to solve the tracking problem in unknown scenes, which have been successfully adopted by many practical systems. However, they are restricted to deal with a small scene with only thousands of features.

In this paper, we solve the above problems and develop a complete real-time tracking system. Our contribution is threefold. First, we propose an effective keyframe-based method to enable global localization in large-scale scenes. A novel keyframe selection algorithm is employed to effectively reduce the online matching ambiguity and redundancy. These keyframes are selected from all reference images to abstract the space with a few criteria: (i) the

☆ This paper has been recommended for acceptance by Nikos Paragios.
* Corresponding author. Fax: +86 571 88206680.
E-mail address: zhangguofeng@cad.zju.edu.cn (G. Zhang).
[1] http://www.cs.cornell.edu/snavely/bundler/.
[2] http://www.zjucvg.net/acts/acts.html.

1077-3142/$ - see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.cviu.2013.08.005

keyframes should be able to approximate the original reference images and contain as many salient features as possible; (ii) the common features among these frames are expected to be minimum in order to reduce the redundancy; (iii) the features should be distributed evenly in the keyframes such that given any new input frame in the same environment, the system can always find sufficient feature correspondences and compute accurate camera poses.

Second, with the extracted keyframes, in the real-time camera tracking stage, we contribute an extremely efficient algorithm to find candidate keyframes which are most similar to the online input frame. Because the frame is only matched with the candidate keyframes, the computation can be greatly saved compared to the conventional global feature matching.

Third, we develop an online reference map extension method to significantly enhance the ability of camera tracking and global localization. Especially, while the camera moves into a new place which is not sufficiently covered by reference images, new online keyframes will be inserted into the existing keyframe set, with newly reconstructed 3D features, so that the camera motion can still be reliably estimated.

All of above modules are integrated under a parallel-computing framework using GPU and multi-threading for further speedup. A preliminary version of the work appeared in [16]. In this paper, we have made the following improvements.

1. Introduced an improved real-time camera tracking framework based on an incomplete offline reference map reconstruction, which combined global localization and online reference map extension to make the camera tracking more robust in a larger scale scene with less offline preprocessing effort.
2. Used GPU to accelerate SIFT feature extraction and reduce system latency.
3. Introduced the two-pass keyframe-based matching, which can quickly obtain a set of evenly distributed 2D–3D correspondences to make the camera pose estimation more reliable. Temporal information was also utilized to improve the matching efficiency and robustness.

2. Related work

2.1. Markless real-time camera tracking

The simultaneous Localization and Mapping (SLAM) techniques have been extensively studied [14,17,7,15,29] for real-time camera tracking. SLAM methods can simultaneously estimate camera parameters as well as 3D scene structure online, using a partial observation model. They typically use the frame-to-frame matching and confining-search-region strategies for rapid feature matching. However, drifting and tracking failure can frequently occur because this scheme relies highly on the estimation accuracy in past frames. The major issues that were addressed include relocalization after camera lost [50,6,29], submap merging and switching [5], and loop detection and closure [10,18,49].

To recover from tracking failure in SLAM, Williams et al. [50] proposed a rapid relocalization method, which uses a variant of Lepetit and Fua's feature description and performs matching with the randomized tree classifier [32]. However, this method is memory intensive requiring about 1.3 MB per feature. It thus is not suitable for handling a large-scale scene. Invariant feature descriptors [35,4] require much less memory resource and can be used for global localization; but the computation is more expensive.. Chekhlov et al. [6] proposed using the appearance-based indexing on space and scale to facilitate the use of invariant feature descriptors for relocalization. This approach requires the camera pose information to reduce the search space; or else, the matching is not reliable when dealing with large maps.

Klein and Murray [28] presented an impressive real-time SLAM system for small AR workspaces, where tracking and mapping are separately processed in different threads. Castle et al. [5] extended this work to allow multiple tracking cameras to simultaneously work in several maps. Based on the framework of PTAM, J. Stuehmer and Cremers [27] and Newcombe and Davison [37] tried to recover the dense 3D structures of the scene, which are useful for some AR applications. Angeli and Davison [1] also clustered the 3D points of PTAM online with consistent appearance and position to aid AR tasks. However, all of them are only suitable to handle small desktop scenes due to the frequent use of expensive bundle adjustment. In order to deal with large work spaces, SLAM methods typically need to divide the map into a series of sub-ones, each of which is small enough to enable real-time processing [19,8], or use multiple cameras [22]. Under this background, overlap detection and loop closure among multiple submaps becomes an important issue. Several methods [10,18,49] have been proposed to address it. Recently, Hauke Strasdat and Konolige [25] proposed a double window optimization framework for scalable SLAM, which scales for both accurate local reconstruction and large-scale loopy camera motion. Their method can be integrated with the PTAM framework easily to achieve realtime camera tracking in a larger scene. However, this method relies heavily on loop closure, and the estimated camera pose and 3D structure may drift without loops.

If the 3D representation for the space is available, real-time camera tracking can be quickly accomplished. Several markerless algorithms [48,11] employed the object's CAD model to facilitate camera pose estimation. However, these CAD models are usually difficult, if not impossible, to be constructed. Skrypnyk and Lowe [45] proposed modeling natural scenes using a set of sparse invariant features. The developed system contains two modules, i.e. the offline feature-based scene modeling and online camera tracking. It relies on the distinctiveness of the SIFT features, and is therefore limited to a relatively small workspace. For large scenes such as urban environments, not only the tracking process, but the 3D modeling of the scenes is a problem because of the camera drift. Gay-Bellile et al. [21] proposed to use the coarse 3D city models from Google Earth to correct the drift while reconstructing the 3D landmarks. They used vocabulary tree to recognize the viewpoints during the on-line localization and integrated a SfM process to model the scene variations. The main drawback of their framework is the requirements of city models. We solve these problems with a two-stage tracking system in this paper.

2.2. Keyframe-based methods

Keyframe selection is a common technique to reduce data redundancy. In the real-time camera tracking method of [28], a set of online keyframes were selected, which facilitated bundle adjustment for the 3D map recovery. In [29], the keyframes were used for relocalization with simple image descriptors. For model-based tracking, Vacchetti et al. [48] selected keyframes manually in the offline mode, and matched each online frame to a keyframe with the closest visible area. To track multiple objects simultaneously, Park et al. [39] selected a set of snapshots of each object as keyframes. The online frames were compared with the keyframe set in a round-robin fashion.

In all these methods, keyframes are selected manually or through a simple procedure, which cannot guarantee optimality for the tracking task when the camera undergoes complex motion. There has been research in video processing [34,23,47] aiming to extract the frames for video reconstruction, browsing, or retrieval. In our method, a set of optimal keyframes are selected for

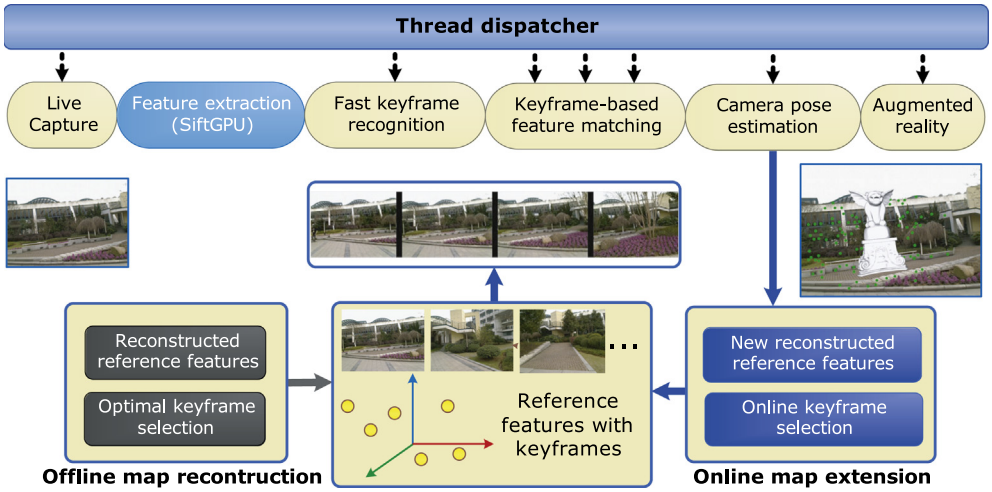

Fig. 1. Framework overview. Our system employs a parallel-computing framework, where both GPU and multi-threading are used. The system contains modules, which run in separate working threads, and are synchronized by the frame time stamp.

representing and abstracting a space. They are vital for efficient online feature matching.

2.3. Feature-based location recognition

There are approaches which employ invariant features for object and location recognition [44,42,43,12,13,2]. These methods typically extract invariant features for each image, and use them to estimate the similarity of different images. For dealing effectively with a large-scale image database, a vocabulary tree [38,9] was adopted to organize and search millions of feature descriptors. It is thus notable that these methods do not extract sparse keyframes to reduce the data redundancy, and cannot be directly used for real-time tracking. The appearance-based SLAM method of [13] considered the inter-dependency among features and applied the bag-of-words method to increase the speed of location recognition. However, the computational cost is still high.

A keyframe-based method to deal with tracking failure and closing loops in the real-time monocular SLAM was described in [18]. This method builds visual vocabulary incrementally, which is used to match online images. For relocalization, the online frame needs to be compared with all local nodes (or keyframes), which would cause obvious system latency if the number of nodes is large. In contrast, our keyframe recognition algorithm is able to produce matching almost instantly. The processing time can be nearly constant even when lots of keyframes are selected.

