CoSLAM: Collaborative Visual SLAM in Dynamic Environments

Danping Zou and Ping Tan, *Member*, *IEEE*

Abstract—This paper studies the problem of vision-based simultaneous localization and mapping (SLAM) in dynamic environments with multiple cameras. These cameras move independently and can be mounted on different platforms. All cameras work together to build a global map, including 3D positions of static background points and trajectories of moving foreground points. We introduce intercamera pose estimation and intercamera mapping to deal with dynamic objects in the localization and mapping process. To further enhance the system robustness, we maintain the position uncertainty of each map point. To facilitate intercamera operations, we cluster cameras into groups according to their view overlap, and manage the split and merge of camera groups in real time. Experimental results demonstrate that our system can work robustly in highly dynamic environments and produce more accurate results in static environments.

Index Terms—Visual SLAM, swarm, dynamic environments, structure-from-motion

1 Introduction

Many vision-based Simultaneous Localization and Mapping (SLAM) systems [23], [10], [18] have been developed and have shown their remarkable performance on mapping and localization in real time. Recent works took further steps to provide high-level scene understanding [20] or to improve the system accuracy and robustness, such as "loop closure" [16], "relocalization" [36], and dense depth map reconstruction [22]. These works have made cameras become more and more favorable sensors for SLAM systems.

Existing vision-based SLAM systems mainly focus on navigation in static environments with a single camera. However, the real world is full of moving objects. Although there are robust methods to detect and discard dynamic points by treating them as outliers [18], [5], conventional SLAM algorithms tend to fail when the portion of moving points is large. Further, in dynamic environments, it is often important to reconstruct the 3D trajectories of the moving objects [35], [38] for tasks such as collision detection and path planning. This 3D reconstruction of dynamic points can hardly be achieved by a single camera.

To address these problems, we present a collaborative visual SLAM system using multiple cameras. The relative positions and orientations between cameras are allowed to change over time, which means cameras can be mounted on different platforms that move independently. This setting is different from existing SLAM systems with a stereo camera [23], [25] or a multicamera rig [17] where all cameras are fixed on a single platform. Our camera configuration makes the system applicable to the following interesting cases: 1) wearable augmented reality [5], where multiple cameras are mounted on different parts of the body; 2) robot teams [4], [34], [1], where multiple robots work in the same environment and each carries a single camera because of limited weight and energy capacity, e.g., micro air vehicles (MAVs) [40].

In our system, we use images from different cameras to build a global 3D map. We maintain the position uncertainty of each map point by a covariance matrix, and iteratively refine the map point position whenever a new observation is available. This design increases the system robustness and accuracy in dealing with complex scenes. Further, we classify map points as dynamic or static at every frame by analyzing their triangulation consistency. False points caused by incorrect feature matching are also detected and removed. For robust localization in dynamic environments, we use both dynamic and static points to simultaneously estimate the poses of all cameras with view overlap. We divide cameras into groups according to their view overlap. Cameras within the same group share a common view and work collaboratively for robust mapping and localization. Groups can split or merge when cameras separate or meet.

Our system was tested in static and dynamic, indoor and outdoor scenes. Experimental results show that our method is more accurate and robust than existing single camera-based SLAM methods. Our system succeeds in highly dynamic environments and is able to reconstruct the 3D trajectories of moving objects. The system currently runs at approximately 38 ms per frame for three cameras. In the next section, we shall briefly review existing visual SLAM methods and discuss our work in detail.

2 Related Work

Visual SLAM with a single camera. There are mainly two types of methods for single camera-based visual SLAM. One is based on the structure-from-motion (SFM) technique

• *The authors are with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore 117576. E-mail: dannis.zou@gmail.com, eletp@nus.edu.sg.*

Manuscript received 5 Oct. 2011; revised 31 Mar. 2012; accepted 14 Apr. 2012; published online 27 Apr. 2012.
Recommended for acceptance by I. Reid.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number TPAMI-2011-10-0712.
Digital Object Identifier no. 10.1109/TPAMI.2012.104.

[15]. Royer et al. [26] first reconstructed a 3D map of the scene offline from a learning sequence, and then estimated the camera pose in real time by referring to that map. Mouragnon et al. [21] proposed a local bundle adjustment method so that mapping and pose update can run in nearly real time. Klein and Murray [18] put the time-consuming bundle adjustment into an asynchronous thread and made the system much faster.

The second type of methods model SLAM as a Bayesian inference problem and solve it through the Extended Kalman Filter [9], [10]. In [30], line features were used to complement point features to improve the matching robustness. Eade and Drummond [12] applied particle filter and a top-down approach to handle a relatively large number of landmarks (hundreds) more efficiently. To improve the robustness of SLAM, Williams et al. [36] proposed a relocalization method to recover the SLAM system from tracking failures.

Strasdat et al. [31], [33] compared both types of methods and concluded that the SFM-based methods produce more accurate results per unit computing time, while filter-based methods could be more efficient when processing resource is limited.

These methods often do not consider dynamic scenes. Some of them, such as [18], [5], detected and discarded dynamic points as outliers. However, this approach tends to fail when the portion of dynamic points is large. Some more recent methods, such as [24], [19], applied multibody SFM to deal with dynamic environments. However, this approach is only applicable to rigid moving objects, and the 3D reconstruction of moving points is up to a scaling ambiguity [24]. In comparison, we propose to solve the SLAM problem in dynamic scenes with multiple independently moving cameras. Our approach can reconstruct the full 3D of dynamic points within the scene map.

Visual SLAM with multiple cameras. Nister et al. [23] proposed a visual odometry system with a stereo rig. Their system was much like an SFM-based single camera SLAM system with an additional camera to generate map points at every frame. They also pointed out that the narrow baseline between stereo cameras could affect the map quality. To address this problem, Paz et al. [25] separated close and far 3D points and used far points to estimate camera rotation only. To obtain wider FOV, Kaess and Dellaert [17] mounted multiple cameras in a rig facing different directions to combine the advantages of omnidirection vision [37] and monocular vision. Castle et al. [5] used multiple cameras distributed freely in a static environment, where each camera was processed by an independent single camera-based SLAM system. A camera could be localized according to different maps by registering its feature points to the other map points.

These methods still focus on static scenes and do not take full advantage of multiple cameras. Further, the relative positions of their cameras are often fixed. In comparison, we allow cameras to move independently and achieve more flexible system design. For example, our cameras can be mounted on different robots for robot team applications.

SLAM in dynamic environments. Existing works on SLAM in dynamic environments mostly use filter-based methods and have been successfully applied to SLAM problems with sensors such as laser scanners [14], [38], [35] and radar systems [2], [3]. In this work, we aim to use cameras to address the SLAM problem. Compared with other sensors, cameras are passive, compact, and energy efficient, which has important advantages for microrobots with limited weight and energy capacity (such as MAVs [40], [1]). To the best of our knowledge, this work is the first visual SLAM solution in a dynamic environment with multiple cameras moving independently. This method could be applied to emerging swarm robotics applications [1], [27].

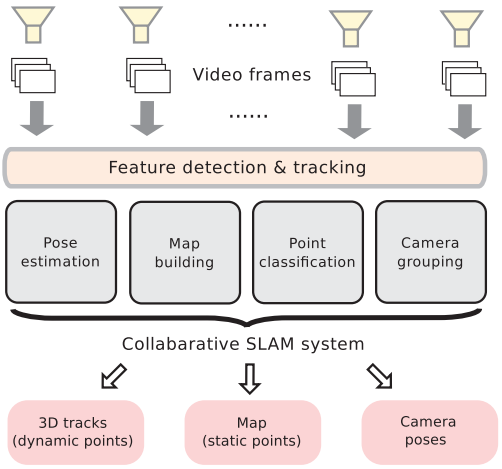


Fig. 1. CoSLAM system architecture.

3 System Overview

The intrinsic parameters of all our cameras are calibrated beforehand. Our collaborative SLAM system treats each camera as a sensor input, and incorporates all inputs to build a global map, and simultaneously computes the poses of all cameras over time. The system framework is illustrated in Fig. 1. The system detects and a tracks feature points at every frame and feeds them to the four SLAM components. We use the Kanade-Lucas-Tomasi (KLT) [28] tracker for both feature detection and tracking because of its good balance between efficiency and robustness. However, there is no restriction on using other feature detectors and trackers such as the "active matching" [6].

The four SLAM components are "camera pose estimation," "map building," "point classification," and "camera grouping." The main pipeline of our system follows conventional sequential structure-from-motion methods. We assume that all cameras look at the same initial scene to initialize the system. After that, the "camera pose estimation" component computes camera poses at every frame by registering the 3D map points to 2D image features. From time to time, new map points are generated by the "map building" component. At every frame, points are classified into different types by the "point classification" component. The system maintains the view overlap information among cameras throughout time. The "camera grouping" component separates cameras into different groups, where cameras with view overlap are in the same group. These groups could merge and split when cameras meet or separate. In the following section, we shall describe these four components in detail.

