Multithreaded Hybrid Feature Tracking for Markerless Augmented Reality

Taehee Lee, *Student Member*, *IEEE*, and Tobias Höllerer, *Member*, *IEEE*

Abstract—We describe a novel markerless camera tracking approach and user interaction methodology for augmented reality (AR) on unprepared tabletop environments. We propose a real-time system architecture that combines two types of feature tracking. Distinctive image features of the scene are detected and tracked frame-to-frame by computing optical flow. In order to achieve real-time performance, multiple operations are processed in a synchronized multithreaded manner: capturing a video frame, tracking features using optical flow, detecting distinctive invariant features, and rendering an output frame. We also introduce user interaction methodology for establishing a global coordinate system and for placing virtual objects in the AR environment by tracking a user's outstretched hand and estimating a camera pose relative to it. We evaluate the speed and accuracy of our hybrid feature tracking approach and demonstrate a proof-of-concept application for enabling AR in unprepared tabletop environments, using bare hands for interaction.

Index Terms—Vision-based registration and tracking, augmented reality, mixed reality, multithreading, interaction techniques, hand tracking.

1 Introduction

In recent years, augmented reality (AR) research has focused on various subareas of interest. Highly developed techniques in Graphics and Virtual Reality allow for increasingly realistic AR visuals. Elaborate tracking systems, input devices, and computer vision techniques improve the registration of images and user interaction in AR setups. However, most commonly used AR systems still either require sophisticated hardware tracking solutions, or at least fiducial markers.

Now that mobile computing devices have been widely deployed to customers, the interest in mobile AR is increasing. As the need for mobility is growing, computing devices continue to shrink in size and gain acceptance beyond an audience of tech-savvy specialists. With the help of enhanced computing power, small devices now have sufficient computational capability to process complex visual computations. For example, a cell phone with a camera and a wide screen can serve as an inexpensive AR device for the general public. Emerging wearable computers are also excellent platforms for enabling AR anywhere users may go. This paper introduces methodology to support inexpensive AR in unprepared environments.

We present and evaluate new techniques for markerless computer-vision-based tracking in arbitrary physical tabletop environments, as an extended version of our previously presented work [1]. Moreover, we discuss the use of this tracking technology for implementing the idea of an augmented desktop [2], [3] without the need for fiducial markers. Three-dimensional coordinate systems can be established on a tabletop surface at will, using a user's outstretched hand as a temporary initialization pattern. Building upon an existing framework for fingertip and hand pose tracking [4], users can establish a coordinate system for the tabletop AR environment simply by placing their outstretched hand onto a working plane surface.

Our camera tracking solution separates interest point detection and tracking and pose estimation in a real-time multithreaded framework, enabling the use of landmark or *distinctive* features previously deemed too expensive for real-time computations. Distinctive image features can play an important role in AR interfaces, and our approach utilizes them as part of a real-time tracking solution. The tracking methodology is efficiently modularized and synchronized with other AR system components and tasks, such as video capture, user interaction with objects in the environment, and real-time rendering. We detect distinctive image features in the scene and track them using a hybrid tracking mechanism involving feature synchronization and optical flow. We use the distinctive features to recognize a scene, providing a scheme to resume a stabilized AR experience after tracking failure, or in a new, previously viewed environment, for example for transferring augmentations to an alternate physical desk. We track the user's bare hands as an input device for placing and manipulating virtual objects in the AR environment.

1.1 Related Work

While there are many systems available to conduct AR in a prepared environment, we want to lower the barrier to

- *T. Lee is with the Vision Laboratory, Computer Science Department, University of California, Los Angeles, Boelter Hall, Room 3811, Los Angeles, CA 90095. E-mail: taehee@cs.ucla.edu.*
- *T. Höllerer is with the Department of Computer Science, University of California at Santa Barbara, Harold Frank Hall (Eng. I Bldg.), Room 2104, Santa Barbara, CA 93106-5110. E-mail: holl@cs.ucsb.edu.*

Manuscript received 2 July 2008; revised 15 Sept. 2008; accepted 6 Oct. 2008; published online 13 Oct. 2008.
Recommended for acceptance by M.C. Lin.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCGSI-2008-07-0087.
Digital Object Identifier no. 10.1109/TVCG.2008.190.

initiating AR systems, so that users can easily experience AR anywhere [5]. Considering a mobile user entering a new workplace, such as an office desk environment, we would like to enable the user to start using AR without spending much effort on setting up the environment. Several marker-based AR systems [6], [7] have shown successful registration of virtual objects on top of cardboard markers. Markers also provide a tangible user interface that users can physically handle or place in the scene. However, markers inevitably occlude other existing objects in the environment, causing limited space for interactions, especially in cluttered desktop environments.

Interactive "tabletop" systems aim to provide an enhanced user experience for tasks carried out on a horizontal working plane, which can either be a conventional desk- or tabletop, or an interactive display surface. Tabletops based on various setup technologies have been proposed, including projector/camera combinations [8], [9], AR using head-worn displays or projector/display combinations [10], [11], [2], [12], and interactive touch screens of various kinds [13], [14]. Enabling AR for unprepared tabletop environments by employing tangible interaction with virtual and physical objects on the tabletop is a promising approach to creating a digitally enhanced physical workspace.

With regard to tangible user interfaces, Handy AR [4] has shown that a user's bare hand can replace a cardboard marker [6], [7] for local coordinate estimation. Especially for highly mobile users, hands are the most obvious and most readily available user interface device for AR. We employ the Handy AR approach in our work for providing an initial camera pose estimation with scale information and also as a user interface for object selection and placement.

There exists significant related work on camera pose estimation for markerless AR using natural features such as planes, edges, or corner points [15], [16], [17]. With the assumption of a planar scene, computing a homography between frames provides an efficient tracking mechanism for registering augmentations with a scene [18]. In order to cope with general scenes, simultaneous localization and mapping (SLAM) technologies from the robotics field have been employed to track the camera pose while building a 3D map of the scene [19]. Recently, Klein and Murray [20] demonstrated a more robust markerless AR system by separating tracking and map building tasks into different threads. By carefully optimizing the structure-from-motion problem and taking advantage of a large number of point features, they enable robust real-time camera tracking for an AR system. While their approach also uses multiple threads for different tasks, this paper focuses more on enabling the use of highly distinctive image features, traditionally deemed too expensive for real-time tracking and interaction solutions. We detect high-quality landmarks and use them effectively for both tracking and scene recognition for AR user interfaces. Meanwhile, all existing tracking require manual calibration or detection of a known-size object in order to provide initial metric scale information. We want to minimize such start-up costs and scene preparations and enable correctly scaled AR user interaction from the start using hand tracking.

For fast and accurate tracking, optical-flow-based algorithms are widely used [21]. In order to arrive at stable tracking, landmark features are often used in order to recover from accumulated errors or complete tracking failures. While image patches [19] can be matched as templates to locate landmark features, they can be ambiguous in the presence of repetitive patterns, due to their fixed sizes. Recent results from [20] showed that using a large number of multiscale image patches can afford robust camera tracking in a small to medium-sized AR environments. While more distinctive image features that are invariant to scale and orientation [22], [23] can be detected robustly for tracking over a sequence of frames, their complex computation commonly renders the approach inappropriate for real-time applications that require around 30 fps [17]. For real-time performance that is at the same time robust and reliable in accuracy, a hybrid approach of combining these different types of tracking methods has been identified as a promising future research direction [24]. We are presenting such an approach within the application scope of tabletop working planes.

The rest of this paper is structured as follows: In Section 2, we present an overview of our approach by reviewing the technologies that our system builds upon and review some background knowledge related to the methods we propose. In Section 3, we describe how to combine hand-based interaction with the proposed hybrid feature tracking for establishing a coordinate system and estimating the camera pose for tabletop AR environments. In Section 4, we show experimental results and evaluations and introduce a proof-of-concept application. We discuss the merits and limitations of the proposed approach in Section 5 and conclude in Section 6.

2 Overview and Background

In this section, we review three technologies that our system builds upon: 1) Hand tracking and camera pose estimation using "Handy AR"; 2) Detection of distinctive invariant image features; and 3) Optical-flow-based techniques.

2.1 Handy AR

Using the "Handy AR" approach [4], we estimate a six-degree-of-freedom camera pose, substituting a cardboard marker with a user's outstretched hand. The hand is segmented by a skin-color-based classifier [25] with an adaptively learned skin color histogram [26]. While the hand is tracked over time, fingertips are located on the contour of the hand by finding points that have high curvature values. The fingertips are accurately fitted to ellipses and are used for estimating a camera pose. Five tracked fingertips in a known pose (measured for each user as a once-in-a-lifetime offline calibration step) provide more than the minimum number of point correspondences for a pose estimation algorithm [27]. The estimated camera pose is then used for rendering virtual objects on top of the hand, as shown in Fig. 1, and is used as an initial pose for further 3D scene acquisition and camera tracking, as shown in Fig. 3. In this way, users do not have to carry any tracking device or fiducial marker for AR with them.