Irschara et al. [26] proposed a fast location recognition technique based on SFM point clouds, which is tightly related to our work. In order to reduce the 3D database size to improve recognition performance, synthetic views are involved to introduce a compressed 3D scene representation. In comparison, our method selects an optimal set of keyframes from the input reference images by minimizing an energy function, which yields a balance between the representation completeness and the redundancy. In addition, our two-pass keyframe-based matching can obtain reliable and evenly distributed 2D–3D correspondences with low resolution images and less features, which is very important for robust camera pose estimation.

3. Framework overview

We first give an overview of our framework in Fig. 1. It contains two modules. In the offline module, we gather a group of *reference images* to model the space. SIFT features [35] are detected to establish the multi-view correspondences. Then we use the SfM method [52] to estimate the camera poses for these reference images together with the 3D locations of the matched SIFT features. Note that we do not need to capture the scene completely, because in our online module, our system can quickly recover and extend the local 3D structure to remedy the incomplete structure information.

In the online module, we estimate the camera parameters for each input frame at real-time rate given a captured live video in the same space. We assume that the intrinsic parameters of the camera are already known, which can be recovered by using an offline SfM technique or a camera calibration tool.[3] For each online image, we need to estimate the 6-DOF pose of the camera relative to its surroundings. Instead of frame-by-frame matching using all reference images, in our approach, we select several optimal keyframes to represent the scene, and build a vocabulary tree for online keyframe recognition. With the 3D locations of reference features, we can establish sufficient and evenly distributed 2D–3D correspondences for all matched features, which enable the estimation of the camera poses of all live frames.

After estimating the camera pose of the input frame, we will further determine whether to select the input frame as a new keyframe. If the input frame is selected as a keyframe, we will extract more SIFT features and match them to nearby keyframes for triangulating new 3D features. The selected keyframe with reconstructed 3D features is fused with the existing reference map to make the camera tracking more robust.

4. Offline optimal keyframe selection

Given a live frame, directly using all reference images for feature detection and matching to estimate the camera pose is impractical. So the first step is to process reference images taken with the camera moving in a workspace in the offline module. We propose selecting keyframes from all reference ones to boost the feature detection efficiency and reduce the ambiguity in the following matching process. To this end, there should exist as many salient features as possible in a minimum set of keyframes. The features should also be (near-) uniformly distributed in the scene.

In the beginning, we apply SfM to all reference images to compute 3D points for the corresponding features. This process

[3] http://www.vision.caltech.edu/bouguetj/calib_doc/.

involves matching SIFT features based on the descriptors [35] in different frames. As a result, each 3D point corresponds to a few features in multiple frames.

For simplicity's sake, we cluster the matched features in different reference images and unify their representation by averaging their descriptors. Each of the resulted feature tracks is denoted as §, which contains a series of common features located in multiple frames. It is written as $\mathcal{X} = \{\mathbf{x}_i | i \in f(\mathcal{X})\}$, where $f(\mathcal{X})$ denotes the reference image set spanned by $\mathcal{X}$. All $\mathbf{x}$s in each $\mathcal{X}$ map to a single 3D point. It is notable that $|f(\mathcal{X})|$ should be at least 2, otherwise its 3D position cannot be determined. In practice, the larger $|f(\mathcal{X})|$ is, the more precise the 3D information of $\mathcal{X}$ will be. We denote the subset of $\mathcal{X}$ where $|f(\mathcal{X}) \geqslant l|$ as $V(\widehat{I})$, namely *superior feature tracks*. l is usually set to 5 in our experiments, and we only estimate the 3D locations of these superior feature tracks.

We define the keyframe selection problem as follows. Given a total of n reference images $\hat{I} = \{I_i | i = 1, 2, \ldots, n\}$, where n could be a large value if dense frames are provided, we aim to select an optimal set of keyframes $F = \{I_k | k = i_1, i_2, \ldots, i_K\}$, which minimizes the cost defined in the function $E(F; \hat{I})$. The keyframe number K is adaptive in our method to maximize the flexibility. $E(F; \hat{I})$ consists of two terms, i.e., the completeness term $E_c(F)$ and the redundancy term $E_r(F)$, modeling respectively the feature completeness and frame redundancy:

$$E(F; \hat{I}) = E_c(F) + \lambda E_r(F), \tag{1}$$

where λ is a weight. The two terms are described further below.

4.1. Completeness term

For real-time camera tracking, it is expected that any feature in the live frame can find matches with the reference ones to provide enough information for its 3D coordinate determination. So our completeness term is to constrain that the selected keyframes contain features in as many superior feature tracks as possible.

We define the saliency of one track as the combination of two measures, i.e. the feature count in different reference images $|f(\mathcal{X})|$ and the Difference-of-Gaussian (DoG) strength, and write it as

$$s(\mathcal{X}) = D(\mathcal{X}) \cdot \min(|f(\mathcal{X})|, T), \tag{2}$$

where T is the threshold to prevent a long track to suppress other features' contribution to the function. It is usually set to 30 in our experiments. A large value in $|f(\mathcal{X})|$ indicates high reliability to compute the corresponding 3D point for $\mathcal{X}$ in SfM. $D(\mathcal{X})$ is expressed as

$$D(\mathcal{X}) = \frac{1}{|f(\mathcal{X})|} \sum_{i \in f(\mathcal{X})} D_i(\mathbf{x}_i),$$

where D_i denotes the DoG map [35]. $D(\mathcal{X})$ denotes the average DoG map for all features in $f(\mathcal{X})$. The larger $D(\mathcal{X})$ is, the higher saliency the feature track $\mathcal{X}$ has.

Despite the above two measures, another important criterion to make real-time camera tracking reliable is the spatially near-uniform distribution of all features. It is a basic guarantee for finding matches given any online frame in the same space.

The feature density $d(\mathbf{y}_i)$ for the pixel $\mathbf{y}$ in image i is the feature count in the local window. Its computation is described in Algorithm 1. The local window size is generally set to 31×31 for 640×480 images. If there are 300 uniformly distributed features, the feature density of these features will be 1. In our experiments, we generally extract 300 features for each online image. With the feature density of all images, we define the *track density* as

$$d(\mathcal{X}) = \frac{1}{|f(\mathcal{X})|} \sum_{i \in f(\mathcal{X})} d(\mathbf{x}_i),$$

where $d(\mathbf{x}_i)$ denotes the feature density of the pixel at $\mathbf{x}_i$ in image i.

Algorithm 1. Feature density computation for image i

1. Initialize all densities to zeros.
2. for $j = 1, \ldots, m$, % m is the number of features in image i
   for each pixel $\mathbf{y} \in W(\mathbf{x}_j)$,
   %W is a 31×31 window centered at $\mathbf{x}_j$ and
   %$\mathbf{x}_j$ is the coordinate of feature j
   $d_i(\mathbf{y})$ += 1.

Finally, our completeness term is defined as:

$$E_c(F) = 1 - \left(\sum_{\mathcal{X} \in V(F)} \frac{s(\mathcal{X})}{\eta + d(\mathcal{X})} \right) \Bigg/ \left(\sum_{\mathcal{X} \in V(\widehat{I})} \frac{s(\mathcal{X})}{\eta + d(\mathcal{X})} \right), \tag{3}$$

where scalar η controls the sensitivity to feature density. $V(F)$ and $V(\widehat{I})$ denote the set of superior features in the keyframes F and all reference frames $\hat{I}$ respectively. $d(\mathcal{X})$ is put in the denominator to discourage selecting multiple features in a small region, which can help scatter superior features more uniformly in images.

4.2. Redundancy term

Redundancy term leads to the reduction of keyframe overlapping. Since we have detected feature set $f(\mathcal{X})$, the redundancy minimization problem is equivalent to minimizing the features in each $f(\mathcal{X})$ included in multiple keyframes. We therefore define

$$E_r(F) = \frac{1}{|V(\widehat{I})|} \sum_{\mathcal{X} \in V(F)} (|f(\mathcal{X}) \cap F| - 1), \tag{4}$$

where $1/|V(\widehat{I})|$ is for normalization with respect to the number of the superior features in all reference frames. $|f(\mathcal{X}) \cap F|$ indicates the number of features in $f(\mathcal{X})$ that are also included in the keyframes. $|f(\mathcal{X}) \cap F| = 1$ means no redundancy.

4.3. Keyframe selection

An exhaustive search in the reference images can certainly find the optimal set of keyframes that minimizes the cost in Eq. (1). However, it is not computationally efficient to evaluate the 2^n subsets. In [34], with a fixed number of keyframes, dynamic programming (DP) was used to search for the solution for video summarization. Note that this scheme does not suit our system because our objective function has a much more complex format and the number of keyframes is supposedly not fixed. Further, the method of Liu and Kender [34] assumed that only adjacent frames possibly overlap, and accordingly proposed a greedy algorithm while we do not make the same assumption.

Our keyframe selection algorithm is based on a steepest-descent method as described in Algorithm 2. It proceeds in the following way. To begin with, we construct an empty frame set F and then progressively add frames. In each pass, a new keyframe that reduces the *most* energy is added to F. This process continues until the cost cannot be reduced anymore. The computation complexity is $O(n^2)$. In our experiments, it takes only a few seconds to find keyframes from hundreds or thousands of images.