4 Camera Pose Estimation

Our system alternatively uses two different methods for camera pose estimation: intracamera pose estimation and intercamera pose estimation. In the former, each camera works independently, where tracked feature points from a camera are registered with static map points to compute its pose. In dynamic environments, the number of static map points could be small or the static points are distributed within a small image region, which can make the intracamera pose estimation fail. In such a case, we switch to the intercamera pose estimation method that uses both static and dynamic points to simultaneously obtain poses for all cameras.

4.1 Intracamera Pose Estimation

If the camera intrinsic parameters are known, the camera pose $\Theta = (\mathbf{R}, \mathbf{t})$ can be computed by minimizing the reprojection error (the distance between the image projection of 3D map points and their corresponding image feature points), namely,

$$\Theta^* = \arg\min_{\theta} \sum_i \rho(\|\mathbf{m}_i - \mathcal{P}(\mathbf{M}_i, \Theta)\|), \tag{1}$$

where $\mathcal{P}(\mathbf{M}_i, \Theta)$ is the image projection of the 3D point $\mathbf{M}_i$, $\mathbf{m}_i$ is the image feature point registered to $\mathbf{M}_i$, $\|\cdot\|$ measures the distance between two image points. i is an index of feature points. The M-estimator $\rho : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ is the Tukey biweight function [39] defined as

$$\rho(x) = \begin{cases} t^2/6 \left(1 - \left[1 - \left(\frac{x}{t}\right)^2\right]^3\right), & \text{if } |x| \le t, \\ t^2/6, & \text{otherwise.} \end{cases} \tag{2}$$

Assuming that the error of feature detection and tracking obeys a Gaussian distribution $\mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$, we set the threshold t in $\rho(\cdot)$ as 3σ. Equation (1) is minimized by the iteratively reweighted least squares (IRLS) method, where Θ is initialized according to the camera pose at the previous frame. At each iteration of the IRLS, the Levenberg-Marquart algorithm is used to solve the nonlinear least square problem, where Θ is parameterized in Lie algebra $se(3)$ as [32].

4.2 Intercamera Pose Estimation

When the number of visible static points is small or the static points are located in a small image region, the intracamera pose estimation is unstable and sometimes fails. Fortunately, points on moving objects give information about the relative camera poses. Fig. 2 provides such an illustration. The pose of camera A at the nth frame cannot be decided on its own since it only observes the moving object (the gray square). However, its relative pose with respect to the camera B can be decided. We can therefore use both static and dynamic points together to decide all camera pose simultaneously.

Actually, the 3D coordinates of dynamic points can only be computed when the camera poses are already known. Hence, our system in fact simultaneously estimates both the camera poses and the 3D positions of dynamic points. We formulate the intercamera pose estimation problem as a minimization of reprojection error,

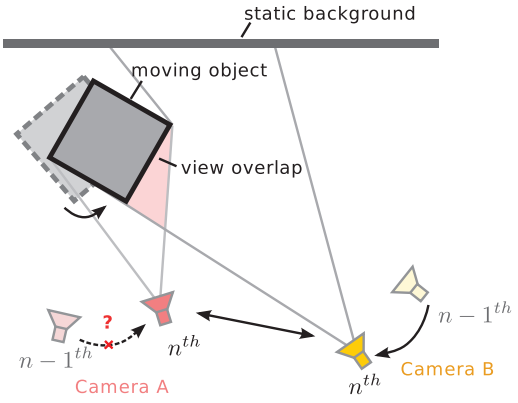


Fig. 2. The pose of camera A at the nth frame cannot be estimated from its previous pose since it observes only the moving object (the gray square). However, its relative pose with respect to camera B can be determined. So, the absolute pose of camera A can be computed.

$$\{\Theta_c\}^* = \arg\min_{\mathbf{M}_D, \{\Theta_c\}} \sum_c \left\{ \sum_{i \in S} v_i^c \rho(\|\mathbf{m}_i - \mathcal{P}(\mathbf{M}_i, \Theta_c)\|) + \sum_{j \in D} v_j^c \rho(\|\mathbf{m}_j - \mathcal{P}(\mathbf{M}_j, \Theta_c)\|) \right\}. \tag{3}$$

Here, c is an index of cameras, S and D are the set of "static" and "dynamic" map points. v_i^c represents the visibility of the ith map point at camera c (1 for visible, 0 otherwise).

The difference between the "intracamera pose estimation" and the "intercamera pose estimation" lies in the second term of (3), where the dynamic points are included in the objective function. The relative poses between cameras are therefore enforced by minimizing the reprojection error of the dynamic points. Hence, our system can determine the poses of cameras where few static points are visible. As the cameras need to have view overlap, we only apply intercamera pose estimation to cameras within the same group. We refer the reader to Section 6 for more details about camera grouping.

The optimization of (3) is also solved by the IRLS. The camera poses and 3D positions of dynamic points are initialized from the previous frame. We call intracamera pose estimation by default. We call the intercamera pose estimation only when the number of dynamic points are greater than that of static points or the area covered by the convex hull of static feature points is less than 20 percent of the image area.

5 Map Maintenance

Unlike previous SFM-based SLAM systems [18], [5], where only the 3D position is recovered for each map point, we further maintain the position uncertainty of each map point to help point registration and distinguishing static and dynamic points. Specifically, we recover a probability distribution of the map point position, which is represented by a Gaussian function $\mathcal{N}(\mathbf{M}_i, \Sigma_i)$. $\mathbf{M}_i$ is the triangulated position. The covariance matrix $\Sigma_i \in \mathbb{R}^{3\times3}$ measures the position uncertainty.

To facilitate computation, for each feature point we keep a pointer directing to its corresponding map point. Similarly, for each map point, we also keep pointers

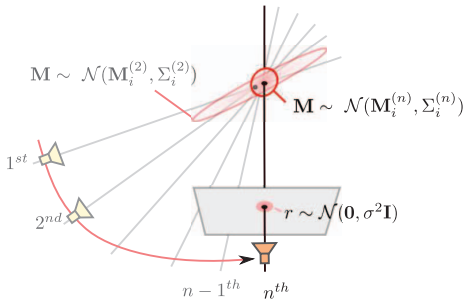


Fig. 3. Both the mean position and position covariance matrix are updated when a new observation is coming.

directing to its corresponding image feature points in each view, and store the local image patches centered at these feature points. We downsize the original input image to its 30 percent size and take a patch of 11×11 pixels. We further keep track of the frame numbers when a map point is generated or becomes invisible.

5.1 Position Uncertainty of Map Points

When measuring the uncertainty in map point positions, we only consider the uncertainty in feature detection and triangulation. In principle, we could also include the uncertainty in camera positions. We assume the feature detection error follows Gaussian distribution $\mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$. The position uncertainty of a 3D map point is described by the covariance computed as

$$\Sigma_i = (\mathbf{J}_i^T \mathbf{J}_i)^{-1} \sigma^2, \tag{4}$$

where $\mathbf{J}_i \in \mathbb{R}^{(2k)\times 3}$ is the Jacobian of the camera projection function that maps a 3D map point to its 2D image coordinates in all views, and k denotes the number of views used for triangulation.

When there is a new image observation $\mathbf{m}_i^{(n+1)}$ of a map point, we can quickly update its 3D position with Kalman Gain:

$$\mathbf{M}_i^{(n+1)} = \mathbf{M}_i^{(n)} + \mathbf{K}_i \left[\mathcal{P}_{n+1}(\mathbf{M}_i^{(n)}) - \mathbf{m}_i^{(n+1)}\right]. \tag{5}$$

Here, $\mathcal{P}_{n+1}(\mathbf{M}_i^{(n)})$ computes the image projection of $\mathbf{M}_i^{(n)}$ in the $(n+1)$th frame. The Kalman Gain $\mathbf{K}_i \in \mathbb{R}^{3\times 2}$ is computed as

$$\mathbf{K}_i = \Sigma_i^{(n)} \hat{\mathbf{J}}_i^T \left(\sigma^2\mathbf{I} + \hat{\mathbf{J}}_i^T \Sigma_i^{(n)} \hat{\mathbf{J}}_i\right)^{-1}. \tag{6}$$

Here, $\hat{\mathbf{J}}_i \in \mathbb{R}^{2\times 3}$ is the Jacobian of $\mathcal{P}_{n+1}(\cdot)$ evaluated at $\mathbf{M}_i^{(n)}$. The triangulation uncertainty can meanwhile be updated by

$$\Sigma_i^{(n+1)} = \Sigma_i^{(n)} - \mathbf{K}_i \hat{\mathbf{J}}_i \Sigma_i^{(n)}. \tag{7}$$

We illustrate the idea of this refinement in Fig. 3.

Unlike SLAM algorithms based on Kalman filter, such as [9], [11], which spend a high computational cost $O(N^2)$ (N is the number of map points) to maintain the covariance matrix for both camera states and the whole map states (including correlation between positions of different points), we only maintain the triangulation uncertainty for each individual map point. The computation cost of our method is only $O(N)$. Further, the computation is independent at each point, which enables parallel computation to achieve better efficiency. Maintaining position uncertainty is important for the following map operations such as point classification and registration described in Sections 5.3 and 5.4. For example, the point classification benefits from the position uncertainty. Even for static points, their reconstructed positions are always changing over time due to triangulation uncertainties. (This can be seen clearly in our supplementary videos, which can be found in the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPAMI.2012.104.) With the position covariance matrix Σ_i, we can distinguish static and dynamic points better. Though we could also maintain the covariance among different points and uncertainty of camera positions, we do not include them for efficiency consideration.