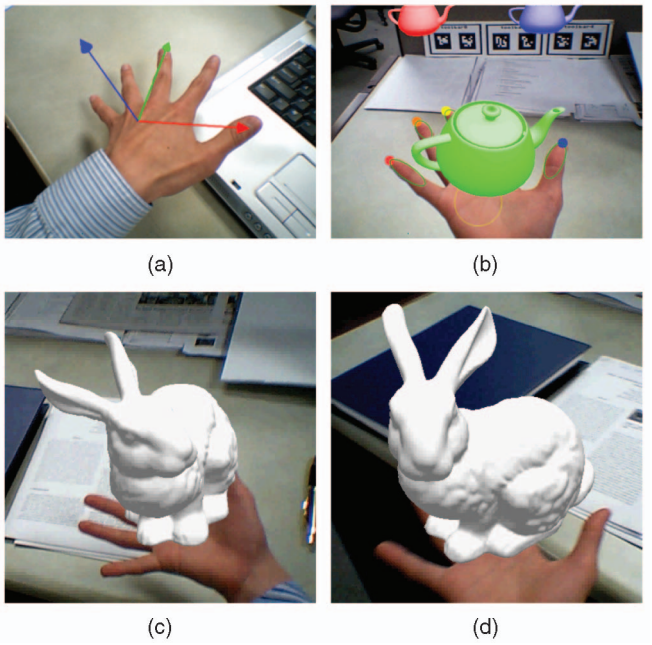


Fig. 1. Snapshots of Handy AR: (a) the hand's coordinate system, (b) selecting and inspecting world-stabilized augmented objects, and (c), (d) inspecting a virtual object from various angles.

2.2 Invariant Feature Detection

In the image formation process, several unknown variables play a role in varying the image properties [28], i.e., viewing angle, illumination, lens distortion, and so forth. Among these image variations, the perspective difference between frames constitutes a significant factor, especially when the camera baseline is large between the views. In these wide-baseline cases, the feature matching problem requires the features to maintain invariant properties for large differences in viewing angles and camera translation. The features also need to be discriminative in order to be used for recognizing the scene repeatedly and robustly. In this sense, these invariant features are often also referred to as landmark features.

In order to detect such distinctive landmark features in the AR environment, we apply the scale invariant feature transform (SIFT) [22] to extract keypoints from a captured video frame. The local extrema of the Difference-of-Gaussian (DoG) in scale space are detected as keypoints, characterized accurately by four components: the pixel location (x, y), the scale σ, and the major orientation θ. We then compute the local image descriptors for such keypoints as histograms of the image gradients, obtaining an element feature vector for each keypoint. The dimension of the descriptor depends on the number of histograms and the number of bins in each histogram, i.e., a 4×4 array of eight-bin histograms results in $4 \times 4 \times 8 = 128$ dimensions. The keypoints selected by the SIFT algorithm are shown to be invariant to scale and orientation changes, and the descriptors remain robust to illumination changes. From here on, we will refer to these keypoints together with their descriptors as SIFT features. SIFT features are shown to be useful both for drift-free tracking and for initially recognizing a previously stored scene [17].

For our purpose of detecting SIFT features for tracking the camera pose in AR applications, given a captured video frame, newly detected SIFT features are matched to the reference features in another frame or an overall 3D feature map. The SIFT descriptors can be compared and matched by using an approximate Best-Bin-First (BBF) algorithm as in [22], or by using vocabulary trees as in [29] and [30]. The point correspondences between the features in the captured frame and the reference frame can be used for estimating a camera pose [27], based on the world coordinates of the SIFT features in the reference frame. This process of matching new SIFT features to previously stored features provides a simple track-by-detection method [24], which is one component of our hybrid feature tracking mechanism that will be described in Section 3.3.

Since today's available computing power is insufficient to detect and match SIFT features for every frame in real time, we take two additional measures to enable real-time performance: limit the search space and perform SIFT detection asynchronously in a separate thread. In our implementation, we reduce the scale of the search space by skipping the initial image doubling step (cf. [22, Section 3.3]). In this way, computation time is reduced at the cost of rejecting small high-frequency features. The challenge of using SIFT detection as fast as the processing power allows is met by our multithreaded framework running a video capturing thread and a SIFT processing thread separately as explained in Section 3.5.

2.3 Optical-Flow-Based Tracking

Tracking features between two consecutive image frames is considered a small-baseline tracking problem in the sense that the transformation from the image at time t to the image at time $t + dt$ for small dt can be modeled using the translational model. Computing the optical flow for the feature points provides us an algorithm of frame-to-frame feature tracking [31] that can be described as follows:

Given two images $I(x)$ at time t and $I'(x)$ at time $t + dt$ with a fixed size window, we compute the spatial derivative (i.e., image gradient) $\nabla I = (I_x, I_y)$ and the temporal derivative I_t. Then, the translation $u(x, t)$ of a pixel x at time t is computed as follows:

$$G(x) \doteq \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}, \tag{1}$$

$$b(x,t) \doteq \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}, \tag{2}$$

$$u(x,t) = -G(x)^{-1} b(x,t). \tag{3}$$

In order to compute $u(x, t)$ from the equations above, appropriate feature points x should be selected so that the 2×2 symmetric matrix $G(x)$ is invertible. This gives the criteria for Shi and Tomasi's *Good Features to Track* [32] algorithm, which examines the eigenvalues of $G(x)$ and chooses the feature point if the minimum eigenvalue is larger than a certain threshold, i.e., $min(\lambda_1, \lambda_2) > \lambda$. Another well-known criterion for selecting feature points is provided by the Harris corner detector [33], which prefers

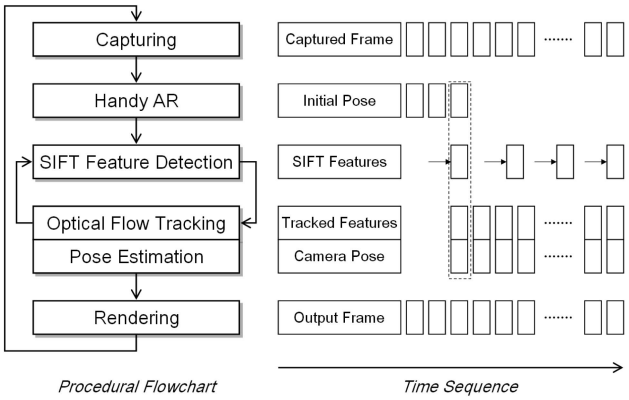


Fig. 2. Procedural flowchart and conceptual time sequence for each tracked entity, establishing a coordinate system for a tabletop AR environment, using Handy AR and our hybrid feature tracking method.

to select well-textured image features that have high values of the Harris response R:

$$R = det(G) - k \times trace^2(G). \quad (4)$$

These feature detection and tracking algorithms are widely used with efficient implementations [21] for small-baseline feature matching tasks, of which tracking algorithm is used for frame-to-frame tracking in our hybrid feature tracking method.

3 Method Description

We establish a coordinate system for an unprepared tabletop AR environment using fingertip tracking and introduce a hybrid feature tracking method that combines landmark feature detection and optical-flow-based feature tracking in a real-time approach. A camera pose is estimated from the tracked features relative to the 3D points that are extracted from the scene while establishing the coordinate system for the environment. An overall flowchart for this approach is illustrated in Fig. 2. The established coordinate system is then propagated to the environment and the tracking region is expanded as we detect new features in the scene incrementally. We use the Handy AR hand tracking method for user interaction with virtual objects in AR.

3.1 Initializing a Coordinate System

Using the Handy AR system, we enable a user to initialize a coordinate system for AR using the simple gesture of putting the hand on the working space surface. The procedure of initializing a coordinate system is summarized as follows:

1. **User places** a hand on a desktop surface.
   - Handy AR estimates the coordinate system.
   - SIFT features are detected in the camera frame.
   - The coordinate system is propagated to the surface.
2. **User removes** the hand.
   - SIFT features are detected in the camera frame, matching previously stored features and providing new features in the space previously occupied by the hand.

While the hand rests on the surface plane, features around the hand are detected in order to establish the global coordinate system of a tabletop AR environment. Fig. 3 shows the steps of propagating the coordinate system: The camera pose from the Handy AR subsystem is used to project the detected landmark image features to the working plane, calculating their world coordinates. While tracking the features for each frame, the camera pose is estimated from feature point correspondences. Once the features in the scene are detected and the world coordinate system is initialized, the user may move the hand out of the scene and start interactions with the AR environment. Note that the area covered by the hand contains several feature points as shown in Fig. 3b. After moving the hand out of the view, as in Fig. 3c, the features previously clinging to the hand are not detected any more. Instead, there is new space to detect more features that will be filled in by the next iteration of SIFT detection, which allows us to estimate the camera pose more robustly while expanding the tracking region.

3.2 Computing 3D Feature Locations

In this section, we describe how we compute and map 3D landmark locations from the image features.

