Virtual Reality (2014) 18:89–100
DOI 10.1007/s10055-013-0228-7

ORIGINAL ARTICLE

Dynamic learning, retrieval, and tracking to augment hundreds of photographs

Julien Pilet · Hideo Saito

Received: 19 November 2009 / Accepted: 15 November 2010 / Published online: 13 September 2013

Abstract Tracking is a major issue of virtual and augmented reality applications. Single object tracking on monocular video streams is fairly well understood. However, when it comes to multiple objects, existing methods lack scalability and can recognize only a limited number of objects. Thanks to recent progress in feature matching, state-of-the-art image retrieval techniques can deal with millions of images. However, these methods do not focus on real-time video processing and cannot track retrieved objects. In this paper, we present a method that combines the speed and accuracy of tracking with the scalability of image retrieval. At the heart of our approach is a bi-layer clustering process that allows our system to index and retrieve objects based on tracks of features, thereby effectively summarizing the information available on multiple video frames. Dynamic learning of new viewpoints as the camera moves naturally yields the kind of robustness and reliability expected from an augmented reality engine. As a result, our system is able to track in real-time multiple objects, recognized with low delay from a database of more than 300 entries. We released the source code of our system in a package called Polyora.

Keywords Augmented reality · Multiple object tracking · Image retrieval

J. Pilet (✉) · H. Saito
Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Kanagawa 223-8522, Japan
e-mail: julien.pilet@gmail.com

H. Saito
e-mail: saito@hvrl.ics.keio.ac.jp

1 Introduction

Object tracking is an important issue for many applications, especially in the domains of virtual, mixed, and augmented reality. The base process to visually integrate virtual elements on real ones is the following: a camera captures a scene. A registration technique provides the relative pose of the camera with respect to the target object. A standard CG method renders virtual contents from the appropriate point of view and integrates it on the camera image. Examples of such applications are numerous and include augmenting the pages of a book, playing cards, and trading cards. In these cases, as often, the camera is the only registration device available.

Beside adequate display and computing hardware, high quality augmented reality requires reliable and flexible tracking method. Many methods have been proposed for visual registration (Lepetit and Fua 2005). However, when it comes to tracking multiple natural objects, they all have drawbacks. A classical approach to track multiple objects is to detect a part common to all objects, typically a highly contrasted square, and use the registration to ease searching the characteristic area in the database. However, the necessity of marking target objects severely restricts the application domain of such tracking approaches. Recently, wide-baseline point matching has been demonstrated to effectively detect and track naturally textured objects (Ozuysal et al. 2007; Wagner et al. 2008). However, when multiple objects are to be recognized, these approaches try to match each known object in turn. Such a linear complexity, both in computation and memory requirements, limits the number of known targets to a few objects (Park et al. 2008).

Image retrieval approaches address the scalability issue. These methods can deal with millions of images (Nister

and Stewenius 2006). However, they focus on static images and are not designed for real-time tracking.

In this paper, we propose a real-time tracking method that integrates image retrieval techniques to achieve both accurate tracking and scalability with respect to the number of target objects. More specifically, we exploit information available in video stream, as well as the variability of feature descriptors, to efficiently establish correspondences with, index, and retrieve target objects. We also integrate dynamic learning of new viewpoints to increase detection robustness and accuracy. As a result, our method can reliably augment more than 300 objects individually, as depicted by Fig. 1.

To better describe the algorithmic contribution of our paper, let us sketch a typical image retrieval system. The first step is detection of features such as SIFT (Lowe 2004) or MSER (Matas et al. 2002), to summarize an image with a set of vectors. These vectors are quantized, turning features into *visual words*. Vector quantization involves clustering a large number of representative features, forming a *visual vocabulary*. Once an image is expressed as a *bag of word*, it can be indexed and searched for using information-retrieval techniques (Nister and Stewenius 2006; Sivic and Zisserman 2003). However, quantization errors combined with feature instability reduce retrieval performances. A key contribution of our work is a method that exploits quantization effects and the nonlinear variations of features to improve retrieval and to reach a real-time computation speed.