5.2 Map Points Generation

We propose two methods to generate new map points. The first one, "intracamera mapping," reconstructs static map points from feature tracks in each individual camera. To deal with moving objects, we propose the second method, "intercamera mapping," to generate map points from corresponding points across cameras within the same group (i.e., cameras with view overlap).

5.2.1 Intracamera Mapping

Previous methods usually use key frames to generate new map points [18], [26], where feature points in selected key frames are matched and triangulated. Since all feature points are tracked over time in our system, it is not necessary to select key frames to match those feature points again. If there are unmapped feature tracks (whose pointers to map points are NULL) that are long enough ($>N_{min}$ frames), we use the beginning and the end frames of this track to triangulate a 3D point. Once the 3D position is computed, the covariance can also be evaluated by (4). We then check the reprojection error at all frames of the feature track. If the Mahalanobis distance (described in Section 5.3) between the projection and the feature point is smaller than θ for all frames, a new map point is generated and marked as "static."

5.2.2 Intercamera Mapping

Intercamera mapping is applied to unmapped feature points only. We match the image features between different cameras by zero-mean normalized cross correlation (ZNCC). To avoid ambiguous matches, the corresponding points are searched only in a narrow band within 3σ distance to the epipolar line. Only matches with ZNCC $> \tau_{ncc}$ are considered. We further use the correspondence of existing map points as seeds to guide matching—a pair of feature points is not considered to be matched if it has a very different disparity vector from that of the nearest seed.

As shown in Fig. 4, suppose we want to find the corresponding point for the unmapped feature point i. There are two candidates, k and j, within the band of the epipolar line. u is the closest mapped feature point to i. If it is within ϕ_r pixels distance to i, we use the disparity vector D_{uv} to guide feature matching. We compare the difference between the disparity vectors. A candidate with very different disparity from the seed is discarded, e.g., candidate k is removed in Fig. 4 because $\|D_{uv} - D_{ik}\| > \phi_d$. The best match is then obtained from the remaining candidates by the winner-take-all strategy.

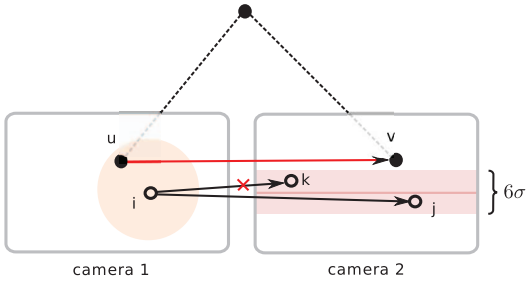


Fig. 4. Guided feature matching between two cameras. u, v are the projections of a map point. We will prefer the match $i \leftrightarrow j$ over $i \leftrightarrow k$ for its better consistency to $u \leftrightarrow v$.

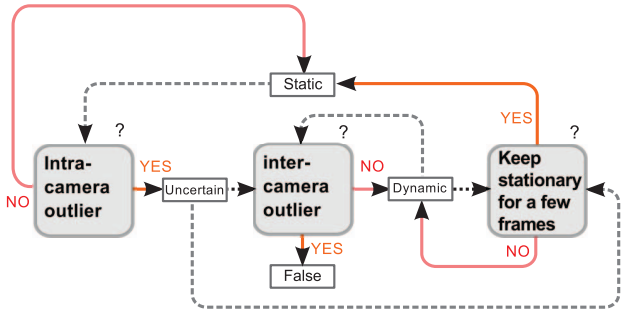


Fig. 5. Map point classification. The pipeline for classifying a point into four types: "static," "dynamic," "false," or "uncertain."

After matching feature points between cameras, we triangulate the corresponding points to generate new map points. Exhaustive feature matching in all possible camera pairs is inefficient. We construct a graph for cameras within the same group where cameras are linked according to their view overlap. We select a spanning tree of the graph and only match features between cameras that are connected by the spanning tree edges. More details of this spanning tree are discussed in Section 6. Intercamera mapping are called every five frames in our system when dynamic points are detected.

5.3 Point Registration

At every frame, we need to associate each map point with its incoming observations—newly detected image feature points from different cameras. Many of the feature points are registered to a map point through feature tracking. We further process the remaining unmapped feature points. For these points, we only consider *active* map points which are static and have corresponding feature points within the most recent N_{rec} frames. These map points are cached in our system for fast access. For each unmapped feature point detected at the current frame, we go through these *active* map points for registration.

We project *active* map points to the images and compare the image patches centered at the projections with that of the feature point through ZNCC. Considering the uncertainty in map point position and feature detection, the projected position of a map point $\mathbf{M}_i$ should satisfy a Gaussian distribution $\mathcal{N}(\mathbf{m}_i, \mathbf{c}_i)$, where the covariance $\mathbf{c}_i = \hat{\mathbf{J}}_i \Sigma_i \hat{\mathbf{J}}_i^T + \sigma \mathbf{I}$, $\hat{\mathbf{J}}_i \in \mathbb{R}^{2\times 3}$ is the Jacobian of the image projection $\mathcal{P}(\cdot)$ evaluated at $\mathbf{M}_i$. The Mahalanobis distance is computed as

$$\mathcal{D}^2(\mathbf{m}_j, \mathbf{m}_i) = (\mathbf{m}_j - \mathbf{m}_i)^T \mathbf{c}_i^{-1} (\mathbf{m}_j - \mathbf{m}_i). \quad (8)$$

We only consider the feature point $\mathbf{m}_j$ which has the smallest Mahalanobis distance to $\mathbf{m}_i$. We then check the ZNCC score between $\mathbf{M}_i$ and $\mathbf{m}_j$. To alleviate problems caused by perspective distortion, when selecting the image patch for $\mathbf{M}_i$ we choose the one stored from the nearest camera. $\mathbf{m}_j$ is discarded if its ZNCC score $< \tau_{ncc}$. We further traverse back along the feature track of $\mathbf{m}_j$ to check if its previous positions are also nearby to the projections of $\mathbf{M}_i$. If the Mahalanobis distances between them in all frames are smaller than θ, then the $\mathbf{m}_j$ is registered to $\mathbf{M}_i$.

Once an unmapped feature point is registered to a 3D map point, the 3D position and the position covariance of this map point can be updated based on the new observation. However, the new observation obtained from point registration, unlike those obtained from feature tracking, usually deviates largely from the previous observations (e.g., this new observation often comes from a different camera). It will lead to inaccurate estimation if we use the iterative refinement described in (5) and (7). In such case, we retriangulate the 3D position of this map point with all observations and recompute the covariance by (4). To reduce the computational cost, we select only two observations from the feature track in each camera for retriangulation which have the largest viewpoint changes.

5.4 Point Classification

At every frame, we need to distinguish "dynamic" points on moving objects and "static" points on the background. A naive method for this classification is to threshold the position variation of the 3D map points. It is, however, difficult to set a common threshold for different scenes. Further, in our system the positions of static points are also changing over time since their positions are updated whenever new observations are available. Especially for static points in the distant background, their positions may change significantly as cameras move from far to near.

We instead distinguish static and dynamic points by the reprojection error, which can be easily measured on the image plane. For a dynamic point, if we project its 3D position at the $(n-1)$th frame to the nth frame since the 3D position actually changes, the projection should be distant from its tracked corresponding feature points. In other words, if we use image feature points from different (or the same) time to triangulate the 3D position of a dynamic point, the reprojection error should be large (or small). In comparison, if the point is static, the reprojection error should always be small, no matter if we use image feature points from the same or different time. Based on this observation, we design a process to distinguish "static" and "dynamic" points. The whole process is illustrated in Fig. 5. We use "false" points to denote map points generated from incorrect correspondence. An intermediate state "uncertain" is also introduced for points need further investigation.

Initially, we consider all points as static. At every frame, we check the reprojection errors of all "static" points. The projected position of a static map point obeys Gaussian distribution $\mathcal{N}(\mathbf{m}_i, \mathbf{c}_i)$. The Mahalanobis distance between corresponding feature points and $\mathbf{m}_i$ should be less than θ. If the tracked feature point has larger Mahalanobis distance, the map point is likely to be "dynamic" or "false." We consider these points are *intracamera outliers*, i.e., outliers for

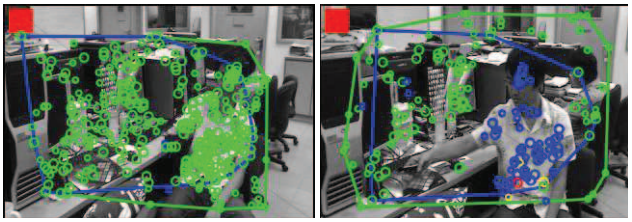


Fig. 6. Map point classification. Green and blue points indicate static and dynamic map points. Left: During the initialization, all points were marked as "static." Right: Our system correctly identified the dynamic points by map classification.

intracamera triangulation. We mark them as "uncertain" for the next step of classification.