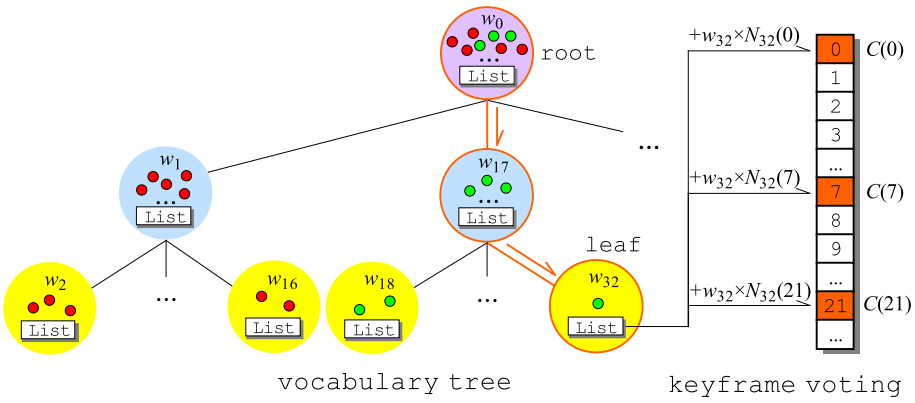

Fig. 2. A vocabulary tree. All feature descriptors are originally in the root node, and are partitioned hierarchically. Each node has a weight to represent its distinctiveness.

Algorithm 2. Keyframe selection

1. Let $F = \emptyset$.
2. If $\forall I_i \in \{\widehat{I} \setminus F\}, E(F \cup \{I_i\}) \geqslant E(F)$, exit.
3. Otherwise, $I\prime = \arg\min_{I_i \in \{\widehat{I} \setminus F\}} E(F \cup \{I_i\})$, and $F = F \cup \{I\prime\}$, go to step 2.

5. Online keyframe recognition and matching

With the collected keyframes, we perform feature matching for online camera tracking. However, it is still costly to find all the matches between the input frames and all keyframes, especially when there exist a considerable number of keyframes. We observe that any incoming frame only covers a small portion of the space; so it is inevitable that many keyframes do not share any common content with the input. We thus exclude those unrelated keyframes to save computation by employing a vocabulary-based fast selection algorithm.

5.1. Vocabulary tree construction

Given a set of keyframes, we construct a visual vocabulary tree by hierarchically clustering all the descriptors of the superior feature tracks. Our tree construction is similar to those of [38,43] where the vocabulary V is organized in l levels with the branching factor b. The root node contains all descriptors. The K-Means method is used to partition the descriptors into b clusters; then all of them become children of the root node. This process continues, recursively partitioning nodes until a specified level l is reached. The final vocabulary tree has $|V|$ nodes, and each node i is associated with a mean descriptor for all feature tracks under it. Fig. 2 gives an illustration. Each node i thus is related to a group of keyframes L_i spanned by the tracks. $N_i(k)$ denotes the number of superior features in keyframe k that are clustered under node i.

We assign each node i a weight w_i, which represents the distinctiveness. In our system, the weight is defined as

$$w_i = \log \frac{K}{|L_i|}, \tag{5}$$

where K is the total number of the keyframes and $|L_i|$ denotes the number of the keyframes spanned by the tracks for node i. Nodes near the root generally relate to a large group of keyframes (with large $|L_i|$). Their frequent appearance implies small importance in the following keyframe selection. We thus assign small weights to them. On the contrary, the leaf nodes are associated with a small number of keyframes and are more important for identification.

The count of all nodes $|V|$ is determined by the branching factor b and tree depth l. For example, if 20–80 keyframes are selected and each keyframe contains about 500–1000 features in the superior tracks. We generally set $b = 10$ and $l = 5$.

5.2. Candidate keyframe searching

In [38,43], an appearance vector is used to describe an image, where each element corresponds to one node in a vocabulary tree. The similarity of two images is approximated as the distance between the corresponding vectors. Note that the computation complexity of this strategy grows linearly with the number of keyframes.

Algorithm 3. Candidate keyframe selection

1. Set the matching value $C(k) = 0$ for each keyframe k.
2. For each online frame, the detected m features are matched respectively from the root node to leafs in the vocabulary tree as follows:
   for each feature **x** in the live frame **do**
   for each level of V **do**
   find the node i containing the feature most similar to **x**,
   if its weight $w_i > \tau$,
   for each keyframe $k \in L_i$, **do**
   $C(k)$ += $N_i(k) \cdot w_i$.
   end for
   end for
   end for
3. Select $\mathcal{K}$ keyframes with the largest C.

Here, we introduce a more efficient keyframe recognition algorithm (Algorithm 3 with an illustration in Fig. 2). The computational complexity is $O(m \cdot \tilde{L})$, where $\tilde{L}$ is the average number of the keyframes that are associated in each node in the matching process from the root to leafs, i.e., $\tilde{L} = \text{avg}_i |L_i|$.

It is particularly worth noting that in Algorithm 3, we have used a threshold τ to exclude nodes with small weights. These nodes are primarily located close to the root because they, due to inclusion of multiple features tracks, are generally associated with many frames. So $\tilde{L}$ only counts the keyframes included in the near-leaf nodes. As a result, the time spent for keyframe recognition increases very slowly with the expanding of keyframes.

5.3. Two-pass keyframe-based matching

Algorithm 4. Two-pass keyframe-based matching

1. For each feature $\mathbf{x}_j$ in the online image, set $C^1(\mathbf{x}_j) = 0$ and $C^2(\mathbf{x}_j) = 0$.
2. Perform first-pass Matching:
   for $i = 1, \ldots, 64$ **do**
   for each unmatched feature $\mathbf{x}_j \in B_i$ and $C^1(\mathbf{x}_j) = 0$ **do**
   set $C^1(\mathbf{x}_j) = 1$, and
   for $k = 1, \ldots, \mathcal{K}$ **do**
   find the 10 features $\{\mathcal{N}_s^k(\mathbf{x}_j)|s = 1, \ldots, 10\}$ from keyframe k that are most similar with $\mathbf{x}_j$. If $\mathcal{N}_1^k(\mathbf{x}_j)$ satisfies the 2NN heuristic, stop the matching of B_i.
   end for
   end for
   end for
3. Estimate the fundamental matrix between the online frame and each keyframe, and use it to remove matching outliers.
4. If there are already N inlier matches, stop.
5. Perform second-pass matching:
   for $i = 1, \ldots, 64$ **do**
   for each feature $\mathbf{x}_j \in B_i$ and $C^1(\mathbf{x}_j) = 1$ **do**
   if $C^2(\mathbf{x}_j) = 0$, set it to 1 and
   for $k = 1, \ldots, \mathcal{K}$ **do**
   screen the features and obtain $\{\tilde{\mathcal{N}}_s^k(\mathbf{x}_t)|s = 1, \ldots\}$ by the epipolar geometry. If one of them satisfies the local 2NN heuristic and is very similar to $\mathbf{p}(\mathbf{x}_j)$, stop the matching of B_i.
   end for
   end for
   end for
6. Repeat steps 2–5 until N matches are found or all $\mathbf{x}$s have $C^1(\mathbf{x}) = 1$ and $C^2(\mathbf{x}) = 1$.

After selecting the most related keyframes for an online image, we perform feature matching. For robust and efficient camera pose estimation, the images should contain (1) sufficient but not excessive feature matches since too many feature matches significantly increase the computation cost. (2) The matched features should be distributed in images as uniform as possible to minimize estimation error. All these criteria are important; but not all of them are considered in previous work. We propose a two-pass keyframe-based matching to satisfy them.

This method aims to find the common SIFT features between the input online frame $\tilde{I}_j$ and candidate keyframes. We divide the online image into 64 blocks, denoted as $\{B_i|i = 1, 2, \ldots, 64\}$. Fig. 3 gives an illustration. Each of them contains a list of features that are sorted by their DoG strength. Ideally, if each block contributes one feature match, we will have a total of 64 matched feature pairs, which are enough for robust camera pose estimation. However, in practice, not all blocks can make it. We thus use the method described in Algorithm 4 to find matches that are well distributed in the image. The detailed explanation is as follows.

In the beginning, since all blocks in the online image do not have matched features, we sort them according to the number of features. For each feature $\mathbf{x}_j$ in block B_i, we use the ANN method [3] to search for 10 most similar features in terms of the distance of descriptors from each candidate keyframe k, denoted as $\{\mathcal{N}_s^k(\mathbf{x}_t)|s = 1, \ldots, 10\}$. KD-trees are used to speed up searching in this process. Then we employ the following 2NN heuristic proposed by [35] to measure the matching confidence:

$$c = \frac{||\mathbf{p}(\mathcal{N}_1^k(\mathbf{x}_j)) - \mathbf{p}(\mathbf{x}_j)||}{||\mathbf{p}(\mathcal{N}_2^k(\mathbf{x}_j)) - \mathbf{p}(\mathbf{x}_j)||}. \tag{6}$$

c measures the global distinctiveness. Cases that c is smaller than a threshold ε, where $\varepsilon = 0.7$ in experiments, indicate no ambiguously similar features in matching. We thus take $\mathcal{N}_1^k(\mathbf{x}_j)$ as the match of $\mathbf{x}_j$. The search continues until a suitable match is found or all the features in B_i are visited (i.e., for any feature $\mathbf{x}_j \in B_i$, $C^1(\mathbf{x}_j) = 1$). This matching step refers to step 2 in Algorithm 4.

After the first-pass matching, we estimate the fundamental matrix associating $\tilde{I}_t$ with each candidate keyframe using RANSAC [20]. The results are used to reject outliers among all matches. If there are less than N (N is usually set to 50–100 in our experiments) remaining inliers, the second-pass matching is performed.

Among the reference features that do not match to those in the online frame, many are simply because the 2NN heuristic is not satisfied. So in the second pass, we rematch a few of these indistinctive features using a *local* 2NN heuristic constrained by epipolar geometry.