3.2.1 Planar Structures

When the features of the scene are lying on a planar structure, the 3D locations of the features can be computed by projecting them to the plane. For instance, when a user initiates a reference frame by putting a hand on the surface, the SIFT features are projected to a plane that is parallel to the hand. Using the estimated camera

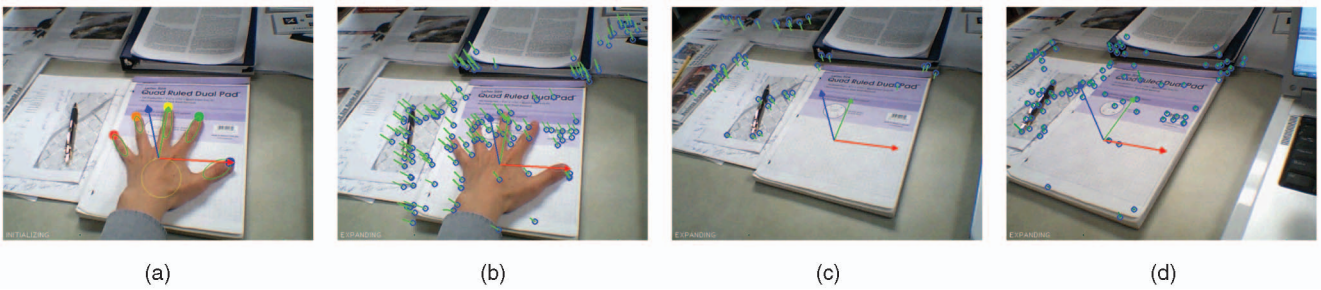


Fig. 3. Establishing a coordinate system using Handy AR (a), propagating it to the scene as we detect scene features (b). After moving the hand out of the scene (c), new features are detected from the area that the hand occluded (d).

pose from Handy AR, the 3D locations of the features are computed in a new world coordinate system according to the projection model:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = P_{3\times4} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}, \tag{5}$$

where $(x\ y\ 1)^\top$ and $(X\ Y\ Z\ 1)^\top$ are homogeneous representations of the SIFT features in the image plane and the world coordinate system, respectively, and $P_{3\times4}$ is the projection matrix of the camera, whose fixed intrinsic parameters are assumed to be known through one-time initial calibration step. In our implementation, we assume the Z value to be a certain height ($Z = 0$ places the tabletop plane just below the hand) and derive equations to calculate the X and Y world coordinates of each SIFT feature from the (x, y) positions in the captured image.

3.2.2 Nonplanar Structures

In case the scene is nonplanar, the 3D locations of the feature points can be computed using two or more views of the camera. This is the traditional *triangulation* problem, which can be solved by the linear method followed by optional nonlinear refinement steps [34]. For example, suppose that a feature point is tracked between two images with the projection model:

$$x_1 = P_1 X \quad \text{and} \quad x_2 = P_2 X,$$

where $x_1,\ x_2 \in \mathbb{R}^3$ are the image measurements of the feature point in homogeneous representation, P_1 and P_2 are the 3×4 projection matrices of the two views, and $X \in \mathbb{R}^4$ is the 3D location of the feature point in homogeneous coordinate representation. From these two equations on measurements and projection matrices, we can construct a linear system for the unknown variable X as detailed in [34]. The solution of the linear method can be refined using Bundle adjustment, which can be performed in parallel with tracking features as demonstrated in [20].

We have implemented these 3D mapping algorithms in our framework but have not yet reached the efficiency demonstrated in [19]. The main contribution of this paper lies in the area of hybrid feature tracking using our multithreaded system architecture, and we demonstrate its real-time feasibility without loss of generality for planar workspaces. Raising the efficiency and robustness of applying our approach to 3D-structure-from-motion techniques is left for future work.

3.2.3 Rejecting Outliers

In order to match the SIFT features more accurately, we use the RANSAC algorithm [35] to eliminate outliers before pose estimation. RANSAC is particularly useful for removing features on vastly nonplanar objects or orthogonal planes when we make planar assumptions for the scene structure. As long as the majority of the visible features remain on the working plane, RANSAC will remove features outside of this planar workspace, thus maintaining the planarity condition even for scenes that consist of more complex geometry than a single working plane. Additionally, we employ the RANSAC algorithm for imposing the epipolar constraint between two or more images in order to reject false matches of SIFT features.

3.3 Hybrid Feature Tracking

In order to track the detected features over a sequence of frames, an optical-flow-based tracking algorithm is used for each successive pair of frames. The interest points in one frame are tracked with respect to the previous frame in 2D by iteratively computing fast optical flow using image pyramids [21]. Instead of the interest points detected by Shi and Tomasi [32], our tracking uses distinctive landmark features (i.e., SIFT features). Therefore, the hybrid feature tracking is a combination of SIFT feature detection and optical-flow-based tracking algorithms.

Tracking distinctive image features can easily become prohibitively expensive for a single threaded approach. In [17], a single thread processes the detection and the tracking sequentially. However, a problem arises because of the relatively long computation time for SIFT feature detection (on the order of 10-20 real-time frames on state-of-the art desktop computers). As a result, the sequential approach stalls tracking significantly when applied in real-time applications.

Therefore, this paper proposes a multithreaded approach for the hybrid feature tracking, in which separate threads are designated for detecting SIFT features and for tracking the features frame-to-frame via optical flow. New interest points are solely introduced by SIFT detection and the interest points to track are selected from the detected SIFT features in a multithreaded manner. The problem that tracking gets stalled during computation of the SIFT features is solved by the nature of multithreading, sharing time slices among the different tracking activities. However, another problem arises: the frame that SIFT features are found in lies in the past at the time the SIFT feature information becomes available (cf. Fig. 4). Consequently, the optical flow tracking needs to be matched with the past frame in which SIFT features were detected.

In Table 1, the algorithm of the proposed hybrid feature tracking is described as a pseudocode. The tasks are designated to two separate threads: detecting SIFT features and tracking them using optical flow. The details of the proposed algorithm are explained in the following sections.

3.3.1 Introducing New Features

In the beginning of the tracking procedure, there are no features yet to track. Therefore, new features have to be introduced. In our proposed method, we detect SIFT features and use them for continuous frame-to-frame tracking. Suppose that, at time t_0, a set of SIFT features are detected from the image in the past time, $frame(t_{past})$. In Fig. 4a, the detected SIFT features are illustrated as blue circles. Since we have two images of $frame(t_{past})$ and $frame(t_0)$, we run the frame-to-frame optical flow tracking algorithm between $frame(t_{past})$ and $frame(t_0)$ on the detected SIFT features (illustrated as blue asterisks in the figure). From this, the features of $frame(t_0)$ are introduced, shown as yellow circles with bold outlines. Now that we introduced new features to track, the tracking mechanism continues iteratively as described in Section 3.3.2.

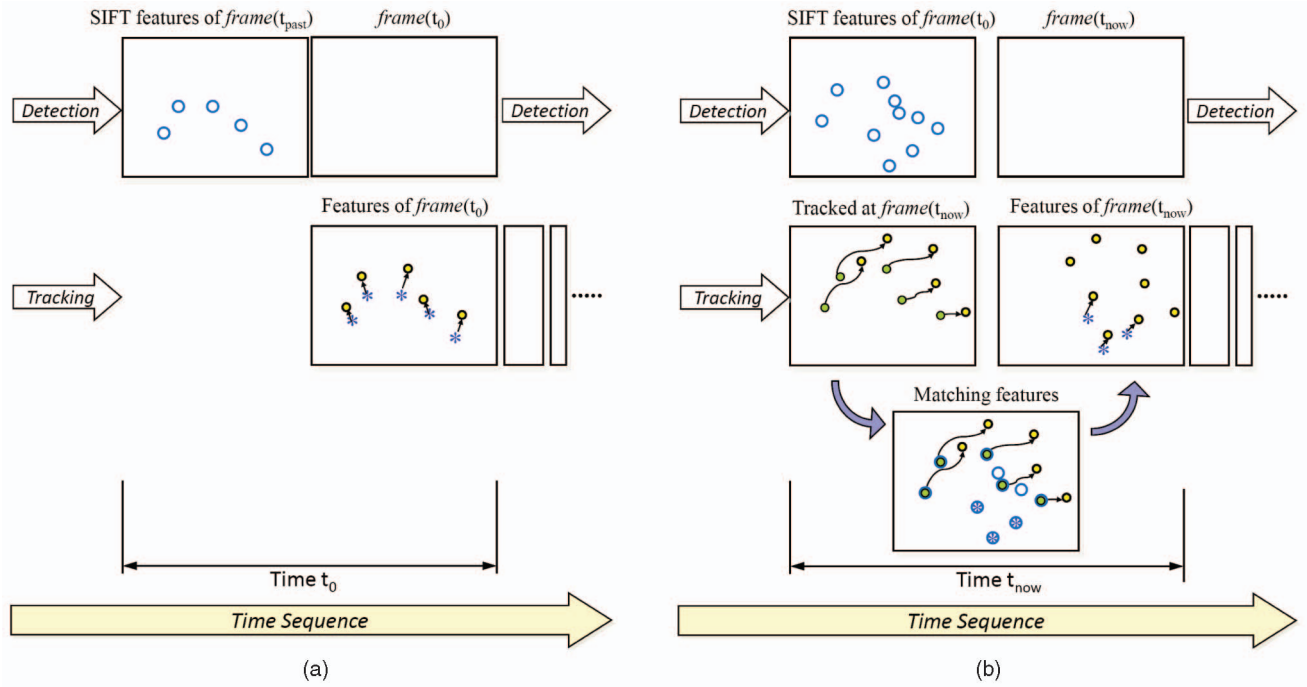


Fig. 4. Illustration of tracking events in frame sequences for our hybrid feature tracking mechanism.

3.3.2 Tracking Existing Features

So far, we have introduced new features at a time t_0, and we have continued to track these features using optical flow, as well as issued another SIFT detection cycle (at time t_0), which will yield results at time t_{now}. This situation at time t_{now} is illustrated in Fig. 4b. From the past time t_0, the target frame for SIFT feature detection is saved as $frame(t_0)$, while fast optical-flow-based tracking has been performed on subsequent frames until $frame(t_{now})$ at time t_{now}. Suppose that at time t_{now} (typically half a second later than t_0) the SIFT detection is finished. At this moment, we have a set of freshly detected SIFT features from $frame(t_0)$ and another set of features that have been tracked by optical flow from t_0 until t_{now}. Now, the SIFT features are matched to the previous positions of tracked features, illustrated as overlapping blue circles and green circles along the curved trails in Fig. 4b, respectively. Additionally, new interest points from unmatched SIFT features are added to $frame(t_0)$ if they are more than a threshold distance (in our implementation: 10 pixels) apart from any other currently tracked feature, illustrated as asterisks in the figure. Then, the newly added features of $frame(t_0)$ are tracked to $frame(t_{now})$ by computing optical flow directly between the two frames the same way we did for introducing new features in the previous section. In this way, newly detected SIFT features are matched to the tracked features.