Our method observes the variability of descriptors over several frames, by tracking the keypoints from frame to frame. Inspired by Ozuysal et al. (2007), we capture the behavior of features during the training phase. The collected data allow our method to create a stabilized visual vocabulary. At runtime, our system matches full point tracks, as opposed to single features obtained from single frames, against training data. It yields a stable entry that can serve as index key. Because the set of detected features tends to vary with the viewpoint, we dynamically learn new views while tracking an object. As a result, detection of an object under a pose related to one that has previously been tracked is likely to succeed.

Fig. 1 To produce this result, our system detects, identifies, and tracks the photographs visible on the input video stream. Each picture is augmented with its corresponding virtual element, precisely registered. The stack of pictures visible on this image contains about 300 images. Our system can recognize all of them. Users can also augment new objects by simply showing them to the camera and clicking. Tracking and augmentation start right away

At a higher level, our contribution is a multiple natural object tracking system designed to scale well with the number of targets. Several of its properties make it perfectly suited for augmented reality applications: It can process live video stream, it can deal with a large number of target objects, adding new targets to the database is perceptually immediate, initialization is fully automatic, and it is robust to viewpoint changes, illumination effects, partial occlusion, and other typical tracking hazards.

We demonstrate the effectiveness of our approach with a toy application that can augment more than 300 pictures and that allows users to interactively augment with virtual elements a large number of new objects.

2 Related work

Pixel level registration techniques have today reached maturity (Lucas and Kanade 1981; Baker and Matthews 2004). Their association with point selection methods to concentrate computation efforts on interesting pixels forms the basis of many computer vision tasks such as object tracking (Harris and Stephens 1988; Shi and Tomasi 1994). Tracking requires initialization, which remained an issue a few years ago. Because marker-based approaches such as ARToolkit did not suffer from this drawback, they quickly became popular in the augmented reality community (Kato et al. 2000). Among the many improved versions proposed, ARTag, for example, can recognize 2002 different markers (Fiala 2005).

In parallel, feature point methods grew powerful enough to achieve wide-baseline matching. The most representative of them is probably Lowe's scale invariant feature transform (SIFT) (Lowe 2004). Establishing correspondences between views with strong differences in pose and illumination addresses many issues, including tracking initialization. Approaches such as SIFT aim at providing features robust to viewpoint changes, from a single image. Lepetit et al. (2004) proposed a real-time tracking by detection method that learns the changing appearance of keypoints from synthetically generated views. Ozuysal et al. (2006) rely on off-line incremental tracking on a training sequence to model the complex appearance