An uncertain point could be either "dynamic" or "false." To distinguish them, we retriangulate its 3D position $\mathbf{M}_i$ with its tracked feature points in the same frame from different cameras. If the Mahalanobis distances of all these feature points to the projection of $\mathbf{M}_i$ are smaller than θ, we consider the map point as "dynamic." Otherwise, it is an *intercamera outlier*, i.e., an outlier for intercamera triangulation. We consider *intercamera outliers* as "false" points caused by incorrect feature matching. Note that the 3D positions of dynamic points are naturally updated over time during point classification. Hence, our system is able to produce the 3D trajectories of moving points.

A dynamic point may become static if the object stops moving. Hence, we project the current 3D position of a dynamic point to the previous frames. If the projection is close to its tracked feature points (Mahalanobis distance $< \theta$) for N_{min} of continuous frames, we consider this point as "static."

Fig. 6 shows two video frames of a camera from the "sitting man" example, where all the points on the body should be dynamic. We use green and blue points to visualize static and dynamic points. Though all map points were marked as "static" initially during the SLAM process, our map point classification component successfully differentiated dynamic and static points. This can be seen from the right in Fig. 6, where all points on the body were marked in blue.

6 Camera Grouping

The intercamera operations, e.g., mapping and pose estimation, can be only applied to cameras with view overlap. As cameras move independently, the view overlap among cameras changes over time. In this section, we describe our method to identify and manage camera groups, where cameras with view overlap are in the same group.

6.1 Grouping and Splitting

Since we store a pointer to the corresponding 3D point for each mapped feature point, we can quickly count the number of common map points N_{ij} between two cameras i, j. We construct an undirected graph where the nodes represent the cameras. If $N_{ij} > 0$, we connect the camera i and j by an edge weighted by N_{ij}. A connected component in this graph forms a camera group. The intercamera operations are only applied to cameras in the same group.

As discussed in Section 5.2.2, to improve the efficiency of intercamera mapping we do not match feature points between all camera pairs with view overlap. Instead, we extract a spanning tree for each camera group with maximum weight, and only match feature points between cameras if the edge connecting them is on the selected spanning tree.

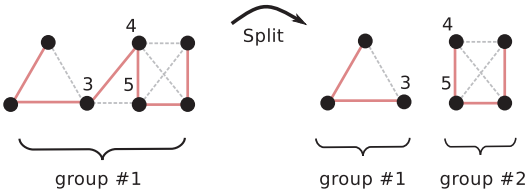


Fig. 7. Camera grouping and splitting. Each node is a camera and each edge links two cameras with view overlap. The solid edges are those on the spanning trees.

A camera group will split when any camera in it moves away and does not have view overlap with others. Such a case is illustrated in Fig. 7. The edges between camera 3, 4 and 3, 5 are removed as they have no common feature points any more. The original graph is therefore separated into two connected components, each of which forms a new camera group at the current frame. A real example is provided in Fig. 8. Four cameras were split into two groups, where the red and green cameras share a common view and the blue and yellow cameras share another common view.

6.2 Merging

Two camera groups will be merged if their cameras meet and have view overlap. To detect if cameras in different groups have view overlap, we project the map points generated from one camera onto the image planes of the cameras in the other group. If the number of visible points is large (> 30 percent of all map points from that camera in our implementation), and the area spanned by these points are large (> 70 percent of the image area), we consider the two cameras to have view overlap and will merge their camera groups. A real example of such a merge is shown in Fig. 9; the separated camera groups meet again.

When cameras move away from each other, the mapping and localization are performed within each camera group independently. When the cameras meet again due to drifting errors [7], the 3D maps reconstructed from different groups are inconsistent. For example, the same object could

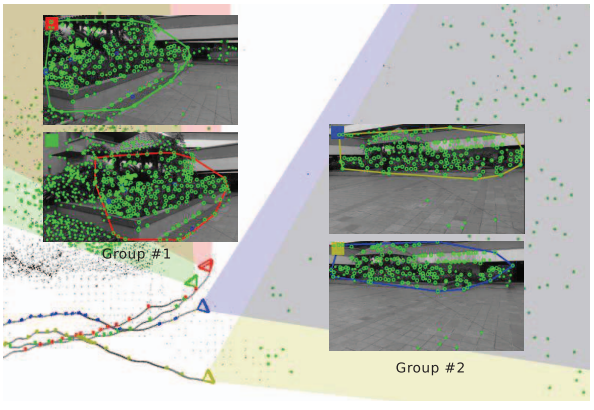


Fig. 8. The cameras were split into two groups according to their view overlap.

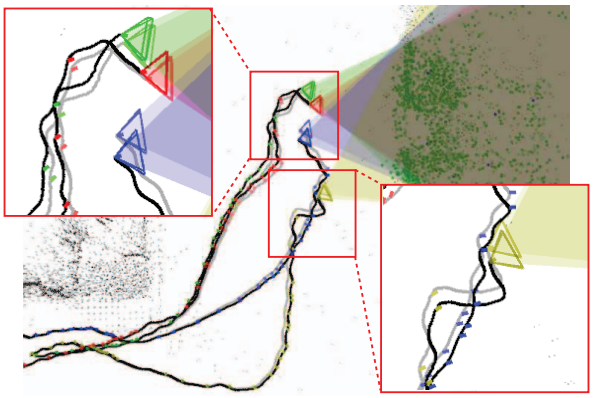


Fig. 9. The two camera groups were merged when the cameras meet again. The camera poses and map points were adjusted for consistent merge. The light gray curves (see the zoomed area) represent the camera trajectories before adjustment and the dark ones are those after adjustment.

be reconstructed at different 3D positions in different groups. Hence, during group merging we need to correct both the camera poses and map points to generate a single global consistent map. Suppose two camera groups are separated at the first frame and are merged at the Fth frame. We will adjust all camera poses from frame 2 to F, and adjust the map points generated within these frames, which consists of two successive steps described in the following section.

6.2.1 Step 1

We first estimate the correct relative poses between cameras at frame F. For this purpose, we detect and match SURF features between cameras in different groups, and then compute their relative poses (i.e., the essential matrices). We use these essential matrices to guide the matching of feature points (i.e., searching for correspondences in a narrow band within 3σ distance to the epipolar line). For each pair of matched feature points, we then merge their corresponding 3D map points by averaging their positions. In the next step, all the map points and their corresponding feature points in the Fth frame are put into bundle adjustment [15] to refine all camera poses.

6.2.2 Step 2

Now, we use the updated relative camera poses at the Fth frame as hard constraints to refine all camera poses. Fig. 10 illustrates our problem formulation. We form a undirected graph where each camera pose is a vertex and each edge enforces a relative pose constraint. As shown in Fig. 10, for each camera, its poses at neighboring frames are connected. For cameras in the same group, their poses at the same frame are connected if they are neighbors in the spanning tree. We fix camera poses in the first frame. Except for the relative poses at the Fth frame, we treat all the other relative poses as soft constraints. Hard and soft constraints are denoted by solid and dashed lines in Fig. 10, respectively.

Let $p = 1, \ldots, P$ and $q = 1, \ldots, Q$ be cameras from different groups. We denote the pose of the camera p at the ith frame by $\mathbf{T}_p^i$, and the relative pose between the camera p and q at the ith frame by $\mathbf{T}_{pq}^i$, where

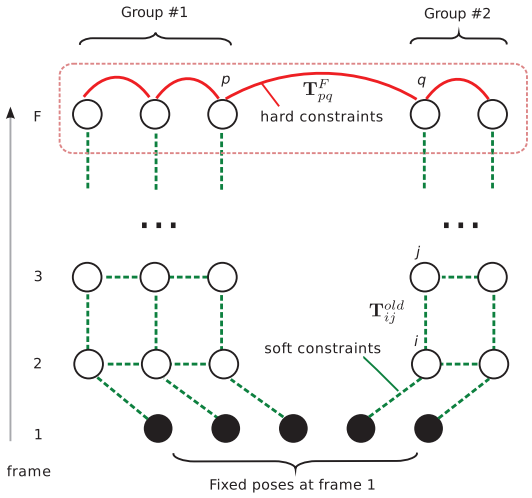


Fig. 10. Camera poses adjustment. Each vertex is a camera pose. Each edge represents a relative pose constraint, where solid and dash edges are hard and soft constraints, respectively.

$$\mathbf{T}_p^i = \begin{pmatrix} \mathbf{R}_p^i & \mathbf{t}_p^i \\ \mathbf{0}^T & 1 \end{pmatrix}, \qquad \mathbf{T}_{pq}^i = \begin{pmatrix} \mathbf{R}_{pq}^i & \alpha \mathbf{t}_{pq}^i \\ \mathbf{0}^T & 1 \end{pmatrix}. \tag{9}$$

$\mathbf{R}_p^i, \mathbf{R}_{pq}^i \in \mathbb{R}^{3\times 3}$ and $\mathbf{t}_p^F, \mathbf{t}_{pq}^F \in \mathbb{R}^3$ are rotation matrices and translation vectors. α is used to account for the global scale difference between the two camera groups.