During these matching and tracking procedures, some features will likely have been dropped due to unsuccessful optical flow tracking and the RANSAC algorithm during pose estimation as described in the previous section. Discarding such outliers helps increase the tracking system robustness and accuracy of camera pose estimation.

TABLE 1
Pseudocode for the Threads of (a) Detecting SIFT Features and (b) Tracking Features Using Optical Flow

```
(A) DETECTING SIFT FEATURES:
  BEGINLOOP
    Set t_sift = t_now.
    Detect SIFT features from Frame(t_sift).
    Identify the SIFT features.
    Indicate that SIFT features are READY.
  ENDLOOP

(B) TRACKING FEATURES:
  BEGINLOOP
    IF there are tracked features,
      Optical Flow: Frame(t_now-1) → Frame(t_now).
      RANSAC with epipolar constraint.
    ENDIF
    IF SIFT features are READY,
      Introduce new features.
      Optical Flow: Frame(t_sift) → Frame(t_now).
    ENDIF
  ENDLOOP
```

3.3.3 Relocalization

In the case that we lose all features during optical flow tracking, the system needs to relocalize the camera to the tracked region. With the help of the discriminative SIFT features, this relocalization problem can be approached in a very similar fashion to introducing new features to start with. The difference is that we have the features of the previously tracked region to match with the newly detected features. When we identify the detected features, matching their descriptors to the existing features of the landmark feature map provides us the capability of relocalizing a camera to a current (but temporarily unregistered) scene, as well as recognizing a scene among a set of previously stored scenes.

Fig. 5. Expanding a tracking region while looking at the scene from different perspectives. Starting from the initial viewing angle (a), the camera moves to the right while expanding the tracking region (b), (c), resulting in a stabilized coordinate system (d). Note that features in (a) and (d) are being tracked from significantly distinct regions.

As described so far, the benefit of the proposed hybrid feature tracking mechanism is that we can utilize the discriminative invariant features while providing faster frame-to-frame tracking capability. In our implementation, the SIFT detection thread runs at about 2.1 fps and optical flow at 26.3 fps when tracking and expanding a region. A more detailed performance analysis for our approach is presented in Section 4 and discussed in Section 5.

3.4 Expanding a Tracking Region

When a coordinate system is established in a specific reference frame, scene tracking would so far be limited to the overall field of view covered by that specific captured video frame. In addition to the limited field of view, the initial frame contains the hand image, as shown in Fig. 3b. We want to introduce new features dynamically as we move the hand away or shift physical objects around on the workspace. In order to increase the robustness of tracking and the coverage area for a tabletop workspace, it is necessary to expand a tracking region incrementally, covering large viewing angles from different perspectives. This is also helpful for reliable feature detection and tracking. Note that the distinctive image features [22] are already invariant to some changes in rotation and scale.

Given an estimated camera pose, 3D locations of newly detected distinctive features are computed as described in Section 2.2. They are added to the set of features that represents the scene for camera tracking, expanding the covered region. As shown in Fig. 5, SIFT features are detected at multiple scales and orientations from different perspectives, increasing robustness of our hybrid feature tracking considerably, compared to methods relying solely on optical flow. After a user establishes a coordinate system using one of their hands as described in Section 3.1, the hand-covered region exhibits no features due to previous occlusion (Fig. 3c). Also, new viewing angles may introduce new areas of the tabletop that were previously unexposed. By detecting new features in such areas, as shown in Fig. 5, we increase robustness and enable users to interact on a larger workspace than the original reference frame covered, providing a useful, growing tabletop AR environment.

3.5 Multithreaded Approach

In order to provide real-time performance, we process operations in a multithreaded manner. We divide the necessary tasks in our AR framework into the following functional groups: capturing a video frame, performing Handy AR, detecting SIFT features, tracking the features using optical flow and performing pose estimation, and rendering an output frame. Separate threads are allocated to each group, as indicated by the rows in Fig. 2.

Since these threads share some common data, such as an image frame, a set of SIFT features, and camera pose parameters, as shown in Fig. 6, we must synchronize some operations within these threads. Among all threads, the SIFT detection thread and the fast feature tracking thread have the strongest need for synchronization, because they are to combine the detected and tracked features seamlessly. Since the SIFT detection procedure takes about 10 times as long as optical flow tracking, we detect features *asynchronously*, store them in the SIFT thread's local variables, and just synchronize the two threads at the time of feature matching.

Since a captured video frame is used by every thread, we synchronize the frame data as well. However, declaring access to a frame to be a critical section would prevent the capturing thread from proceeding quickly in a nonblocking fashion. Instead, we store a time stamp with each captured frame, providing the threads with information to distinguish frames, implementing *soft* synchronization. For example, the SIFT detection thread and the optical flow tracking thread communicate with each other on which frame is ready for feature detection by time-stamp comparison. The last known good SIFT detector frame is stored with its time stamp until it is updated by the SIFT thread.

The rendering thread may be synchronized with other threads depending on the data it displays. For example, if

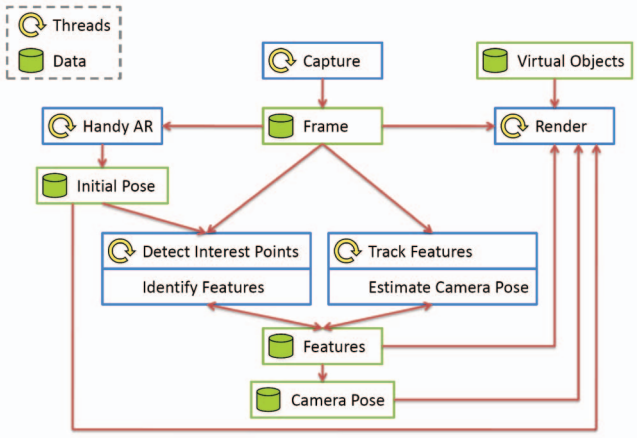


Fig. 6. Diagram of dependency among threads and data. Since the SIFT detection thread and the optical flow tracking thread critically share the feature data, synchronization is required.

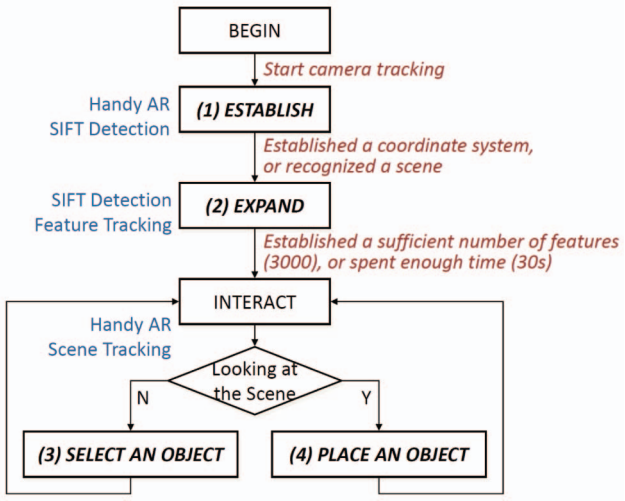


Fig. 7. Flowchart of the AR application task sequence, as used in performance experiments.

features are to be displayed (for debug purposes, as in our video), the rendering thread also needs to be synchronized for feature data access. The behavior of our multithreaded framework is examined in Section 4.1.1 and Fig. 8. In addition to synchronizing the threads, we also pay attention to adjusting their execution priorities for tuned performance. Higher priority for the capturing thread allows for low-latency playback. Yielding to other threads is crucial to maintain a consistent frame rate for each thread.

3.6 Recognizing a Scene

As we described in Section 3.1, distinctive features are detected in the scene while a coordinate system for a new tabletop environment is established. These features are also used for recognizing different previously stored scenes, for resuming tracking after the user looked away from the scene temporarily and for recognizing a scene from various viewing angles.

We match the detected SIFT features between an active frame and previously stored features, following the object recognition algorithm of Lowe [22]. The scene that has the maximum number of matched features among all scenes is recognized to be identical to the current scene if the number of matched features exceeds a certain threshold. This recognition step is performed when a user looks at a new place so that the system fails to detect enough features from the previously tracked scene. In our implementation, the scene recognition time increases proportionally to the number of saved scenes. If the number of matched features to the closest stored scene is below the threshold, tracking commences in a new scene.

Once a scene is recognized, the AR space can be appropriately set up according to a user's purpose: Augmentations can be fixed to the environment so that a user experiences different AR spaces in different physical environments, or users can carry "their AR desktop" with them to any new tabletop environment to resume previous interactions. In the latter case, they may have to adjust the positions of virtual objects if those coincide with locations of physical objects. The hand-based user interface methodology for such adjustments is presented in Section 4.2.