First-pass matching yields the fundamental matrix between $\tilde{I}_j$ and each candidate keyframe k. For each feature $\mathbf{x}_j$ in the online frame, we screen out the reference features $\{\mathcal{N}_s^k(\mathbf{x}_j)|s = 1, \ldots, 10\}$ whose distance to the epipolar line, corresponding to $\mathbf{x}_j$, is larger than 2 pixels. The left-over features are denoted as $\{\tilde{\mathcal{N}}_s^k(\mathbf{x}_j)|s = 1, \ldots\}$. If feature $\mathbf{x}_j$ has its correspondences, they are very likely to be among $\tilde{\mathcal{N}}^k(\mathbf{x}_j)$s. We thus define the *local* 2NN heuristic

$$\frac{||\mathbf{p}(\tilde{\mathcal{N}}_1^k(\mathbf{x}_j)) - \mathbf{p}(\mathbf{x}_j)||}{||\mathbf{p}(\tilde{\mathcal{N}}_2^k(\mathbf{x}_j)) - \mathbf{p}(\mathbf{x}_j)||} < \varepsilon. \tag{7}$$

If one feature in the candidate keyframes satisfies (7) and has $||\mathbf{p}(\mathbf{x}_j) - \mathbf{p}(\tilde{\mathcal{N}}_1^k(\mathbf{x}_j))|| < \varsigma$where $\varsigma$is a threshold, we regard $\tilde{\mathcal{N}}_1^k(\mathbf{x}_j)$ and $\mathbf{x}_j$ as matchable. This strategy refers to step 5 in Algorithm 4.

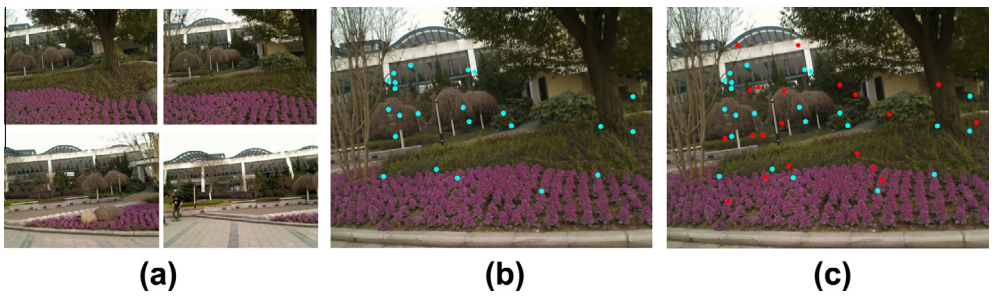

Fig. 4. Two-pass keyframe-based matching. (a) Selected candidate keyframes. (b) The matched features in the first pass. 23 matches are obtained. (c) Our two-pass matching yields 43 matches. The new ones are shown in red. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

It can robustly and faithfully add back feature matches, as demonstrated in Fig. 4.

The above steps repeat until N feature matches are found or all features $\mathbf{x}$ in $\widetilde{I}_t$ have been visited (i.e., $C^1(\mathbf{x}) = 1$ and $C^2(\mathbf{x}) = 1$). Finally, based on the matches, 2D (features) – 3D (reference points) correspondences are established, which make the camera pose be easily computed by the method of [45].

6. Reference map extension with online keyframe selection

With the above keyframe-based camera tracking method, we can reliably track the camera poses if the camera moves around the place that has been captured in offline stage. However, if the camera moves into a new place, or the appearance changes too much due to lighting variation, the camera pose will not be accurately estimated. In order to address this problem, we also involve a parallel tracking and mapping module to online reconstruct new 3D features and extend the reference map. In the PTAM system [29], it needs to project each of the map features to current frame via a prior motion model for locally searching for the correspondence, which is, however, inappropriate for a large-scale scene as it is limited to work on a small scene with thousands of scene points. In contrast, our system can always perform real-time global localization and camera tracking even in a very large scene.

Algorithm 5. Online keyframe selection with 3D recontruction

1. Keyframe verification:
   for each successfully tracked input frame **do**
   if it simultaneously satisfies the following conditions:
   (a) The number of matched 3D feature >20;
   (b) The RMSE of re-projections is less than a threshold;
   (c) The distance to the nearest keyframe is larger than a threshold;
   then
   the frame is considered as a keyframe candidate;
   else
   return false;
   end if
   end for
2. Find 4 nearest keyframes:
   (a) Set $T = \{T_0, T_1, \ldots, T_K\}$, where K is the number of keyframes and all elements are initialized to 0;
   (b) **for** each matched feature $\mathcal{X}$ in keyframe candidate **do**
   for each frame index i in $f(\mathcal{X})$ **do**
   set $T_i = T_i + 1$;
   end for
   end for
   (c) Select the top 4 keyframes with greatest T_i;
3. Triangulate new reference features:
   for each selected candidate keyframe **do**
   (a) Extract about 400 SIFT features from the input frame;
   (b) Match with the 4 most nearest keyframes;
   (c) Triangulate the new matched features;
   if the number of successfully triangulated features >20, **then**
   the input frame is considered as a new keyframe;
   else
   return false;
   end for
   end for

While reconstructing new 3D features, some online images will be selected as keyframes for online extending the scene representation and tracking ability. The criteria of selecting online keyframes could be similar to that introduced in Section 4. However, if we directly employ the method introduced in Section 4 to select keyframes, the computation will be too expensive for real-time performance. So, we employ a simpler but effective online keyframe selection method.

The algorithm is depicted in Algorithm 5. If an input frame satisfies the following conditions, it may be a new keyframe. First, there are sufficient number of matched 3D features (no less than 30), and the RMSE of the re-projections is small. Second, the input frame is not very close to any of the existing keyframes. If the input frame satisfies the above conditions, we will further select the four keyframes which have most overlap with the input frame. Then we extract more SIFT features (e.g. 400) from the input frame, and match them with these 4 nearest keyframes. If the number of matched features is larger than a threshold (20 in our experiments), the input frame will be considered as a keyframe. With the new matched features and the camera poses of keyframes, we can easily triangulate the 3D locations. The new reconstructed reference features will be included into the leaf nodes of the vocabulary tree (similar to [41]). Specifically, we maintain two vocabulary trees, where one is active for real-time keyframe recognition, and the other is waiting for updating. Once a new keyframe is inserted, we immediate update the inactive vocabulary tree with the newly reconstructed reference features, and change its status from inactive to active. The original active vocabulary tree then becomes inactive and wait for update. The newly selected keyframe will be added to the keyframe set F. Then, the new keyframe can be treated in the same way as the offline keyframes during keyframe-based matching, and the reference map is extended. After adding the new keyframe, we immediately employ a local bundle adjustment to refine the corresponding 3D structure of these 5 keyframes. A global bundle adjustment may be employed to further refine the estimated 3D structure if necessary. It should be noted that the above module runs on a separated thread, which does not influence the tracking speed.

Algorithm 6. Add new keyframe to the map

1. All new reference features are assigned to the leaf nodes of the vocabulary tree using the method in the step 2 of Algorithm 3;
2. Update the weights of nodes which has new reference features;
3. Add the new keyframe to the keyframe set F;
4. Employ a bundle adjustment to refine the 3D structure of the new keyframe and its 4 nearest keyframes.

7. Implementation

In this section, we provide more details about implementation.

7.1. Parallel computing

Table 1 shows the time spent in different steps for one input frame of the "Campus" example. The running times are tested in a desktop PC with an Intel Core2 Quad Q9550 @ 2.83 GHz CPU and a GeForce GTX 295 display card. We use the publicly available SiftGPU implementation [51] to accelerate feature extraction. For further speedup, we can first downscale the online image (e.g. from resolution 640×480 to 320×240) and then extract SIFT features

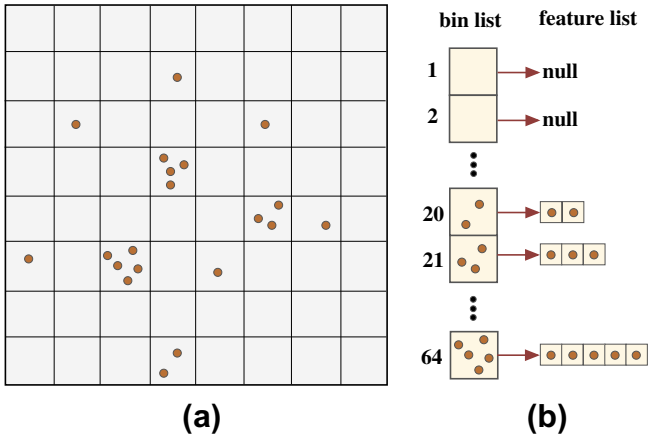

Fig. 3. Matching blocks. (a) One image is evenly divided into 64 blocks, each contains a few features sorted by the DoG values. (b) The sorted blocks with respect to the number of features.

for online camera tracking. This strategy can significantly reduce the running time without much influencing the system robustness. The running time is about 23 ms per frame (320 × 240 pixels) with 300 extracted features, which is enough for online feature matching. Besides GPU acceleration, we also applies multi-core CPUs to improve the performance. In our system, the SIFT extraction (on GPU) and matching (on CPU) run on *each* frame. This makes our system able to handle fast camera motion and very robust. Our framework contains two parallel hierarchies – that is, the inter-frame and intra-frame computations.