We treat the relative poses at the Fth frame as hard constraints. Hence,

$$\mathbf{T}_q^F - \mathbf{T}_{pq}^F \mathbf{T}_p^F = \mathbf{0}_{4\times 4}, \tag{10}$$

which is equivalent to

$$\mathbf{R}_q^F - \mathbf{R}_{pq}^F \mathbf{R}_p^F = \mathbf{0}_{3\times 3}, \tag{11}$$

$$\mathbf{t}_q^F - \mathbf{R}_{pq}^F \mathbf{t}_p - \alpha \mathbf{t}_{pq}^F = \mathbf{0}_{3\times 1}. \tag{12}$$

Although there are $(P+Q)\times(P+Q-1)/2$ relative poses at the Fth frame, we select only $(P+Q-1)$ of them, which either lie on the spanning trees of the camera groups or connect the two spanning trees, as illustrated by the solid lines in Fig. 10. Putting all these constraints together, we get two linear systems with the following forms:

$$Ur^F = 0 \quad \text{and} \quad Vt^F = 0, \tag{13}$$

where $r^F \in \mathbb{R}^{9(P+Q)}$ is a vector stacked with elements of all the rotation matrices at the Fth frame, and $t^F \in \mathbb{R}^{3(P+Q)+1}$ is a vector that consists of all the translation elements at the Fth frame together with the scale factor α.

The relative camera poses from the original SLAM process are used as soft constraints. For any cameras m and n connected by the dashed edge, we expect their relative pose to have small change by the adjustment. Hence,

$$\mathbf{T}_m - \mathbf{T}_{mn}^{old} \mathbf{T}_n \approx 0. \tag{14}$$

Here, $\mathbf{T}_{mn}^{old}$ is the relative pose between m and n according to the SLAM process before merging. Putting all soft constraints together, we obtain two similar linear systems:

$$Ar \approx a \neq 0 \quad \text{and} \quad Bt \approx b \neq 0, \tag{15}$$

where $r \in \mathbb{R}^{9F(P+Q)}$ and $t \in \mathbb{R}^{3F(P+Q)}$ are vectors stacked by all the rotation and translation elements of all frames. Notice that the right sides of the two linear systems are not equal to zero because the camera poses at the first frame are fixed.

Combining both the hard constraints in (10) and soft constraints in (14), we obtain the updated cameras poses and the scale factor by solving two constrained linear least square problems:

$$\arg\min_{r} \|Ar - a\|^2 \quad \text{s.t.} \quad \hat{U}r = 0, \tag{16}$$

and

$$\arg\min_{\hat{t}} \|\hat{B}\hat{t} - \hat{b}\|^2 \quad \text{s.t.} \quad \hat{V}\hat{t} = 0, \tag{17}$$

where $\hat{t} \in \mathbb{R}^{3F(P+Q)+1}$ is t appended with a scale factor α. $\hat{U}, \hat{V}, \hat{B}, \hat{b}$ are the augmented matrices and vectors by adding zero elements. Note that we do not impose orthonormality condition to the rotation matrices in this formulation. Hence, once we obtain results from the above two equations, we further find the closest rotation matrices to the initial matrices by SVD (i.e., setting all the singular values to one).

The above optimization problem is converted to a set of sparse linear equations [13]. We use the CSparse [8] library to solve them in our system. After the camera poses have been updated, the 3D positions of map points are also updated by retriangulating their corresponding feature points.

In Fig. 9, the camera poses are updated when the two camera groups merge. The light gray curves (and dark curves) represent the camera trajectories before (and after) merging. To further exemplify the importance of pose updates, we examine the epipolar geometries in Fig. 11. In the left of Fig. 11, we plot the epipolar lines of the feature point A in the first camera. Because of the map inconsistency between the two camera groups, the corresponding feature points in the third and the fourth camera do not lie on the epipolar lines. In comparison, after camera pose update and retriangulation of map points, the corresponding feature points in all cameras lie on their respective epipolar lines as shown in the right of Fig. 11. To provide an additional validation, when visualizing the projection of a map point in Fig. 11 we use its circle radius to indicate its number corresponding feature points. It is clear that the overall circle size is larger on the right, which suggests map points have more corresponding feature points after the adjustment. This increase in corresponding feature points comes from the merge of duplicated 3D map points.

7 Incremental Refinement

We refine both the camera poses and the 3D map points from time to time by bundle adjustment. For better efficiency, bundle adjustment only refines the camera poses of some selected key frames and the map points reconstructed from these frames. Whenever there is a significant drop (30 percent) in the number of tracked feature points in any camera, we insert a key frame for all cameras.

The bundle adjustment runs in a separate thread, which operates with the most recent K key frames. It is called when $K-1$ key frames have been inserted consecutively

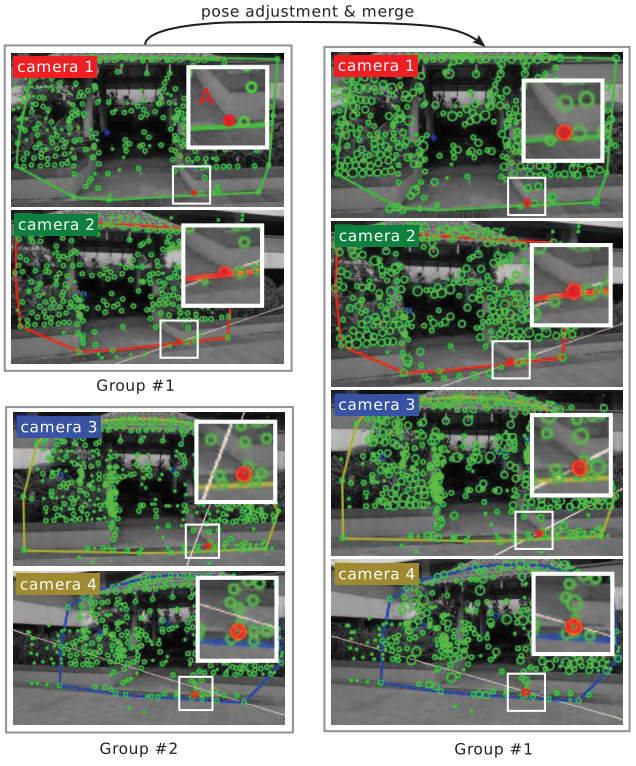


Fig. 11. Camera poses and map point positions are adjusted during group merge. Left: Before the adjustment, the corresponding feature points of A in the "camera 3" and "camera 4" do not lie on the its epipolar lines. Right: After adjustment, all its corresponding feature points lie on the epipolar lines.

(i.e., in two successive bundle adjustment calls, there is one common key frame). The bundle adjustment only refines camera poses of key frames and map points reconstructed from these frames. To refine the camera poses of the other frames, we adopt a similar method as Section 6.2.2. Basically, we fix the camera pose at key frames, and use the relative poses between successive frames before bundle adjustment as soft constraint. In other words, we enforce $\mathbf{T}_m - \mathbf{T}_{mn}^{old}\mathbf{T}_n \approx 0$, where $\mathbf{T}_{mn}^{old}$ is the relative pose between camera m, n before bundle adjustment. We then update all the camera poses while keep those at key frames unchanged. After the pose refinement, the 3D positions of other map points are updated by retriangulating their corresponding feature points.

8 Results

We tested our collaborative SLAM system on both static scenes and dynamic scenes. All data were captured by hand-held cameras with Field of View about 70 degrees and processed offline. In all our experiments, we set the standard deviation of feature detection uncertainty σ as 3.0 pixels. The threshold for Mahalanobis distance θ is set to be 2.0 to decide if a feature point is an inlier or outlier (with confidence of 95 percent according to Gaussian distribution). The ZNCC threshold τ_{ncc} to measure the similarity between image patches is set to 0.7. The minimum number of frames N_{min} to triangulate a feature track is set to 60. The number of frames N_{rec} for active map point caching is 200. The radius ϕ_r for searching nearby seed matches in

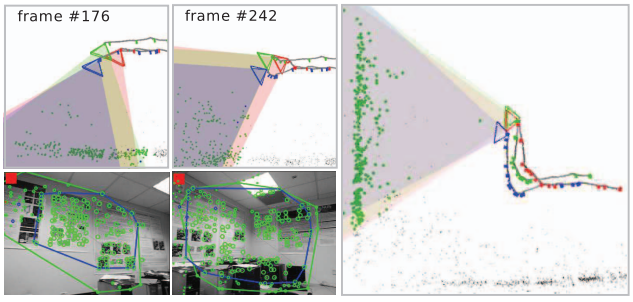


Fig. 12. Our results on the "wall" sequence. Trajectories of different cameras are visualized in different color from the top-down viewpoint. Map points are projected in the input video frames for a reference.

"intercamera mapping" is set to 10 percent of max{image width, image height}, and $\phi_d = 3\phi_r$. In practice, we found our results are not sensitive to these parameters. We also present the results on an accompanying video.

8.1 Static Scenes

8.1.1 Critical Camera Motion

Single camera-based visual SLAM systems usually fail under critical motion, such as camera rotation without translation. This is because when a camera rotates, the number of visible map points drops quickly. However, new map points cannot be generated because of little camera translation. This problem is more serious when the camera field of view is small. Our CoSLAM system can deal with such situation by the collaboration among multiple cameras, as demonstrated in "wall" example in Fig. 12. In this example, cameras started rotating at around the 180th frame and finished at about the 256th frame. Our method successfully captured the motion of all cameras and the two perpendicular walls.