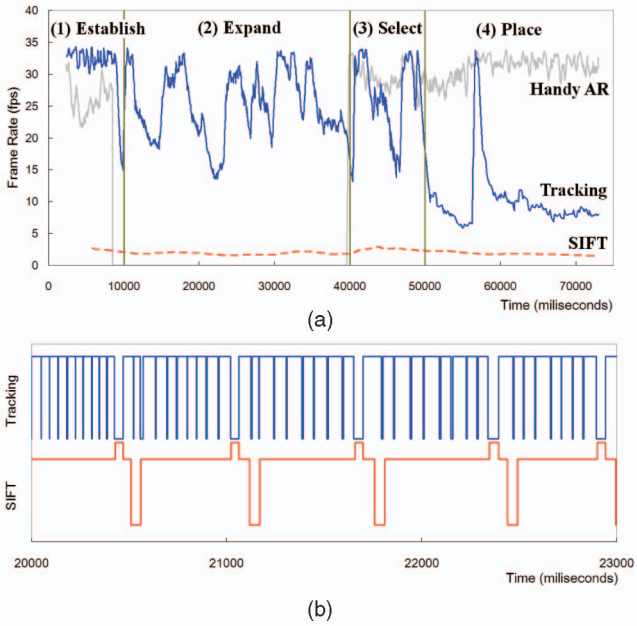


Fig. 8. (a) Frame rates of threads processing Handy AR (gray), SIFT detection (dashed red), and feature tracking (blue) over time. (b) Thread activity for SIFT detection (red) and feature tracking (blue). The threads are partially synchronized.

4 Evaluation Results

We tested our implementation of hybrid feature tracking with regard to the speed and robustness of the system. We especially examined the multithreaded framework's behavior with respect to real-time performance. Accuracy was tested by monitoring the achieved registration in a sample AR application, placing and stabilizing virtual objects within a tabletop scene. The results show that our method is useful for easily establishing a coordinate system for tabletop environments, is robust against drift errors, and is able to swiftly recover from tracking failures. Our test system detects SIFT features in a reduced scale space by using a tuned implementation of the SIFT detector [36], and it runs our hybrid feature tracking method using the presented multithreaded framework for real-time AR.

The experiments were performed on a small laptop computer with a dual-core 1.8-GHz CPU, using a USB 2.0 camera with 640×480 resolution. For Handy AR, the internal computations were processed in 320×240 resolution for faster speed while still maintaining reasonable accuracy [4]. Intrinsic camera parameters were calibrated in a one-time offline step using a 6×8 checkerboard pattern and the OpenCV [21] implementation in [27].

4.1 Evaluations

Evaluations for our system were performed with regard to the speed of the system and robustness of the hybrid feature tracking algorithm. The (typical) task sequence we used for our evaluations is described in Fig. 7:

1. In the initial stage of the program, the user establishes the coordinate system either by hand gesture using Handy AR (described in Section 3.1) or by the system recognizing a stored scene (described in Section 3.6).

TABLE 2
Average Frame Rates of Each Thread (in Frames per Second)

	Interaction Steps			
Threads	Establish	Expand	Select	Place
Capture	46.8	33.2	47.1	17.3
Render	37.7	28.0	37.9	18.6
HandyAR	26.5	Idle	30.8	
SIFT	2.1			
Tracking	Idle	26.3	Idle	13.8

2. The tracking region is expanded while the system performs hybrid feature tracking.
3. When the region is expanded enough for user interactions, the user can select an object.
4. Otherwise, the user can place an object.

4.1.1 Speed of Multiple Threads

As far as speed is concerned, the goal was to achieve real-time performance: strive for 30 fps while supporting interactivity with at least 15 fps. Table 2 lists the average frame rates of each thread for the different interaction steps of a typical tabletop AR session: establishing a coordinate system, expanding a tracking region, selecting a virtual object, and placing it to the AR environment. Overall, threads run at around real-time frame rate during the establishing, expanding, and selecting phases. The capturing thread and the rendering thread run at around real-time speed, providing fast feedback. The Handy AR system runs at 26.5 fps at the beginning stage of establishing a coordinate system and at 30.8 fps while interacting with virtual objects stabilized in the environment. After propagating the coordinate system to the environment, the hybrid feature tracking threads (SIFT detection and optical flow tracking) provide real-time pose estimation updates at around 26.3 fps. Because the rendering thread is running at a similarly fast speed, the user perceives fast and smooth registration of augmentations. When a user is placing a selected virtual object into an AR environment, the frame rate drops because of increased occlusion and need for synchronization in the scene. However, every thread still runs at interactive frame rates (i.e., around 15 fps), including both Handy AR and feature tracking.

We measured the behavior of multiple threads running as a single framework as shown in Fig. 8a, while a user performed the same sequence of actions mentioned above. For Fig. 8a, these phases were, over time

1. establishing a coordinate system (initial 10 seconds),
2. expanding the tracking region (30 seconds),
3. selecting an object as looking away from the tracked region (10 seconds), and
4. placing virtual objects into the environment (25 seconds).

The SIFT detection thread runs as a background process, showing a steady frame rate at around 2.1 fps. The Handy AR and the feature tracking threads become idle when they are not required to be processed, as shown in Table 2, and run at real-time speed when one of them is running (for the expansion or selection step). When a user places a virtual object into the environment, Handy AR maintains real-time speed, while feature tracking runs at interactive speed.

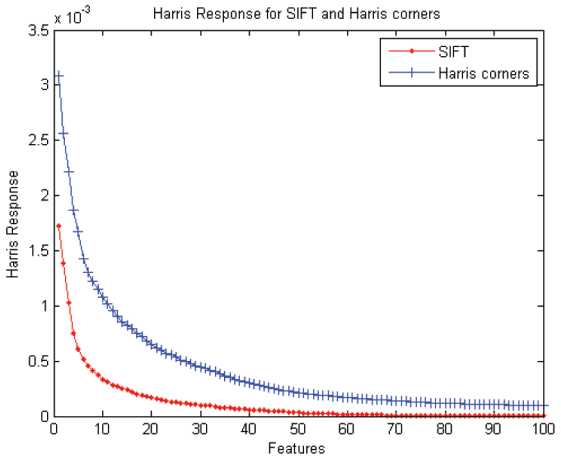


Fig. 9. The Harris responses of features from SIFT and Harris corner detectors.

We also measured the behavior of the SIFT detection thread and the feature tracking thread, as shown in Fig. 8b. The status of each thread is plotted, indicating whether it is idle (bottom level), working asynchronously (middle level), or working synchronized (top level). The two threads are synchronized when the SIFT detection thread adds new features to global variables. This synchronization is shown in the plot as the highest steps of the SIFT thread's plot and the lowest steps of the feature thread's plot meet periodically in Fig. 8b.

4.1.2 Comparison between SIFT and Harris Corners

Because our hybrid feature tracking mechanism relies on optical flow frame-to-frame tracking, we need to analyze how well the SIFT features are tracked by such an optical-flow-based algorithm. The traditional Shi and Tomasi algorithm [32] using the Harris corner detector chooses features with high Harris response as described in Section 2.3.

As shown in Fig. 9, the Harris corners detected by the Shi-Tomasi algorithm have overall higher Harris responses than the SIFT features. The figure was plotted by sorting the top-100 Harris responses of the features, averaged from multiple frames of a cluttered desk scene. When we analyze the figure in more detail, about 30 percent of SIFT features have higher responses than the 100th Harris corner. As described in Section 2, this difference is caused by the different feature detection criteria of the SIFT detector against the Shi-Tomasi detector, and the scale differences of the image features. As a result, this implies that not all SIFT features are appropriate for being tracked well by the optical flow algorithm. However, at the same time, this indicates that we can utilize these *good* 30 percent of SIFT features by tracking them with a faster frame rate. Note that the other 70 percent of SIFT features with low Harris responses can be used for other tasks in our approach, such as robustly recognizing scenes and relocalizing after tracking failures, by matching their descriptors. The benefit