For the inter-frame process, we assign the computation tasks for different frames to separate threads. Therefore, the frame rate can easily be several times higher than using a single-core CPU. However, the system latency (i.e. the elapsed time between capturing and rending a frame) is not reduced because the total computation time for each frame does not decrease. To tackle it, we assign feature matching to multiple threads, as the features can be matched independently and simultaneously on multiple threads. With this intra-frame parallelism, the average matching time and latency can be further reduced. Fig. 1 illustrates the workflow of our parallel computing system. All the modules are connected and synchronized by thread-safe buffers. Even with 3D rendering in the augmented reality, our system can yield real-time performance.

7.2. Utilizing temporal information

Given an incoming image, our keyframe-based tracking scheme can quickly compute the candidate keyframes, and use them to estimate the camera pose. If the input is an online video sequence, while estimating the camera pose for each frame, the information of the past frames can be utilized.

In the keyframe recognition step, we introduce the following method to speed up search of candidate keyframes for image $\widetilde{I}_t$ with previously computed results for frame $\widetilde{I}_{t-1}$. The basic idea is that if a keyframe I_k has the most features matching to those in $\widetilde{I}_{t-1}$, it is quite possible that I_k and $\widetilde{I}_t$ also share many common features. We thus search for 10 keyframes closest to I_k with respect to the distance between the computed camera centers, then 3 of them that are with the most similar orientations, along with I_i, are used for online feature matching with $\widetilde{I}_t$. This process continues until the camera information and matching result in the previous frame are not available (e.g., in case of camera lost). Then, we resort to the method described in Section 5.2 for candidate keyframe searching.

Similar strategies are also used to improve the matching efficiency. Since the matched features in frame $\widetilde{I}_{t-1}$ already find their 3D locations for camera pose estimation (as described in Section 5.3), we employ the KLT method [36] to search for the correspondences of these features in $\widetilde{I}_t$ and similarly assign them the 3D positions. However, due to occlusion, noise or out-of-view, feature dropout inevitably occurs. In these cases, keyframe-based matching is still employed to maintain a stable number of feature matches. Compared to using only the keyframe-based matching, utilizing the temporal information can effectively alleviate the feature 'dropout' problem.

Finally, the estimated camera poses of past frames are used as in the jittering reduction method of [45] to further improve the robustness of camera tracking.

Table 1
Processing time per frame with a single thread. $\mathcal{K}$ is the number of the candidate keyframes.

Module	Average processing time (ms)
Feature extraction (resolution: 320 × 240)	≈23
Keyframe recognition	≈2
Keyframe-based matching	$\approx 4 \times \mathcal{K}$
Temporal matching	≈8
Camera pose estimation	≈5

8. Experimental results

We have conducted experiments with some challenging live video sequences. The reference and live frames are captured by a Logitech Quick-Cam Pro 9000 or C905 web camera or a Sony HDR XR550 camera.

8.1. Large-scale outdoor examples

We first show an outdoor example in Fig. 5. This example is very challenging for real-time camera tracking due to the large scale repeated structures. Fig. 5(a) shows the recovered 72,616 sparse 3D points in the offline stage. The number of the reference images is 1913. Fig. 5(c) shows the selected keyframes in our method. By our optimization, they do not significantly overlap and their coverage is almost the entire space, as shown in Fig. 5(b). The reference frames, captured live frames, and real-time rendering results are shown in the supplementary video.

Table 2 shows how different λs influence the keyframe selection. It can be observed that if we select 123 keyframes, more than 95% of the reference feature tracks (corresponding to different 3D points) in all reference images are included in the keyframes. Even with only 33 keyframes, about 42.17% reference feature tracks can be maintained. The keyframe sparsity makes online feature matching robust and fast. In this example, we set $\lambda = 2.0$ as it can maintain sufficient features. Some of the keyframe images seem very similar to others, e.g. the first 3 keyframes in the first row, and the last 2 keyframes. As a matter of fact, they contain different features because of image noises and the imperfect repetition of feature extraction. Our keyframe selection is rather efficient, only requiring 11 s for this example.

Our keyframe recognition is very efficient, which spends only 2 ms even with a single working thread. Fig. 6 shows the performance comparison. Compared to the appearance-vector-based method [38], our running time is much less. It is also less variant to the change of the keyframe numbers. Even using all the reference images (thousands of images) as keyframes, our keyframe recognition only requires a few milliseconds. However, selecting a subset of images as keyframes is still necessary due to the following two reasons. First, we need to build a KD tree for each keyframe.

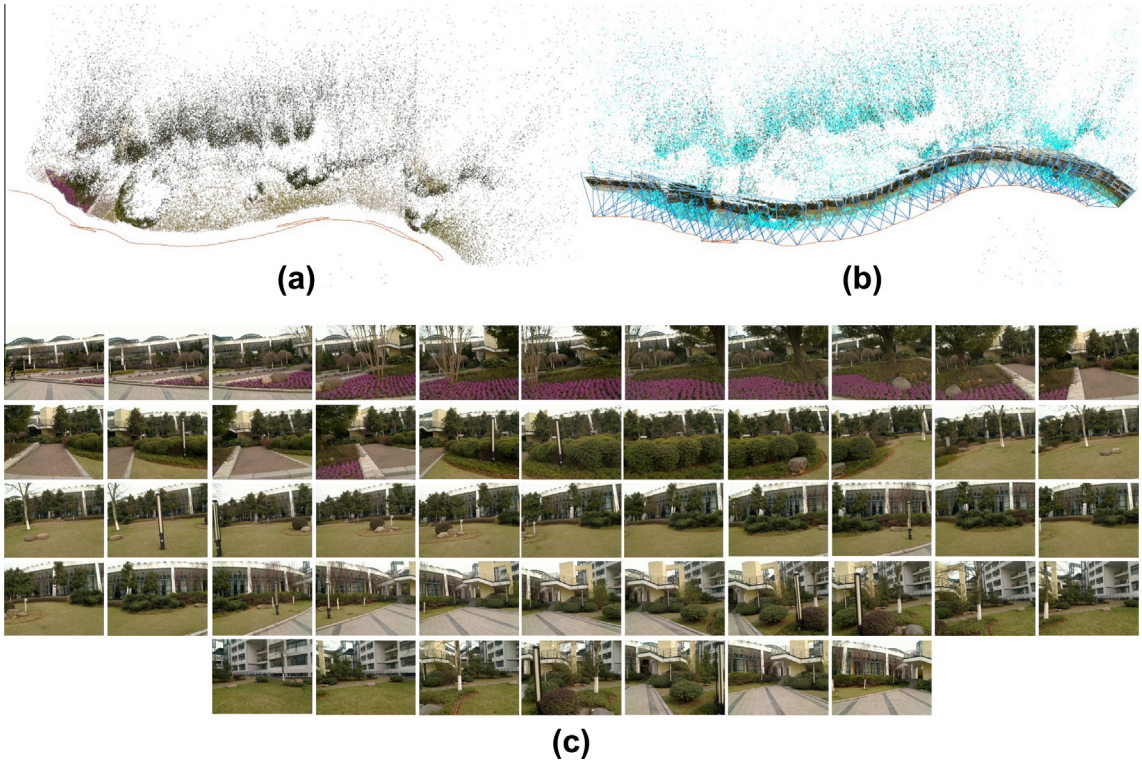

Fig. 5. The recovered reference 3D points and the selected keyframes for the outdoor "Campus" example. (a) The computed 3D points in the offline stage by SfM. (b) The computed keyframes viewed in 3D. The cyan points represent the 3D points included in at least one keyframe. (c) The selected keyframes.

Table 2
The statistics of feature completeness and energies of E_c and E_r with different λs and keyframe numbers.

λ	Keyframe number	E_c	E_r	Feature completeness (%)
0.1	123	0.020207	0.584265	95.7998
1.0	65	0.194271	0.142999	70.2325
2.0	51	0.289593	0.071128	58.4981
5.0	33	0.443603	0.022406	42.1656
10	24	0.558724	0.006362	31.1694
100	12	0.739066	0.000220	16.4674

Therefore, too many keyframes will require a significant memory space. Second, if we choose all reference images as keyframes, there will be significant overlapping among neighboring keyframes. The selected candidate keyframes by keyframe recognition method will contain almost identical and similar features, which will make online feature matching inefficient and unreliable.

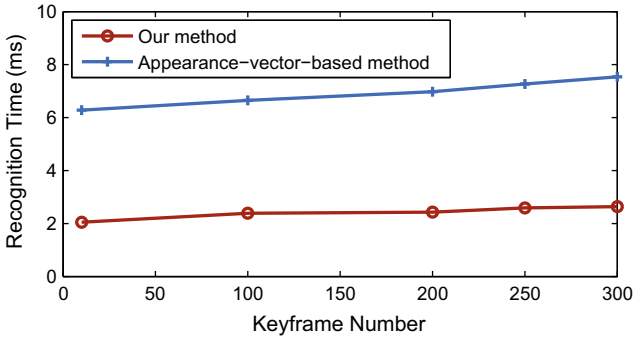

Fig. 6. Time spent in keyframe recognition. The computation of the appearance-vector-based method [38] grows linearly with the keyframe number while our method keeps it almost constant. The total running time of our method is much less.

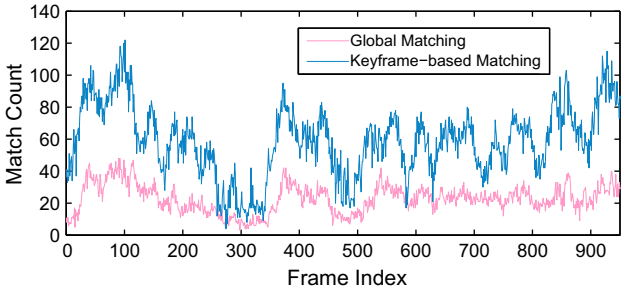

Fig. 7. Comparison of global matching and our keyframe-based matching. For fair evaluation and comparison, we do not limit the maximum number of matches. Even without using the temporal information, our keyframe-based method yields much more reliable matches than global matching.