For a comparison, we applied the single camera-based SLAM system, PTAM [18], on the same data. The results from PTAM are provided in Fig. 13. PTAM failed on all cameras when they started to rotate. Even the relocalizatoin cannot recover it from this failure. This is because the relocalization works only when the newly captured frames have view overlap with previous key frames. In camera rotation, the camera is turning to a novel view that is not observed before. Hence, relocalization cannot help in this case.

8.1.2 Drift Analysis

We tested our CoSLAM system in a middle scale "courtyard" example. The average length of camera trajectories was 96 m. The input videos were captured in a courtyard by four hand-held cameras. The view overlap among cameras changed over time, which led to camera group splitting and merging as discussed in Section 6. During data capturing, we walked around the courtyard and returned to the starting place. Hence, the drift error can be measured by manually identifying corresponding points in the first and the last video frames. We analyzed the drift errors with different numbers of cameras in the system. An overview of the scene is provided in Fig. 14.

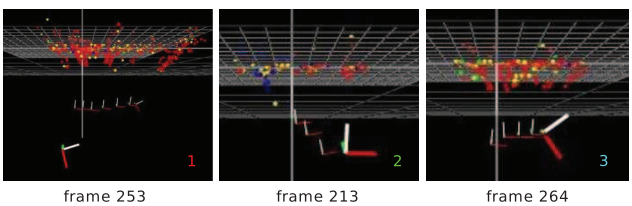


Fig. 13. Results generated by PTAM [18] on the "wall" sequence. The system failed on all three sequences when the camera started rotating.

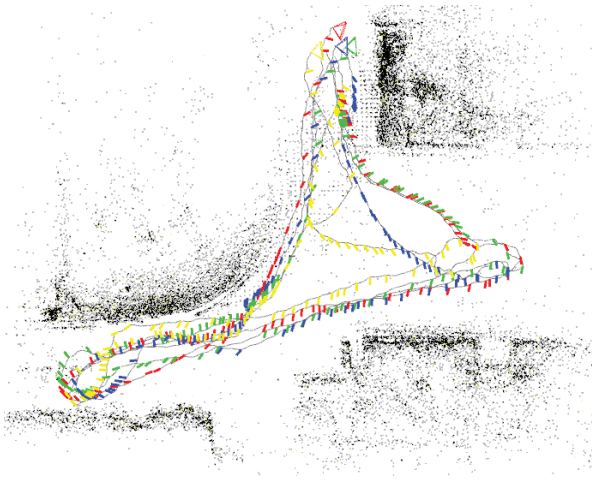


Fig. 14. The overview of the "courtyard" example. Four hand-held cameras were used to capture the data. The view overlaps between cameras changed over time.

We tried all possible one-camera, two-camera, three-camera, and four-camera CoSLAM. The average distance drift errors were 2.53, 1.57, 1.19, 0.65 m, respectively. The average scale drift errors were 0.76, 1.10, 0.96, 1.00, respectively. This result is visualized in Fig. 15, where the red and green line segment indicates the reconstruction of the sign board from the first and last frames. This result indicates that CoSLAM with multiple cameras can successfully reduce the drift errors.

For a comparison, we tried to apply the PTAM system to this example and measure its drift error. However, PTAM failed at all four one-camera tests and cannot finish the whole loop. In comparison, our system only failed in one one-camera test. We believe this is because we maintain the triangulation uncertainty of map points and continuously improve map accuracy when more images are captured.

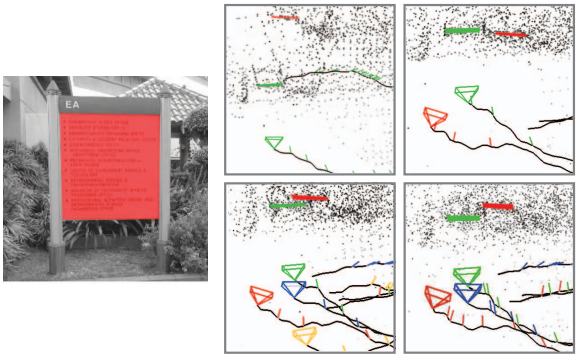


Fig. 15. The final drift error is measured by the difference between the board reconstructed from the first (marked by red) and the last video frames (marked by green). The left is an example image of the board. The right are the CoSLAM results with different numbers of cameras.

Fig. 16. Our results on the "sitting man" sequence. There are only a few static points in the scene. Our "intercamera pose estimation" successfully estimates all camera poses (shown in the first two rows). The bottom row shows the result by only applying the "intracamera pose estimation." The pose of the blue camera (camera #3) was completely wrong at the 665th frame because of little static points. The reconstructed map points (both static and dynamic points) and camera trajectories of the last 150 frames are provided on the right (visualized from the top view)

8.2 Dynamic Scenes

The capability of the our method to robustly estimate camera poses in highly dynamic scenes is demonstrated in Fig. 16. Note that the number of static points (green dots) was small in all cameras. Further, the static points were often distributed within a small region, which usually leads to large error if the camera pose is estimated only according to static points. Our CoSLAM system automatically switched to the "intercamera pose estimation" and successfully estimated the poses of all cameras in such a challenging case. For a comparison, we disabled the "intercamera pose estimation" and applied the "intracamera pose estimation" to the same sequence. The system failed at the 665th frame because of the large error in pose estimation. This comparison can be seen clearly from the visualization of the camera trajectories in the right column of Fig. 16.

We tested our CoSLAM system in an indoor scene, the "walking man" example in Fig. 17. The relatively dim indoor lighting usually leads to blurry images, which make feature tracking difficult and pose estimation inaccurate. Our CoSLAM system can successfully handle this data. The estimated map points and camera trajectories are visualized from the top view in the left of Fig. 17. An important feature of our system is that we can recover the 3D trajectories of moving points, which is demonstrated in Fig. 18, where the blue curves are 3D trajectories of the dynamic points on the walking person. We also manually specify corresponding points to measure the drift error. Our CoSLAM system had 1.2 m distance drift and 1.12 scale drift. The average length of camera trajectories is 28.7 m.

We further tested the robustness of our system on a middle scale "garden" example, where the average length of camera trajectories is 63 m. Three hand-held cameras were used to capture the input videos in a garden. Several people walked in front of the camera to disturb the capturing process. As the moving people frequently occupied a large

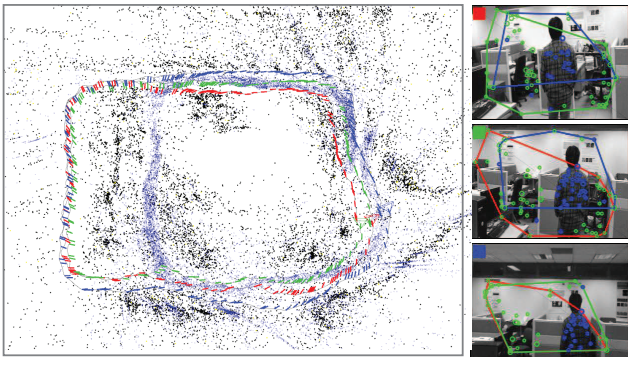


Fig. 17. Result on the "walking man" example. The blue points represent the moving points in the scene.

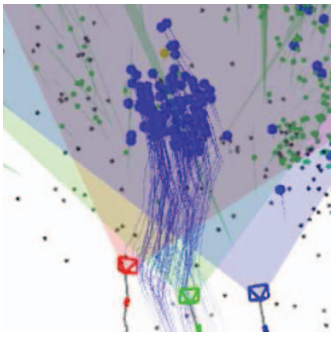


Fig. 18. Our system can track the time-varying 3D positions of the dynamic points on the moving objects.