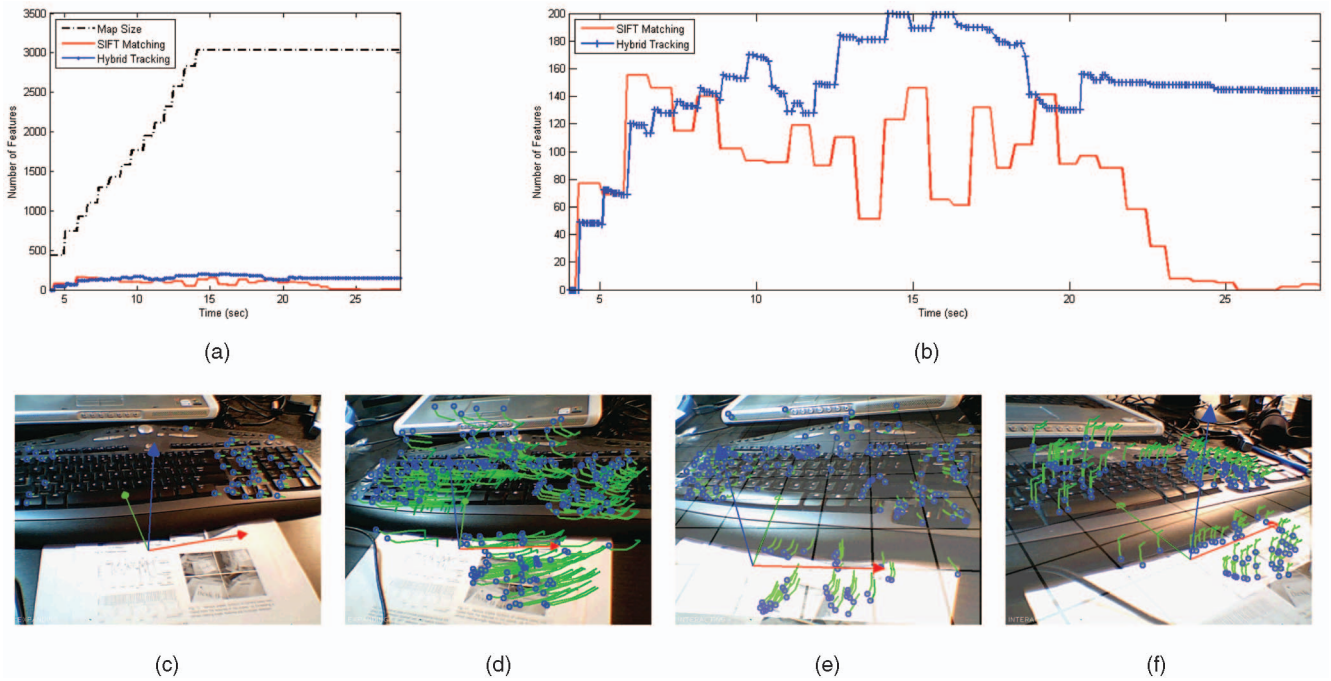


Fig. 10. A sample sequence for recognizing a stored scene and expanding a tracking region. (a) The number of features matched by SIFT descriptors and by hybrid tracking, while increasing the map size up to 3,000 features. (b) Zoomed plot of the number of features. (c), (d), (e), (f) Snapshots of the sequence: (c) recognizing the scene, (d) tracking large number of features while expanding, (e) reaching the enough map size, and (f) viewing from an extreme angle.

of our hybrid feature tracking mechanism is shown in Section 4.1.3.

4.1.3 Hybrid Feature Tracking Quality

Fig. 10 shows a sample sequence of recognizing a stored scene and moving the camera around while expanding the tracking region from various viewing angles. Fig. 10a shows the number of features over time. The dashed line shows the size of the map, growing up to 3,000 features. The red line shows the number of SIFT features detected for a frame and matched to the map features. After the map reaches a size that is sufficient for starting interaction (in our experiment 3,000), no new features are added to the map. Therefore, the number of SIFT features matched to the map varies according to the viewing angle. For example, after 25 seconds into the sequence, few SIFT features are matched because of the extreme viewing angle.

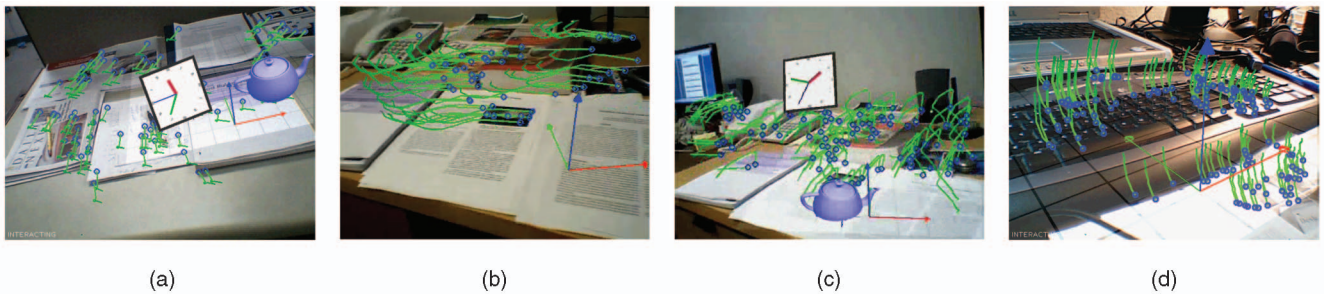
The blue line with crosses shows the number of features tracked by our proposed hybrid feature tracking mechanism. The different behavior of SIFT matching and the hybrid tracking can be seen clearly in Fig. 10b. In the early stages of expanding the region, only some portion of SIFT features are tracked. However, the frame-to-frame tracking maintains its tracked features while introducing newly detected SIFT features to track. This is apparent in the period between about 8 and 16 seconds, when the system is tracking more features than there are matched SIFT features. Fig. 10d corresponds to this situation.

Moreover, after 20 seconds, the number of matched SIFT features decreases because of the narrow viewing angle of the camera that the map does not cover well. From this extreme angle, the SIFT features are not matched well, since there is not a sufficient number of features to estimate the camera pose. However, by continuing the hybrid feature tracking, the features detected from previous frames are maintained through frame-to-frame tracking, providing more than 150 features—enough to estimate the camera pose, as shown in Fig. 10f. This behavior of the hybrid feature tracking shows that by combining the two different feature detection and tracking algorithms, we can achieve better performance in the sense of speed and robustness.

Additionally, we examined the robustness of the hybrid feature tracking approach while moving a camera in the scene in systematically varied ways, exhibiting stable location and orientation, slightly shaky or jittery motion, smooth movement with constant speed, abrupt quick motion, constant fast motion, looking at a completely new space, and returning back to the original space. The tracking method showed successful feature detection in the scene and tracking was mostly continuous, as illustrated in Fig. 11. Under fast motion, the system loses track of the features but quickly recovers to track them whenever the distinctive features are redetected. With our tabletop AR environment scenarios, this robust recovery is very useful to reregister augmentations swiftly whenever tracking is lost. Dead reckoning could be used to avoid disruption of AR annotations during frames that suffer from tracking failure.

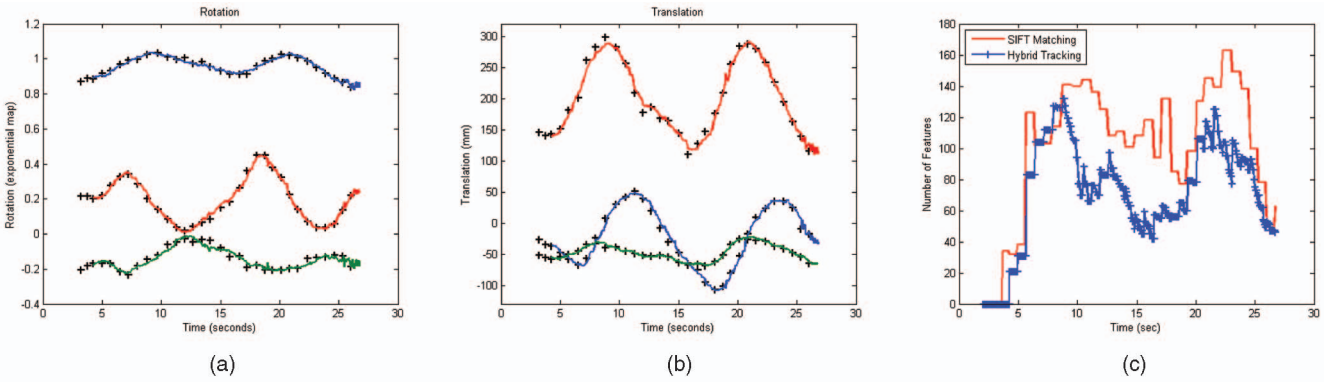
4.1.4 Camera Pose Estimation

We tested the performance and quality of camera pose estimation while looking at the scene from various angles. We compared the estimation result from matching only SIFT features to the estimation from utilizing our hybrid tracking mechanism. In Figs. 12a and 12b, the camera's estimated rotation and translation are displayed. In this



(a) (b) (c) (d)

Fig. 11. Snapshots of tracking under various camera motions, displayed with the trails of tracked features. (a) Rotating. (b) Moving to the right. (c) Jitter motion. (d) Looking down.



(a) (b) (c)

Fig. 12. Camera pose estimation of (a) rotation in exponential map and (b) translation in (x, y, z). In the estimation plots, the black crosses indicate the estimation from SIFT matching, while the red, green, and blue solid lines are the estimation from the hybrid tracking. (c) The number of features used for the pose estimation by each method.

(a) (b) (c) (d)

Fig. 13. (a), (b), (c) Various angles of camera pose, estimated from the features in the scene. (d) Exceeding a certain viewing angle, SIFT features are no longer matched.

experiment, we assume planarity of the work plane and compute the 3D locations of the features as in Section 3.2.1. Note that the SIFT matching provides less frequent updates of the estimation, shown as black crosses. In order to perform a fair comparison between the two methods, the features are tracked only around the initial tracking region, resulting in a smaller number of features tracked by the hybrid tracking, as shown in Fig. 12c. The result indicates that our hybrid tracking provides not only frequent updates of the pose estimation but is also closely matched to the SIFT trace. The colored lines consistently interpolate between the SIFT-based estimations, without producing radical outliers or systematic bias.

Sample frames from our test runs illustrate specific properties of our tracking solution: While the viewing angle onto the scene may vary drastically from the initial camera pose and the system still copes due to the distinctive SIFT features, as shown in Fig. 13. Under shallow angles like Fig. 13d, however, the SIFT features are not easily matched because they are not perspectively invariant. When tracking the features over time, those viewing angles will still be covered, however, by using optical flow features and newly detected features while expanding the tracking region as described in Section 3.4. Only when the user arrives at such angles through rapid motions will tracking fail until stored SIFT features are rediscovered.