We also compare our keyframe-based matching with the method of [45]. The latter one constructs a single KD tree for all reference features. Each feature in a live frame is compared with those included in the KD tree. We name this scheme *global matching* because it relies on the global distinctiveness of features. Fig. 7 compares the real-time matching quality in the indoor cubicle example. It is measured by the number of correct matches in processing one online frame. It is noticeable that our keyframe method yields much more reliable matches than the global one. The matching complexity of each feature is $O(\log M)$ using the global matching, where M is the total number of features. For our keyframe-based matching, it is reduced to $\mathcal{K} \cdot O(\log m)$, where m is the average number of features in each keyframe and $\mathcal{K}$ is the number of the candidate keyframes.

For global matching, the computation time grows with M. But for our keyframe-based method, the computation time is much smaller and does not grow with the total number of features. In

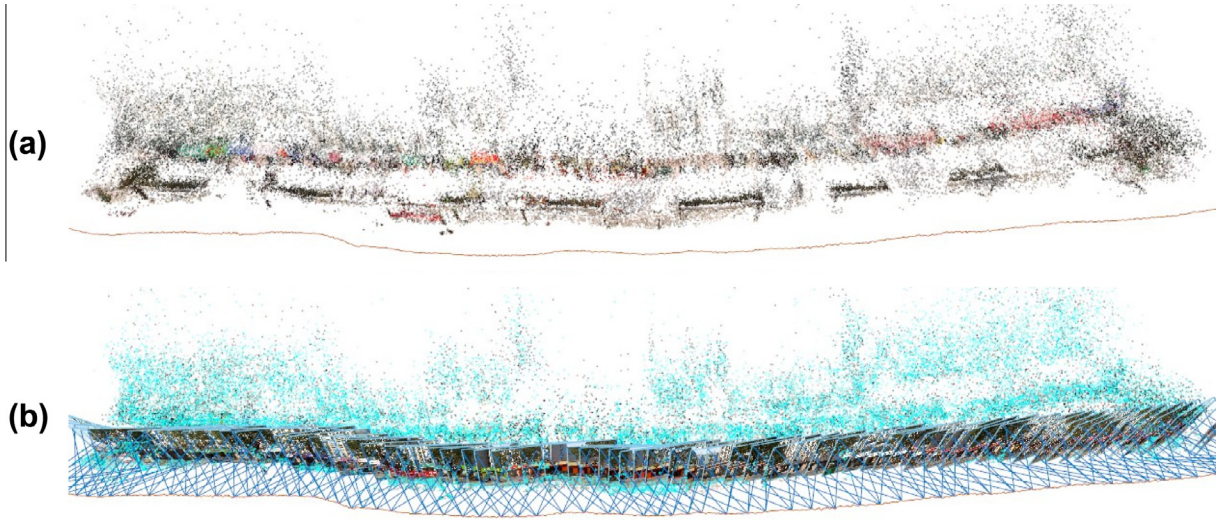

Fig. 8. The recovered 3D points and the selected keyframes of the "Street" example. (a) The computed 3D points corresponding to the computed reference features in the offline stage. (b) The selected keyframes viewed in 3D. The cyan dots denote the 3D points covered by the keyframes.

Fig. 9. The online camera tracking results of the "Street" example. Top: online images with the inserted virtual objects. Bottom: the computed candidate keyframes.

our experiments, for each online frame, we extract about 300 SIFT features. The global matching time is about 45 ms with a single working thread. Our method only uses 16 ms with $\mathcal{K} = 4$ (single working thread), and obtains high quality matches, as shown in Fig. 7. With the temporal information, the number of the feature matches is further increased.

Fig. 8 shows the "Street" example. It contains *3,385* reference images. The camera moves along a street and captures several buildings. The total moving distance is about 200 m. Fig. 8(a) shows the recovered 72,004 3D points. The selected 76 keyframes are shown in Fig. 8(b). The online tracking results are shown in Fig. 9. Readers are referred to our supplementary video for more details.

8.2. Comparison with other state-of-the-art methods

There has been a considerable amount of literature on camera tracking, and on related topics such as landmark recognition, location recognition, etc. Here, we discuss the differences of our method with two recent comparable works [26,33]. The comparisons are summarized in Table 3. Since these two methods do not allow to online reconstruct new 3D features, we also turn off the online map extension module for fair comparison. For fast location recognition, Irschara et al. [26] proposed to use a greedy algorithm to solve the *view cover* problem. Different from our method, they did not explicitly model the energy function, instead they tried to select the views which intersect with as many views as possible. The resulting views along with the point set are similar to the selected keyframes in our method, which contain as many features as possible, and the feature tracks should have long track length. This method also matches the input image to the closest views determined by vocabulary tree for camera location recognition, but do not control the number of extracted features and utilize the temporal information for robust real-time camera tracking of video streams. It extracts about 1600 features from each input frame, and employ a RANSAC procedure to verify the k top ranked keyframes. Due to the large number of extracted features, the execution time of matching each keyframe is about 25 ms (tested on a Intel Pentium D 3.2 GHz with a GeForce GTX 280 display card in their paper).

In our system, the number of extracted features is restricted to about 300, and the execution time of testing each keyframe is only 4 ms. Although fewer features are used, sufficient 2D–3D correspondences can still be obtained by two-pass keyframe-based matching with temporal tracking for robust camera pose estimation.

In the dual work, Li et al. [33] proposed another method to select a *seed point* set to represent a huge number of feature points. In each selection iteration, they choose the feature point which covers the most frames. There is not any corresponding notion in [33] with keyframe, so efficient keyframe-based matching cannot be used, and the proposed prioritized point-to-feature matching

Table 3
Comparison with other methods.

Methods	Compressed representation	Vocabulary tree	Keyframe-based two-pass matching
Irschara et al. [26]	Cover views	Yes	No
Li et al. [33]	Seed points	No	No
Our method	Keyframes	Yes	Yes

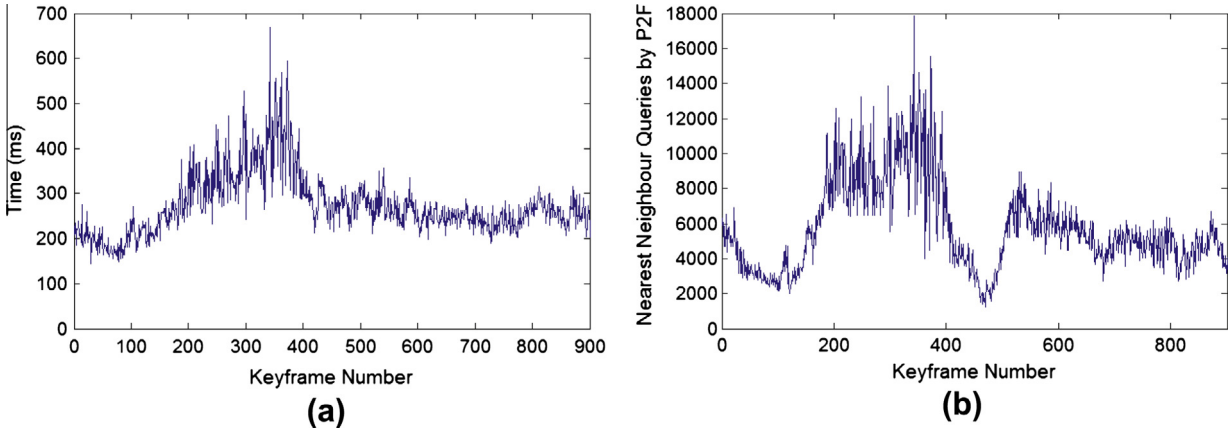

Fig. 10. Matching time of Li et al. [33]'s method for the "Campus" example. (a) The matching time of each input frame. (b) The Nearest Neighbour queries of Point-to-Feature matching method. About 100 matches are required to locate the camera, the average number of NN queries of P2F method is about 5658, and the average matching time is about 270 ms.

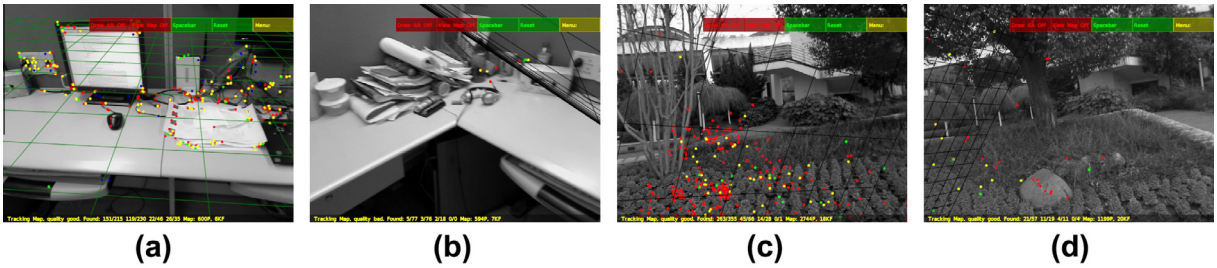

Fig. 11. The camera tracking results of PTAM for the indoor cubicle and outdoor "Campus" examples. (a,b) Two online frames in the indoor cubicle example. (c,d) Two online frames in the outdoor "Campus" example. The large-scale space and fast camera movement make it difficult for PTAM to produce very good camera tracking results.