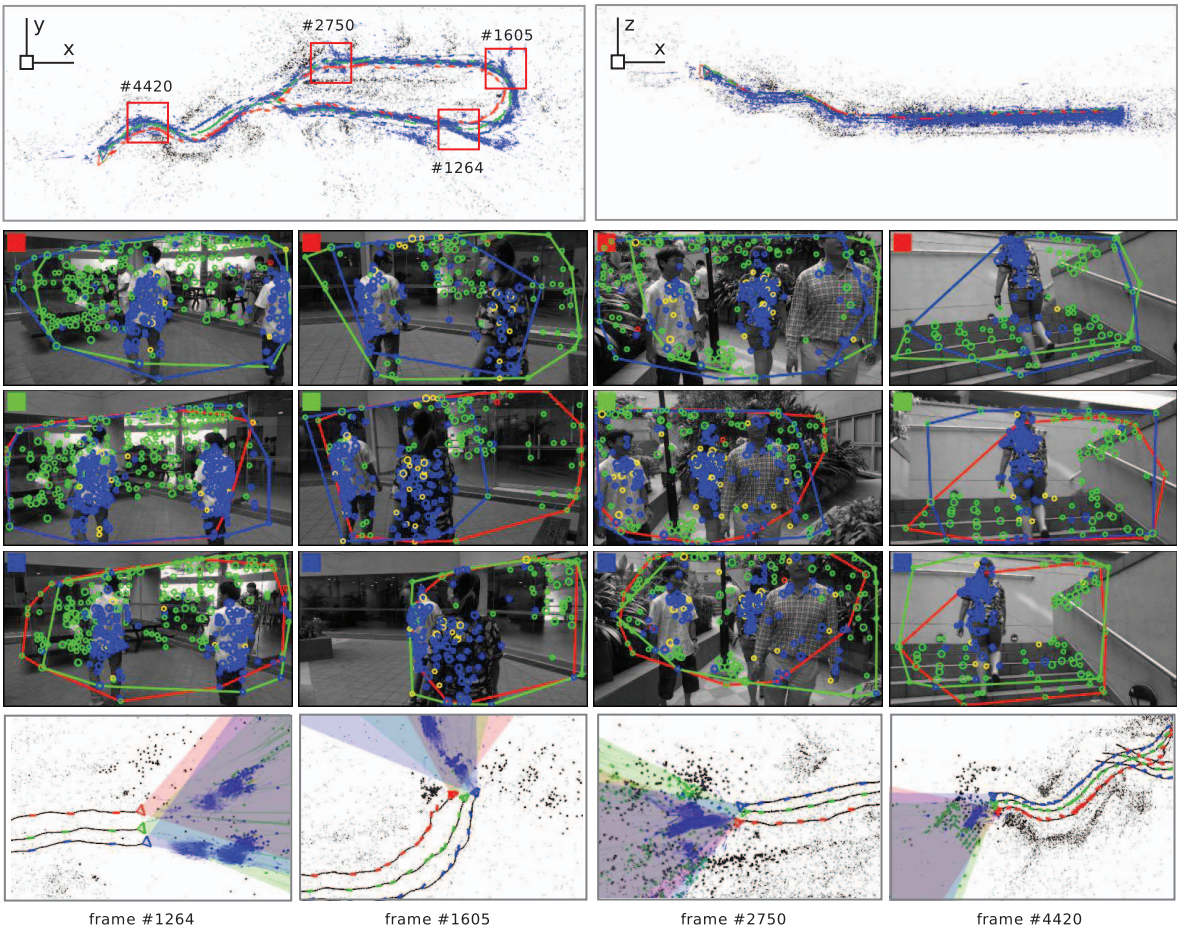


Fig. 19. The collaborative SLAM result in a challenge dynamic scene using three cameras. An overview of the reconstruction is provided in the first row from the top and side view, respectively. The gray point cloud indicates the static scene structures, while the blue trajectories are the moving points. In the middle are the detected feature points on video frames. The green and blue points represent static and dynamic points, respectively, while the yellow points are the unmapped feature points (please zoom in to check details). In the last row, we provide some zoomed views of the camera trajectories with their frame index marked in the first row.

part of the video frame, the SLAM problem with this data was very challenging (please refer to the supplementary video, which is available online.) As shown in Fig. 19, our method succeeded with such data. The manually measured drift errors are 5.3 m in distance and 0.65 in scale. These errors are relative large because the moving objects frequently occluded nearby static points. In this situation, the faraway static points played a more important role in camera pose estimation. However, the positions of these faraway points were less reliable, which led to inaccurate estimation and finally produced a relative large drift error.

8.3 Runtime Efficiency

Our system was developed under the Ubuntu 64-bit system with an Intel i7 CPU (4 cores at 2.80 GHz), 4 GB RAM, and an nVidia GeForce GTX 480 graphics card. The main thread of the system estimated the egomotion of the cameras and generated map points. Bundle adjustment and camera group management were called in separated threads. Feature tracking was implemented in GPU according to the GPU-KLT [29].

We evaluated the runtime efficiency on the "garden" sequence shown in Fig. 19. All 4,800 frames were used for evaluation. The average times spent in each call of all components are listed in Table 1. Most of the components run quickly. The "intercamera pose estimation" and the "intercamera mapping" took about 50 ms to run. The number of map points (including both dynamic and static points) and the processing time of each frame are also shown, respectively, in Fig. 21. Although the runtime efficiency was reduced when intercamera operations were called (see the peaks in Fig. 21), our system on average ran in real time and took about 38 ms to process a frame with about one thousand of map points.

8.4 System Scalability

We tested our system scalability with 12 cameras moving independently in a static scene. Some results are shown in

TABLE 1
Average Timings

components	ms	calling conditions
feature tracking	8.7	every frame (by GPU)
intra-camera pose estimation	10.7	every frame
inter-camera pose estimation	57.2	see Section 4
map point classification	4.9	every frame
map point registration	14.47	every frame
intra-camera mapping	2.3	see Section 5.2
inter-camera mapping	48.3	see Section 5.2

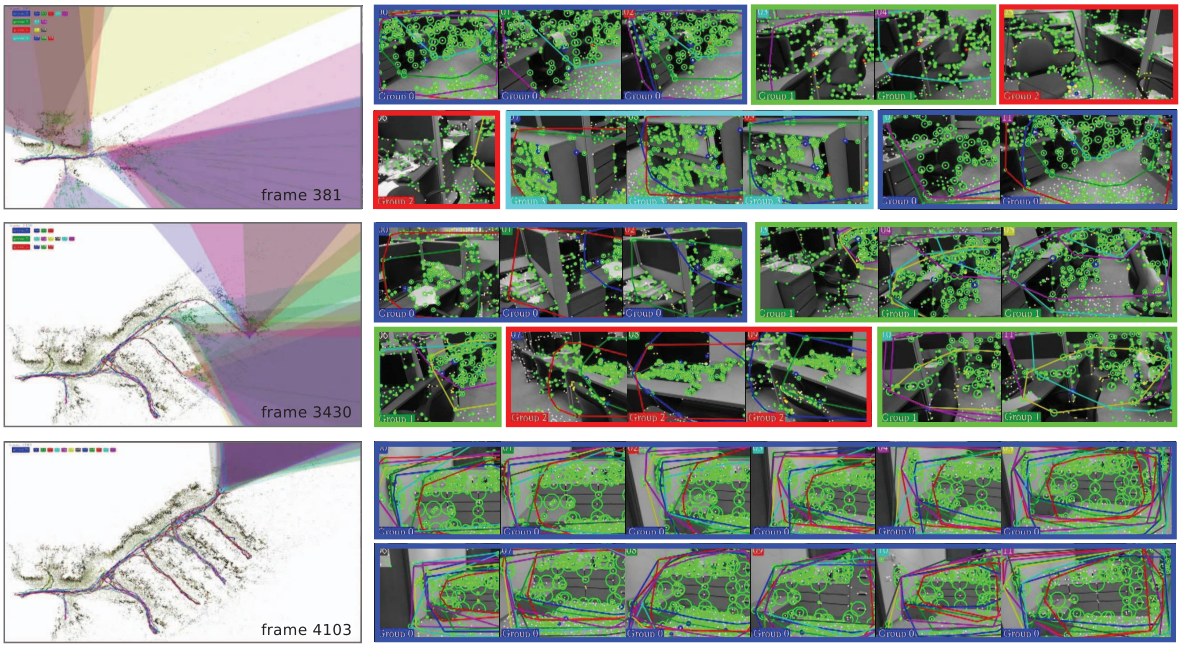


Fig. 20. Our system was tested with 12 cameras. The map and camera poses are shown on the left, where the top left corner shows the camera grouping. Images and feature points are shown on the right, where images framed in the same color come from cameras in the same group. These three rows have four, three, and one camera groups, respectively.

Fig. 20. The left of each row shows the top view of the reconstructed scene map with camera trajectories. On the right are the input images from all 12 cameras. Images framed by the same color come from cameras in the same group. This scene contains several separated "branches," as can be seen from the scene map in the third row. These 12 cameras were divided into several troops, and each troop explored one branch. Our system can correctly handle camera group splitting and merging. (Please refer to the supplementary video, which is available online.) However, when there are 12 cameras, the average runtime efficiency dropped significantly to about 1 fps due to the heavy computation.

9 Conclusion and Future Work

We propose a novel collaborative SLAM system with multiple moving cameras in a possibly dynamic environment. The cameras move independently and can be mounted on different platforms, which makes our system potentially applicable to robot teams [4], [34] and wearable augmented reality [5]. We address several issues in pose estimation, mapping, and camera group management so that the system can work robustly in challenging dynamic scenes as shown in the experiments. The whole system runs in real time. Currently, our system works offline with precaptured video data. We plan to integrate a data capturing component to make an online system. Further, our current system requires synchronized cameras, and all images from these cameras are sent back and processed in the same computer. It will be interesting to develop a distributed system for collaborated SLAM where computation is distributed to multiple computers.

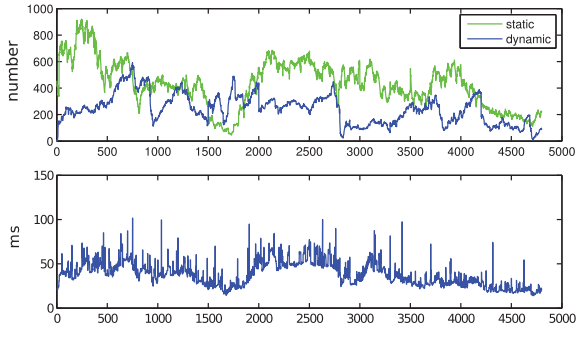


Fig. 21. Runtime efficiency with three cameras. Top: Number of map points over time. Bottom: The time spent to process each frame. The average time to process a frame is 38 ms.