4.2 Application Prototype

We have implemented a proof-of-concept application for tabletop AR environments using Handy AR and our hybrid feature tracking method. A user establishes coordinate systems for several desks separately, as an initial step for each space. The distinctive features are stored persistently, noting their world coordinates. When the user sits at one of the desks and starts the AR application, the features are detected and matched to recognize the scene the user is viewing. Fig. 14 shows four scenes being recognized successfully and the annotations being registered according

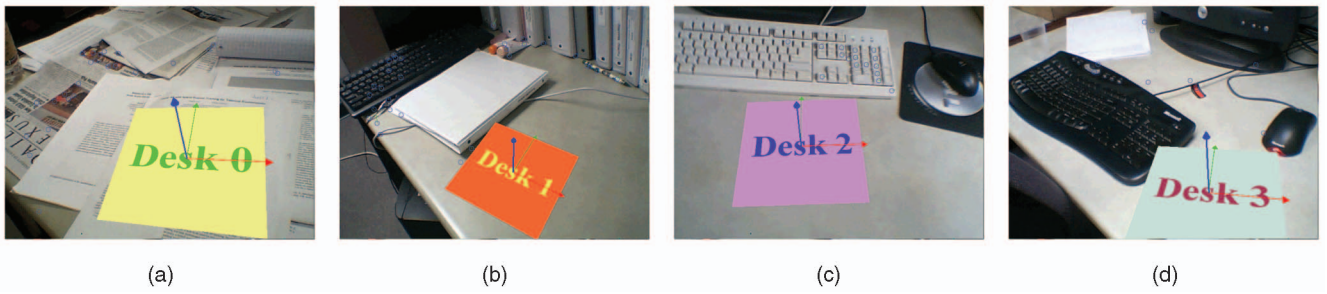


Fig. 14. Snapshots of recognizing a scene. Each desk is indicated with an annotation.

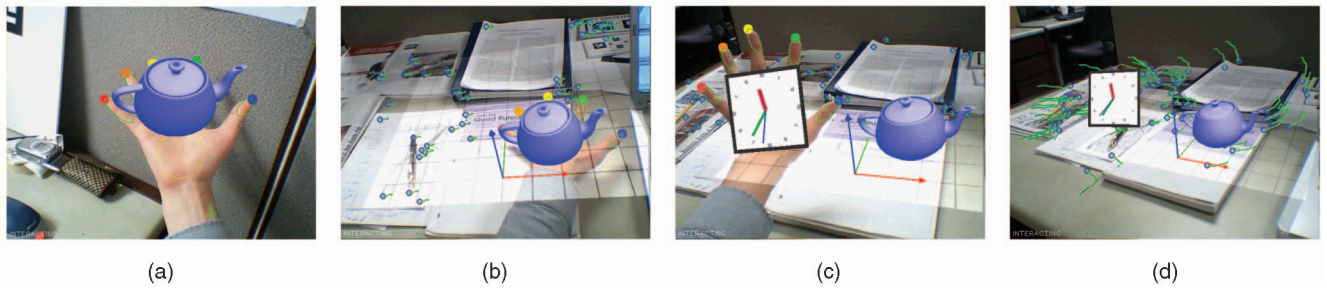


Fig. 15. Snapshots of interacting with the tabletop AR environment. Handy AR is used for selecting a virtual object (a), and placing the virtual objects. Here, teapot (b) and clock (c). (d) The tabletop AR environment is tracked with its virtual objects stabilized.

to the coordinate system that the user established previously. Through the same framework that recognizes each space, the system can also be used for just setting up one AR coordinate system, so that a user can bring an individual AR environment to several locations. This enables users to personalize their tabletop AR system while moving into new locations, as long as their coordinate systems are set up once with Handy AR on the new tabletop environment.

When a coordinate system is established for a tabletop AR environment, a user can place virtual objects in the workspace. By using the Handy AR system, a user can select an object and then place it into the established coordinate system. In absence of a trigger event simulating a mouse button, we cycle through a set of selectable objects when a Handy AR pose is detected in the initial (base interaction) mode. Objects are rendered on top of the hand in a rotating sequence of 1 second intervals. Fig. 15a shows the user selecting a teapot in this manner. Breaking Handy AR tracking (e.g., by making a fist for "grabbing" the displayed object) selects the currently displayed object, and the interaction mode switches to placement mode. Objects can be placed using either hand, so it is entirely possible to select an object with your nondominant hand and place it into the scene with your dominant hand since Handy AR detects either one.

As shown in Figs. 15b and 15c, when a camera pose is estimated from both the user's hand and hybrid feature tracking in placement mode, the object's transformation matrix in the world coordinate system is computed by multiplying the *hand-to-camera* and the *camera-to-world* transformation matrices so that the hand's transform is used to place the object in the environment's coordinate system. The *hand-to-camera* matrix is retrieved from the camera pose estimated by Handy AR, and the *camera-to-world* is the inverse of the camera pose from our markerless tracking system. When a hand gesture is triggered to place the object (by, e.g., again making a fist to break fingertip tracking), the transform of the object is stored to keep it stabilized in the environment, as shown in Figs. 15b and 15c. Then, the stabilized objects can be viewed from different angles, as shown in Fig. 15d. Note that the green trails of features are displayed in these figures for debug purposes only.

5 Discussion

Since Handy AR is using a skin-color-based hand segmentation algorithm, other objects that have similar color will also be detected as candidate regions for initial detection. Once the hand is detected, blob tracking ensures that it is not lost even in presence of other skin-color objects unless there is considerable overlap. In case of initial hand detection failure, the user still needs to be able to let the system work properly. A simple workaround is to use a masking object of different color (e.g., a sheet of paper) as a temporary background during detection. After setting up the AR coordinate system by detecting features in the scene, the user may remove the hand and also the masking object. Since the features in or around the hand will not be used as further features to track as shown in Fig. 3c, the subsequent camera tracking can detect new features in the scene, expanding the tracking region appropriately.

Our current method for propagating the established coordinate system makes the assumption that the workspace is a flat surface, providing at least a dominant plane. The first tracked reference frame should provide reasonable registration for tabletop environments and the tracking region is further expanded while detecting more features over a larger tabletop space. Since common objects that lie on tables, such as sheets of paper, books, notes, or pens are often flat or of low height, this planar assumption is not too restrictive for an office desktop scenario. The features on

nonplanar objects will be discarded when applying the RANSAC algorithm to get rid of outliers for estimating a camera pose based on the planar assumption.

More sophisticated algorithms could enhance the camera pose tracking accuracy and lift the planar assumption. Moving a camera around the scene while establishing the AR environment, the structure of the 3D scene can be reconstructed, either online or offline. A practical online approach of building a 3D landmark map is presented in [19], where a probabilistic framework enables a reliable single camera tracking system, or in [20], in which large numbers of low-quality features are used in fast structure-from-motion computations. Our hybrid feature tracking algorithm can be used for detecting and tracking landmarks to combine with these approaches. For example, we expect to generate a map of 3D points with their SIFT descriptors in addition to the image patches of their methods. On the other hand, offline approaches [37] usually process the input data as a batch job. Although these offline approaches are not feasible yet to be used in real-time applications, combining online and offline algorithms can be beneficial to enhance AR tracking systems.

Our multithreading approach enables the AR system to balance the work load among different tasks while running them simultaneously, letting the operating system control the threads to share the available processing power. In this way, our hybrid feature tracking mechanism works smoothly with a heavyweight SIFT detection thread as a likely background process, helping the lightweight fast tracking method to be more robust to drift. This idea is similar to combining multiple sensor modalities in a system, such as inertial sensors, GPS, optical devices, and vision-based systems. A similar approach of fusing different data sources, presented in the more formal framework of an Extended Kalman Filter [38], can be used for processing multiple tracking sources. However, we need to pay more attention on designing such systems running with multiple threads, because sharing data among several threads makes it hard to predict complex runtime behavior.

As for the behavior of the hybrid feature tracking, the mechanism described in Section 3.3 works well not only for continuous tracking but also for starting and recovering states. In the starting state, there are no features to track yet. Therefore, after about half a second lead-in time, all detected SIFT features are added as interest points to track. However, because of the direct optical flow matching between $frame(t_0)$ and $frame(t_{now})$ as described in Section 3.3.2, the features could be dropped according to large differences between the views. Thus, the camera should stay relatively stable in the beginning stage in order to successfully start to track features. Similarly, after feature tracking is completely lost, the recovery to resume tracking works in the same fashion as the starting state. The condition that the camera should not move much during the beginning or resuming period is easily satisfied in common user interaction situations. For example, people tend to stay focused on target objects when they start interactions with them, therefore naturally keeping the camera stable enough to begin tracking.

One limitation of tracking natural features in the environment is that the scene is considered to be stationary. In tabletop systems, however, users may move objects around while interacting with them. In such cases, the pose estimation may be less accurate when the features on these moved objects are taken into account for feature tracking (instead of being discarded by RANSAC). It may break down completely when the majority of surface texture is changed (e.g., by shuffling papers around). This problem can be addressed by adding new features constantly while interacting with the tabletop AR system. This is in contrast to our implementation, which simply expands the tracked space, adding new features more sparsely. If the object displacements are minor changes in the environment (e.g., removing a book), the major remaining features will still be sufficient to track the camera pose, and new features will be detected on the updated areas.

When the environment does not have sufficiently many corner points or textured regions, different types of features rather than just point features can be helpful. Detecting different features from edges, planes, or primitive objects can be added to our multithreaded framework as introducing new feature detection threads. Generalizing our approach to various feature detection and tracking methods is left for our future work.