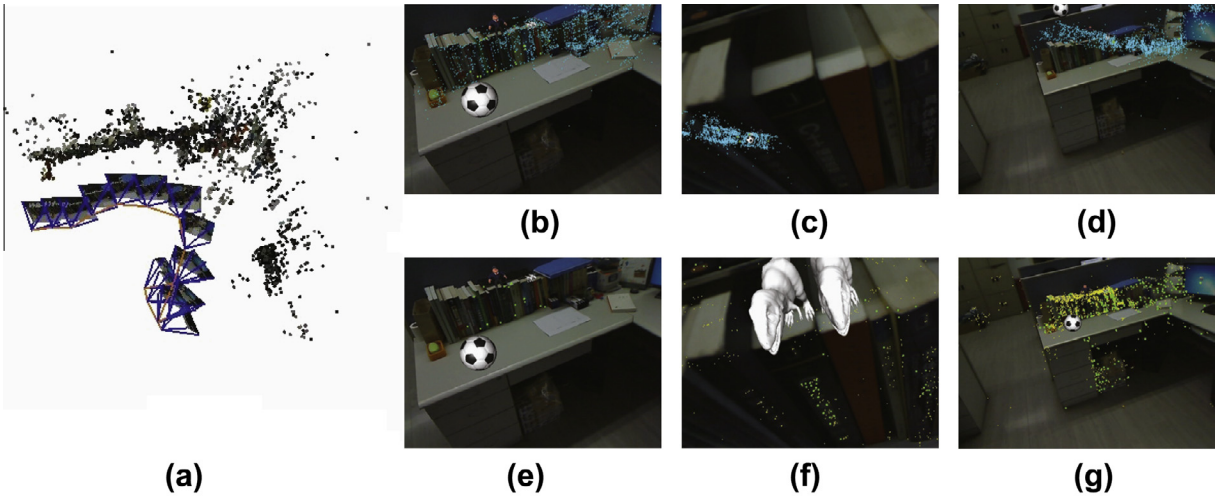

Fig. 12. An indoor example with/without online map extension. (a) The offline reconstructed reference 3D points with selected keyframes. (b–d) The augmented result without online map extension. (e–g) The augmented result with online map extension.

schema is time-consuming. For the "Campus" example in Fig. 5, we construct the seed points as 20-covers with 294 features, and the compressed points as 400-covers with 18,317 features. It takes about 150–600 ms (in our implementation) to find sufficient number of inlier matches for robust camera pose estimation (Fig. 10), which is impractical for real-time application.

Klein and Murray [29] employed online bundle adjustment (BA) with parallel computing to avoid offline 3D reconstruction.

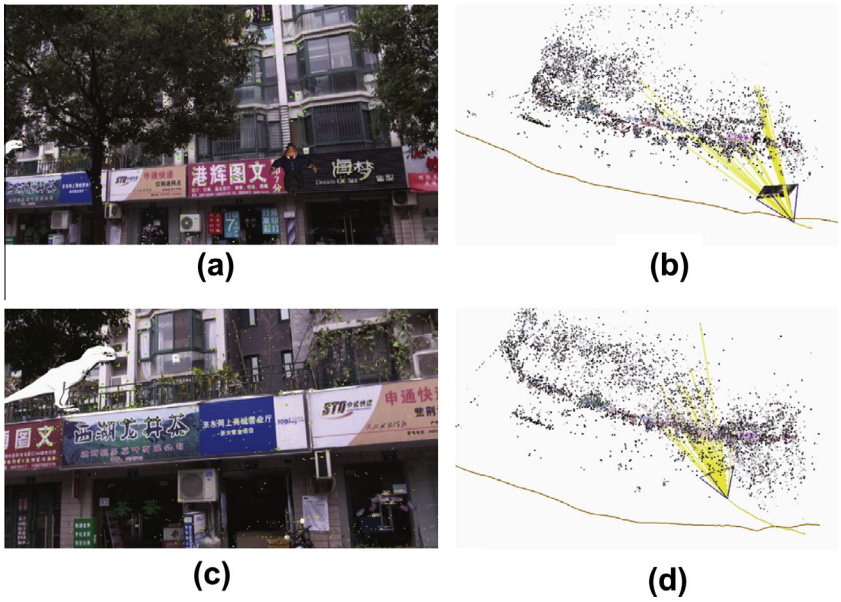

Fig. 13. An outdoor example with online map extension. (a,b) The augmented result of one frame and its corresponding tracking result viewed in 3D. The red curve is the offline camera trajectory, and the yellow curve is the online camera trajectory. (c,d) The tracking and augmented result of another frame. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

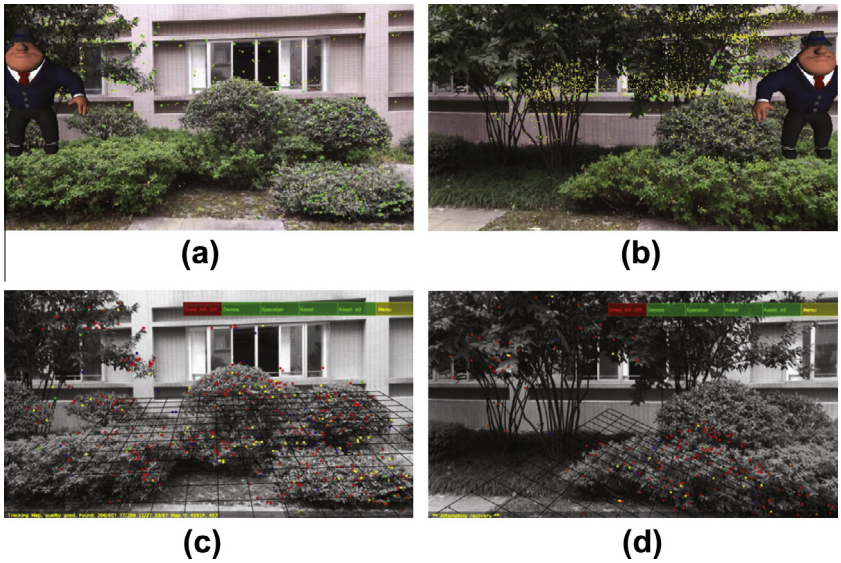

Fig. 14. Comparison with PTAM. (a,b) Our real-time tracking result without offline preprocessing. (c,d) The tracking result by PTAM.

This strategy, albeit effective in a small-scale space, is not suitable for camera tracking in a large scale one because SfM requires computationally-expensive global BA which cannot be accomplished online. We have experimented with sequences using the publicly accessible PTAM code,[4] where the input frame rate is set to 5fps to give enough time to the local BA thread, and each sequence is repeated for the global BA to converge. Even with these operations, we found that PTAM only succeeded in tracking the first half of the indoor cubicle sequence. Fig. 11 shows the results of PTAM for the indoor cubicle and outdoor "Campus" examples. The large-scale space and fast camera movement makes it difficult for PTAM to produce very good camera tracking results.

[4] http://www.robots.ox.ac.uk/~gk/PTAM/.

8.3. Results comparison with and without online map extension

As we know, traditional GL methods typically assume that the visited places have been sufficiently captured and reconstructed. However, if the camera moves to a new place which is not sufficiently covered by reference images, the camera pose estimation will fail. Fig. 12 shows an indoor example. If we did not use online map extension module, camera tracking would fail while the camera came very close to the book, as shown in Fig. 12(c). The reason is that the scales or viewing angles between online and reference images are too different, which eventually makes global localization fail although scale-invariant features are used. With online map extension, the camera pose can be robustly estimated for this challenging case, as shown in Fig. 12(f). Fig. 13 shows an outdoor example. As can be seen, the online camera significantly deviated from the offline camera trajectory, and came very close

to the building. With online map extension, the camera pose can be robustly estimated by our system, which is, however, very difficult for traditional GL methods. The scene scale of this example is also out of the scope of tracking capability for traditional SLAM systems, such as PTAM.

It should be noted that our system can work even without offline preprocessing. In this extreme case, our system actually only uses the online map extension module with keyframe-based tracking to recover the camera pose in real-time. Compared to PTAM, our online map extension is more robust and can handle a larger scene. Fig. 14 gives a result comparison. Because PTAM projects each of the map features to current frame via a prior motion model for robust matching, which seriously restricts its tracking ability and scope. In contrast, our keyframe-based tracking scheme does not have this limitation, and can handle a larger scale scene. Please refer to our supplementary video for more detailed comparison.

9. Conclusion and discussion

We have presented an effective keyframe-based real-time camera tracking system. In the offline stage, keyframes are selected from the captured reference images based on a few criteria. For quick online matching, we introduce an efficient keyframe candidate searching algorithm to avoid exhaustive frame-by-frame matching. Our experiments show that a small number of candidate reference images are sufficient for achieving high coverage of features in the input images. A two-pass keyframe-based matching is employed to find sufficient and evenly distributed feature matches, so that general camera tracking can be achieved. Compared to global matching, our method not only simplifies feature matching and speeds it up, but also minimizes the matching ambiguity when the original images contain many non-distinctive features. Online map extension module is also proposed to significantly extend the ability of global localization and camera tracking, so that our system can still work well even the camera moves to a new place which is not covered by offline reference images. A variety of challenging examples demonstrate that the proposed system outperforms state-of-the-art methods.

Acknowledgments

The authors thank the associate editor and all the reviewers for their constructive comments. They also thank Wei Tan for his help in producing part of the results. This work is partially supported by the 973 Program of China (No. 2009CB320804), National Science and Technology Support Plan Project (No. 2012BAH35B02), NSF of China (No. 61103104), the Specialized Research Fund for the Doctoral Program of Higher Education of China (No. 20110101130011), the Fundamental Research Funds for the Central Universities, Zhejiang Provincial Natural Science Foundation of China (Grant No. Z1111051), and a grant from the Research Grants Council of Hong Kong (Project No. 412911).