Acknowledgments

This work is supported by the Singapore FRC grant R-263-000-620-112 and AORAD grant R-263-000-673-597.

References

[1] J. Allred, A. Hasan, S. Panichsakul, W. Pisano, P. Gray, J. Huang, R. Han, D. Lawrence, and K. Mohseni, "Sensorflock: An Airborne Wireless Sensor Network of Micro-Air Vehicles," *Proc. Int'l Conf. Embedded Networked Sensor Systems*, pp. 117-129, 2007.

[2] C. Bibby and I. Reid, "Simultaneous Localisation and Mapping in Dynamic Environments (Slamide) with Reversible Data Association," *Proc. Robotics: Science and Systems*, 2007.

[3] C. Bibby and I. Reid, "A Hybrid Slam Representation for Dynamic Marine Environments," *Proc. IEEE Int'l Conf. Robotics and Automation*, pp. 257-264, 2010.

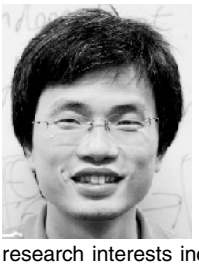
[4] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun, "Collaborative Multi-Robot Exploration," *Proc. IEEE Int'l Conf. Robotics and Automation*, vol. 1, pp. 476-481, 2000.

[5] R.O. Castle, G. Klein, and D.W. Murray, "Video-Rate Localization in Multiple Maps for Wearable Augmented Reality," *Proc. 12th IEEE Int'l Symp. Wearable Computers*, pp. 15-22, 2008.

[6] M. Chli and A.J. Davison, "Active Matching for Visual Tracking," *Robotics and Autonomous Systems*, vol. 57, no. 12, pp. 1173-1187, 2009.

[7] K. Cornelis, F. Verbiest, and L. Van Gool, "Drift Detection and Removal for Sequential Structure from Motion Algorithms," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 26, no. 10, pp. 1249-1259, Oct. 2004.
[8] T. Davis, *Direct Methods for Sparse Linear Systems.* SIAM, 2006.
[9] A. Davison, "Real-Time Simultaneous Localisation and Mapping with a Single Camera," *Proc. Ninth IEEE Int'l Conf. Computer Vision,* pp. 1403-1410, 2003.
[10] A. Davison, I. Reid, N. Molton, and O. Stasse, "MonoSLAM: Real-Time Single Camera SLAM," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 29 no. 6, pp. 1052-1067, June 2007.
[11] H. Durrant-Whyte and T. Bailey, "Simultaneous Localisation and Mapping (SLAM): Part I the Essential Algorithms," *IEEE Robotics and Automation Magazine,* vol. 13, no. 2, 99-110, June 2006.
[12] E. Eade and T. Drummond, "Scalable Monocular SLAM," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* vol. 1, pp. 469-476, 2006.
[13] G. Golub, "Numerical Methods for Solving Linear Least Squares Problems," *Numerische Mathematik,* vol. 7, no. 3, pp. 206-216, 1965.
[14] D. Hahnel, R. Triebel, W. Burgard, and S. Thrun, "Map Building with Mobile Robots in Dynamic Environments," *Proc. IEEE Int'l Conf. Robotics and Automation,* vol. 2, pp. 1557-1563, 2003.
[15] R. Hartley and A. Zisserman, *Multiple View Geometry,* vol. 6. Cambridge Univ. Press, 2000.
[16] K. Ho and P. Newman, "Detecting Loop Closure with Scene Sequences," *Int'l J. Computer Vision,* vol. 74, no. 3, pp. 261-286, 2007.
[17] M. Kaess and F. Dellaert, "Visual Slam with a Multi-Camera Rig," Technical Report GIT-GVU-06-06, Georgia Inst. of Technology, 2006.
[18] G. Klein and D. Murray, "Parallel Tracking and Mapping for Small AR Workspaces," *Proc. IEEE/ACM Int'l Symp. Mixed and Augmented Reality,* pp. 225-234, 2007.
[19] A. Kundu, K. Krishna, and C. Jawahar, "Realtime Motion Segmentation Based Multibody Visual SLAM," *Proc. Seventh Indian Conf. Computer Vision, Graphics and Image Processing,* pp. 251-258, 2010.
[20] B. Leibe, N. Cornelis, K. Cornelis, and L. Van-Gool, "Dynamic 3D Scene Analysis from a Moving Vehicle," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2007.
[21] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd, "Real Time Localization and 3D Reconstruction," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* vol. 1, pp. 363-370, 2006.
[22] R. Newcombe and A. Davison, "Live Dense Reconstruction with a Single Moving Camera," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* pp. 1498-1505, 2010.
[23] D. Nister, O. Naroditsky, and J. Bergen, "Visual Odometry," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* vol. 1, 2004.
[24] K. Ozden, K. Schindler, and L.V. Gool, "Multibody Structure-from-Motion in Practice," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 32, no. 6, pp. 1134-1141, June 2010.
[25] L. Paz, P. Piniés, J. Tardós, and J. Neira, "Large-Scale 6-DoF SLAM with Stereo-in-Hand," *IEEE Trans. Robotics,* vol. 24, no. 5, pp. 946-957, Oct. 2008.
[26] E. Royer, M. Lhuillier, M. Dhome, and T. Chateau, "Localization in Urban Environments: Monocular Vision Compared to a Differential gps Sensor," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* vol. 2, pp. 114-121, 2005.
[27] E. Sahin, "Swarm Robotics: From Sources of Inspiration to Domains of Application," *Swarm Robotics,* pp. 10-20, Springer, 2005.
[28] J. Shi and C. Tomasi, "Good Features to Track," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* pp. 593-600, 1994.
[29] S. Sinha, http://www.cs.unc.edu/ssinha/Research/GPU_KLT/, 2011.
[30] P. Smith, I. Reid, and A. Davison, "Real-Time Monocular SLAM with Straight Lines," *Proc. British Machine Vision Conf.,* vol. 1, pp. 17-26, 2006.
[31] H. Strasdat, J. Montiel, and A. Davison, "Real-Time Monocular SLAM: Why Filter?" *Proc. IEEE Robotics and Automation,* pp. 2657-2664, 2010.
[32] H. Strasdat, J. Montiel, and A. Davison, "Scale Drift-Aware Large Scale Monocular SLAM," *Proc. Robotics: Science and Systems,* 2010.
[33] H. Strasdat, J. Montiel, and A. Davison, "Visual SLAM: Why Filter?" *Image and Vision Computing,* vol. 30, pp. 65-77, 2012.
[34] S. Thrun, W. Burgard, and D. Fox, "A Real-Time Algorithm for Mobile Robot Mapping with Applications to Multi-Robot and 3D Mapping," *Proc. IEEE Conf. Robotics and Automation,* vol. 1, pp. 321-328, 2002.
[35] C. Wang, C. Thorpe, S. Thrun, M. Hebert, and H. Durrant-Whyte, "Simultaneous Localization, Mapping and Moving Object Tracking," *Int'l J. Robotics Research,* vol. 26, no. 9, pp. 889-916, 2007.
[36] B. Williams, G. Klein, and I. Reid, "Real-Time SLAM Relocalisation," *Proc. 11th IEEE Int'l Conf. Computer Vision,* pp. 1-8, 2007.
[37] N. Winters, J. Gaspar, G. Lacey, and J. Santos-Victor, "Omni-Directional Vision for Robot Navigation," *Proc. IEEE Workshop Omnidirectional Vision,* pp. 21-28, 2000.
[38] D. Wolf and G. Sukhatme, "Mobile Robot Simultaneous Localization and Mapping in Dynamic Environments," *Autonomous Robots,* vol. 19, no. 1, pp. 53-65, 2005.
[39] Z. Zhang, "Parameter Estimation Techniques: A Tutorial with Application to Conic Fitting," *Image and Vision Computing,* vol. 15, no. 1, pp. 59-76, 1997.
[40] J. Zuffery, "Bio-Inspired Vision-Based Flying Robots," PhD thesis, École Polytechnique Fédérale de Lausanne, 2005.

Danping Zou received the BS degree from Huazhong University of Science and Technology (HUST) and the PhD degree from Fudan University, China, in 2003 and 2010, respectively. Currently, he is a research fellow in the Department of Electrical and Computer Engineering at the National University of Singapore. His research interests include video tracking and low-level 3D vision.



Ping Tan received the BS degree in applied mathematics from Shanghai Jiao Tong University in China and the PhD degree in computer Science and engineering from the Hong Kong University of Science and Technology in 2000 and 2007, respectively. He joined the Department of Electrical and Computer Engineering at the National University of Singapore as an assistant professor in 2007. He received the MIT TR35@Singapore award in 2012. His research interests include computer vision and computer graphics. He is a member of the IEEE and the ACM.