6 Conclusions

We have introduced a hybrid approach for detecting distinctive image features and tracking them with optical flow computations in a multithreaded framework, providing a six-degree-of-freedom camera tracking system. Our markerless tracking is initialized by a simple hand gesture using the Handy AR system, which estimates a camera pose from a user's outstretched hand. The established coordinate system is propagated to the scene, and then the tabletop workspace for AR is continuously expanded to cover an area far beyond a single reference frame. The system uses distinctive image features in order to recognize the scene and to correct for accumulated tracking errors. Our proposed hybrid tracking approach proves to be useful for real-time camera pose estimation without using fiducial markers in a tabletop AR environment.

For future work, we want to tackle full 3D scene reconstruction. It is a challenging topic, which receives significant attention in the computer vision and robotics communities, with approaches such as structure from motion (SfM) or SLAM. The real-time constraint and user interaction required by AR systems make the problem even more interesting. Along the direction of hybrid feature tracking, using different types of features such as edges and partial models might prove to be beneficial to even more robust and reliable tracking. We also want to compare various feature detection algorithms for tracking in AR systems and to manage those heterogeneous features more efficiently in a unified framework.

Acknowledgments

This research was supported in part by the Korea Science and Engineering Foundation Grant (2005-215-D00316), by a research contract with the Korea Institute of Science and Technology (KIST) through the Tangible Space Initiative Project, and by NSF CAREER Grant IIS-0747520.

REFERENCES

[1] T. Lee and T. Höllerer, "Hybrid Feature Tracking and User Interaction for Markerless Augmented Reality," *Proc. IEEE Conf. Virtual Reality (VR '08),* pp. 145-152, 2008.

[2] H. Kato, M. Billinghurst, I. Poupyrev, K. Imamoto, and K. Tachibana, "Virtual Object Manipulation on a Table-Top AR Environment," *Proc. IEEE/ACM Int'l Symp. Augmented Reality (ISAR '00),* pp. 111-119, 2000.

[3] S. DiVerdi, D. Nurmi, and T. Höllerer, "ARWin—A Desktop Augmented Reality Window Manager," *Proc. IEEE/ACM Int'l Symp. Mixed and Augmented Reality (ISMAR '03),* pp. 298-299, 2003.

[4] T. Lee and T. Höllerer, "Handy AR: Markerless Inspection of Augmented Reality Objects Using Fingertip Tracking," *Proc. 11th IEEE Int'l Symp. Wearable Computers (ISWC '07),* pp. 83-90, 2007.

[5] T. Höllerer, J. Wither, and S. DiVerdi, *Anywhere Augmentation: Towards Mobile Augmented Reality in Unprepared Environments,* Springer Verlag, 2007.

[6] H. Kato and M. Billinghurst, "Marker Tracking and HMD Calibration for a Video-Based Augmented Reality Conferencing System," *Proc. Second IEEE and ACM Int'l Workshop Augmented Reality (IWAR '99),* pp. 85-94, 1999.

[7] M. Fiala, "ARTag, a Fiducial Marker System Using Digital Techniques," *Proc. Int'l Conf. Computer Vision and Pattern Recognition (CVPR '05),* pp. 590-596, 2005.

[8] P. Wellner, "Interacting with Paper on the Digitaldesk," *Comm. ACM,* vol. 36, no. 7, pp. 87-96, 1993.

[9] H. Ishii and B. Ullmer, "Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms," *Proc. ACM Conf. Human Factors in Computing Systems (CHI '97),* pp. 234-241, 1997.

[10] J. Rekimoto and M. Saitoh, "Augmented Surfaces: A Spatially Continuous Work Space for Hybrid Computing Environments," *Proc. ACM Conf. Human Factors in Computing Systems (CHI),* 1999.

[11] A. Butz, T. Höllerer, S. Feiner, B. MacIntyre, and C. Beshers, "Enveloping Users and Computers in a Collaborative 3D Augmented Reality," *Proc. Second IEEE and ACM Int'l Workshop Augmented Reality (IWAR '99),* pp. 35-44, 1999.

[12] M.L. Maher and M.J. Kim, "Studying Designers Using a Tabletop System for 3D Design with a Focus on the Impact on Spatial Cognition," *Proc. IEEE Int'l Workshop Horizontal Interactive Human-Computer Systems (TABLETOP '06),* pp. 105-112, 2006.

[13] P. Dietz and D. Leigh, "Diamondtouch: A Multi-User Touch Technology," *Proc. 14th ACM Symp. User Interface Software and Technology (UIST '01),* pp. 219-226, 2001.

[14] J.Y. Han, "Low-Cost Multi-Touch Sensing through Frustrated Total Internal Reflection," *Proc. 18th ACM Symp. User Interface Software and Technology (UIST '05),* pp. 115-118, 2005.

[15] G. Simon, A.W. Fitzgibbon, and A. Zisserman, "Markerless Tracking Using Planar Structures in the Scene," *Proc. IEEE/ACM Int'l Symp. Augmented Reality (ISAR '00),* pp. 120-128, 2000.

[16] V. Ferrari, T. Tuytelaars, and L.J.V. Gool, "Markerless Augmented Reality with a Real-Time Affine Region Tracker," *Proc. IEEE/ACM Int'l Symp. Augmented Reality (ISAR '01),* pp. 87-96, 2001.

[17] I. Skrypnyk and D.G. Lowe, "Scene Modelling, Recognition and Tracking with Invariant Image Features," *Proc. IEEE/ACM Int'l Symp. Mixed and Augmented Reality (ISMAR '04),* pp. 110-119, 2004.

[18] T. Lee and T. Höllerer, "Viewpoint Stabilization for Live Collaborative Video Augmentations," *Proc. IEEE/ACM Int'l Symp. Mixed and Augmented Reality (ISMAR '06),* pp. 241-242, 2006.

[19] A.J. Davison, I.D. Reid, N.D. 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.

[20] G. Klein and D. Murray, "Parallel Tracking and Mapping for Small AR Workspaces," *Proc. IEEE/ACM Int'l Symp. Mixed and Augmented Reality (ISMAR '07),* pp. 225-234, 2007.

[21] Intel, *OpenCV: Open Source Computer Vision Library Reference Manual,* 2000.

[22] D.G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *Int'l J. Computer Vision,* vol. 60, no. 2, pp. 91-110, 2004.

[23] H. Bay, T. Tuytelaars, and L.J.V. Gool, "SURF: Speeded Up Robust Features," *Proc. Ninth European Conf. Computer Vision (ECCV '06),* pp. 404-417, 2006.

[24] V. Lepetit and P. Fua, "Monocular Model-Based 3D Tracking of Rigid Objects," *Foundations and Trends in Computer Graphics and Vision,* vol. 1, no. 1, pp. 1-89, 2006.

[25] M.J. Jones and J.M. Rehg, "Statistical Color Models with Application to Skin Detection," *Proc. Int'l Conf. Computer Vision and Pattern Recognition (CVPR '99),* pp. 1274-1280, 1999.

[26] M. Kölsch and M. Turk, "Fast 2D Hand Tracking with Flocks of Features and Multi-Cue Integration," *Proc. IEEE Workshop Real-Time Vision for Human-Computer Interaction,* p. 158, 2004.

[27] Z.Y. Zhang, "A Flexible New Technique for Camera Calibration," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 22, no. 11, pp. 1330-1334, Nov. 2000.

[28] Y. Ma, S. Soatto, J. Kosecka, and S. Sastry, *An Invitation to 3D Vision, from Images to Models.* Springer Verlag, 2003.

[29] D. Nister and H. Stewenius, "Scalable Recognition with a Vocabulary Tree," *Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition (CVPR '06),* vol. 2, pp. 2161-2168, 2006.

[30] G. Schindler, M. Brown, and R. Szeliski, "City-Scale Location Recognition," *Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition (CVPR '07),* pp. 1-7, 2007.

[31] B. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," *Proc. Seventh Int'l Joint Conf. Artificial Intelligence (IJCAI '81),* pp. 674-679, 1981.

[32] J. Shi and C. Tomasi, "Good Features to Track," *Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition (CVPR '94),* pp. 593-600, 1994.

[33] C. Harris and M. Stephens, "A Combined Corner and Edge Detector," *Proc. Fourth Alvey Vision Conf. (AVC '88),* pp. 147-151, 1988.

[34] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision.* Cambridge Univ. Press, 2003.

[35] M.A. Fischler and R.C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Comm. ACM,* vol. 24, no. 6, pp. 381-395, 1981.

[36] R. Hess, *SIFT Detector,* http://web.engr.oregonstate.edu/~hess/, 2007.

[37] M. Pollefeys, R. Koch, and L.J.V. Gool, "Self-Calibration and Metric Reconstruction in Spite of Varying and Unknown Internal Camera Parameters," *Proc. Sixth IEEE Int'l Conf. Computer Vision (ICCV '98),* pp. 90-95, 1998.

[38] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," technical report, Univ. of North Carolina at Chapel Hill, 1995.

Taehee Lee received the BS degree in computer science from the Korea Advanced Institute of Science and Technology and the MS degree in computer science from the University of California at Santa Barbara. He is a PhD student in the Vision Laboratory, Computer Science Department, University of California, Los Angeles. He was a member of the Four Eyes Laboratory. His interests are in the areas of computer vision, structure from motion, and applications of vision-based techniques. He is a student member of the IEEE and the IEEE Computer Society.

Tobias Höllerer received the degree in computer science from the Technical University of Berlin and the PhD degree from Columbia University with a thesis on Mobile Augmented Reality Systems. He is an associate professor of computer science in the Department of Computer Science, University of California at Santa Barbara, where he codirects the "Four Eyes Laboratory," conducting research in the four "I"s of imaging, interaction, and innovative interfaces. He is a member of the IEEE and the IEEE Computer Society.