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.cviu.2013.08.005.

References

[1] A. Angeli, A.J. Davison, Live feature clustering in video using appearance and 3d geometry, in: British Machine Vision Conference (BMVC), 2010.
[2] A. Angeli, D. Filliat, S. Doncieux, J.-A. Meyer, A fast and incremental method for loop-closure detection using bags of visual words, in: IEEE Transactions on Robotics Special Issue on Visual Slam, October 2008.
[3] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, A.Y. Wu, An optimal algorithm for approximate nearest neighbor searching fixed dimensions, J. ACM 45 (6) (1998) 891–923.
[4] H. Bay, T. Tuytelaars, L.J.V. Gool, Surf: speeded up robust features, in: ECCV (1), 2006, pp. 404–417.
[5] R.O. Castle, G. Klein, D.W. Murray, Video-rate localization in multiple maps for wearable augmented reality, in: Proc 12th IEEE Int. Symp. on Wearable Computers, Pittsburgh PA, 2008.
[6] D. Chekhlov, W. Mayol-Cuevas, A. Calway, Appearance based indexing for relocalisation in real-time visual slam, in: 19th Bristish Machine Vision Conference, BMVA, September 2008, pp. 363–372.
[7] D. Chekhlov, M. Pupilli, W. Mayol, A. Calway, Robust real-time visual SLAM using scale prediction and exemplar based feature description, in: CVPR, 2007, pp. 1–7.
[8] M. Chli, A.J. Davison, Automatically and efficiently inferring the hierarchical structure of visual maps, in: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2009, pp. 387–394.
[9] O. Chum, J. Philbin, J. Sivic, M. Isard, A. Zisserman, Total recall: automatic query expansion with a generative feature model for object retrieval, in: ICCV, 2007, pp. 1–8.
[10] L.A. Clemente, A.J. Davison, I.D. Reid, J. Neira, J.D. Tardós, Mapping large loops with a single hand-held camera, in: Robotics: Science and Systems, 2007.
[11] A.I. Comport, E. Marchand, M. Pressigout, F. Chaumette, Real-time markerless tracking for augmented reality: the virtual visual servoing framework, IEEE Trans. Visual. Comput. Graph. 12 (4) (2006) 615–628. July–August.
[12] M. Cummins, P. Newman, Fab-map: probabilistic localization and mapping in the space of appearance, Int. J. Rob. Res. 27 (6) (2008) 647–665.
[13] M.J. Cummins, P. Newman, Accelerated appearance-only slam, in: ICRA, 2008, pp. 1828–1833.
[14] A.J. Davison, Real-time simultaneous localisation and mapping with a single camera, in: ICCV, 2003, pp. 1403–1410.
[15] A.J. Davison, I.D. Reid, N.D. Molton, O. Stasse, MonoSLAM: real-time single camera SLAM, IEEE Trans. Pattern Anal. Machine Intell. 26 (6) (2007) 1052–1067.
[16] Z. Dong, G. Zhang, J. Jia, H. Bao, Keyframe-based real-time camera tracking, in: ICCV, 2009, pp. 1538–1545.
[17] E. Eade, T. Drummond, Scalable monocular slam, in: CVPR (1), 2006, pp. 469–476.
[18] E. Eade, T. Drummond, Unified loop closing and recovery for real time monocular slam, in: British Machine Vision Conference (BMVC), 2008.
[19] C. Estrada, J. Neira, J.D. Tardos, Hierarchical slam: real-time accurate mapping of large environments, Trans. Robot. 21 (4) (2005) 588–596.
[20] M.A. Fischler, R.C. Bolles, Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography, Commun. ACM 24 (6) (1981) 381–395.
[21] V. Gay-Bellile, P. Lothe, S. Bourgeois, E. Royer, S.N. Collette, Augmented reality in large environments: Application to aided navigation in urban context, in: 9th IEEE International Symposium on Mixed and Augmented Reality (ISMAR), 2010, pp. 225–226.
[22] A.A. GerardoCarrera, A.J. Davison, Lightweight slam and navigation with a multi-camera rig, in: European Conference on Mobile Robots, 2011.
[23] C. Gianluigi, S. Raimondo, An innovative algorithm for key frame extraction in video summarization, J. Real-Time Image Process. 1 (1) (2006) 69–88.
[24] R.I. Hartley, A. Zisserman, Multiple View Geometry in Computer Vision, second ed., Cambridge University Press, 2004. 052154051.
[25] Hauke Strasdat, Andrew J. Davison, J.M.M. Montiel, K. Konolige, Double window optimisation for constant time visual slam, in: Proceedings of International Conference on Computer Vision, 2011.
[26] A. Irschara, C. Zach, J.-M. Frahm, H. Bischof, From structure-from-motion point clouds to fast location recognition, in: CVPR, 2009.
[27] J. Stuehmer, S. Gumhold, D. Cremers, Real-time dense geometry from a handheld camera, in: Proceedings of the DAGM Symposium on Pattern Recognition, 2010.
[28] G. Klein, D. Murray, Parallel tracking and mapping for small AR workspaces, in: ISMAR 2007, November 2007, pp. 225–234.
[29] G. Klein, D. Murray, Improving the agility of keyframe-based slam, in: ECCV, vol. 2, 2008, pp. 802–815.
[30] T. Lee, T. Höllerer, Hybrid feature tracking and user interaction for markerless augmented reality, in: VR, 2008, pp. 145–152.
[31] V. Lepetit, P. Fua, Monocular model-based 3D tracking of rigid objects, Found. Trends. Comput. Graph. Vis. 1 (1) (2005) 1–89.
[32] V. Lepetit, P. Fua, Keypoint recognition using randomized trees, IEEE Trans. Pattern Anal. Mach. Intell. 28 (9) (2006) 1465–1479.
[33] Y. Li, N. Snavely, D.P. Huttenlocher, Location recognition using prioritized feature matching, in: Proceedings of ECCV, 2010.
[34] T. Liu, J.R. Kender, Optimization algorithms for the selection of key frame sequences of variable length, in: ECCV (4), 2002, pp. 403–417.
[35] D.G. Lowe, Distinctive image features from scale-invariant keypoints, Int. J. Comput. Vis. 60 (2) (2004) 91–110.
[36] B.D. Lucas, T. Kanade, An iterative image registration technique with an application to stereo vision, in: IJCAI, 1981, pp. 674–679.
[37] R.A. Newcombe, A.J. Davison, Live dense reconstruction with a single moving camera, in: CVPR, 2010.
[38] D. Nister, H. Stewenius, Scalable recognition with a vocabulary tree, in: CVPR, IEEE Computer Society, Washington, DC, USA, 2006, pp. 2161–2168.

[39] Y. Park, V. Lepetit, W. Woo, Multiple 3d object tracking for augmented reality, in: 7th IEEE/ACM International Symposium on Mixed and Augmented Reality, 2008 (ISMAR 2008), September 2008, pp. 117–120.
[40] M. Pollefeys, L.V. Gool, M. Vergauwen, F. Verbiest, K. Cornelis, J. Tops, R. Koch, Visual modeling with a hand-held camera, Int. J. Comput. Vis. 59 (3) (2004) 207–232.
[41] Rongrong Ji, Xing Xie, H. Yao, Y. Wu, W.-Y. Ma, Vocabulary tree incremental indexing for scalable location recognition, in: IEEE International Conference on Multimedia and Expo, 2008, pp. 869–872.
[42] F. Schaffalitzky, A. Zisserman, Automated location matching in movies, Comput. Vis. Image Understand. 92 (2–3) (2003) 236–264.
[43] G. Schindler, M. Brown, R. Szeliski, City-scale location recognition, in: CVPR, 2007.
[44] J. Sivic, A. Zisserman, Video google: a text retrieval approach to object matching in videos, in: ICCV, IEEE Computer Society, Washington, DC, USA, 2003, p. 1470.
[45] I. Skrypnyk, D.G. Lowe, Scene modelling, recognition and tracking with invariant image features, in: '04: Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality, IEEE Computer Society, Washington, DC, USA, 2004, pp. 110–119.
[46] N. Snavely, S.M. Seitz, R. Szeliski, Photo tourism: exploring photo collections in 3d, ACM Trans. Graph. 25 (3) (2006) 835–846.
[47] B.T. Truong, S. Venkatesh, Video abstraction: a systematic review and classification, ACM Trans. Multimedia Comput. Commun. Appl. 3 (1) (2007) 3.
[48] L. Vacchetti, V. Lepetit, P. Fua, Combining edge and texture information for real-time accurate 3D camera tracking, in: Third IEEE and ACM International Symposium on Mixed and Augmented Reality, Arlington, Virginia, November 2004, pp. 48–57.
[49] B. Williams, M. Cummins, J. Neira, P. Newman, I. Reid, J. Tardos, An image-to-map loop closing method for monocular SLAM, in: Proc. International Conference on Intelligent Robots and Systems, 2008.
[50] B. Williams, G. Klein, I. Reid, Real-Time SLAM Relocalisation, in: ICCV, 2007, pp. 1–8.
[51] C. Wu, SiftGPU: a GPU implementation of scale invariant feature transform (SIFT), 2007. <http://cs.unc.edu/ccwu/siftgpu>.
[52] G. Zhang, X. Qin, W. Hua, T.-T. Wong, P.-A. Heng, H. Bao, Robust metric reconstruction from challenging video sequences, in: CVPR, 2007.