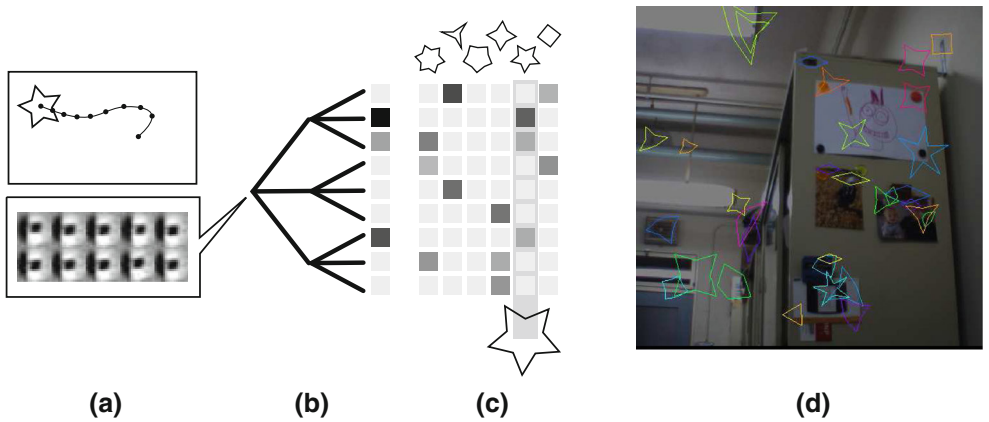

Fig. 2 Computing *visual words* by tracking features. **a** Features are tracked across frames, and patches are collected. **b** Patches pass through a tree and distribute over its leaves, forming a histogram. **c** The histogram is compared to the ones observed during the training phase. We use them as *visual words* and visualize them with *geometric shapes*. In this example, the histogram matches a word represented with a *star*. **d** Some of the *visual words* detected on an input image

changes of a partially transparent object. Taylor et al. (2009) consider a lighter descriptor and compensate its sensitivity to viewpoint changes by modeling a target object with a large number of features, extracted from an artificial set of views. Uchiyama and Saito (2009) use a descriptor specifically designed for text documents and learn both new keypoints and changing descriptors online. Our method also updates the models as new views are observed.

The success of wide-baseline feature matching also opened the way to large-scale image retrieval (Sivic and Zisserman 2003; Obdržálek and Matas 2005). Using vector quantization of descriptors, recent approaches effectively retrieve images from databases containing more than a million of images (Nister and Stewenius 2006; Philbin et al. 2007; Jégou et al. 2008). Interestingly, Philbin et al. (2008) note that quantizing several descriptions of the same physical point can be unstable. They tried to capture this instability by synthesizing affine and noisy deformations of image patches, inspired by a classification-based approach to wide-baseline matching (Lepetit et al. 2004). However, these attempts did not improve recognition performance, maybe because of an inappropriate choice of image perturbation, as the authors explain. Following a similar goal, our method observes descriptors tracked over several frames to mitigate the effect of their variability.

Feature matching has also been used to detect and track multiple 3D objects (Park et al. 2008). Recently, Wagner et al. proposed a method implemented on mobile phones that guarantees a minimum frame rate (Wagner et al. 2009). However, these methods are restricted to a limited number of objects, typically under 10. We focus on scaling the database size, while keeping a detection delay compatible with tracking tasks.

3 Method

The primitives upon which our method is built are feature detection and tracking. Keypoints[1] are detected at each frame, and matched between consecutive frames, using normalized cross-correlation (NCC). When such a simple NCC fails to find correspondences, the Lukas-Kanade (KLT) algorithm tracks lost features (Baker and Matthews 2004). As a result, we obtain stable tracks of features, and a patch is extracted at every frame, as illustrated by Fig. 2a.

During a first training phase, we collect many features from a video stream. We cluster their descriptors with a recursive *K*-mean tree, as suggested by Nister and Stewenius (2006) and as depicted by Fig. 2b. This allows us to summarize a descriptor as the leaf it corresponds to or as an integer, since leaves are numbered.

In a second training phase, we collect tracks of features. The tree quantizes each descriptor, turning the feature tracks into leaf tracks. We then compute for each track a histogram of leaves. Because the training sequence contains effects such as motion blur, perspective distortion, or moving specular reflections, the collected histograms capture the descriptor's instability, including the quantization effects of the tree, in presence of such hazard. The set of collected histograms forms a dictionary, or a *visual vocabulary*. To reduce its ambiguity, similar histograms are recursively merged until ambiguity reaches an acceptable level.

At run-time, the whole history of each tracked feature is summarized by the most relevant histogram found in the dictionary. Each descriptor passes through the tree and ends up in a leaf, forming a histogram of leaves, as

[1] In this text, we define a *keypoint* as a location of interest on an image, a *descriptor* as a vector describing a keypoint neighborhood, and a *feature* as both a keypoint and its descriptor.

depicted by Fig. 2b. We then simply search for the most similar trained histogram (Fig. 2c). The advantage of this technique is double: It allows the algorithm to exploit both the variability of descriptors, which is usually viewed as a problem, and the large amount of data collected by tracking points over multiple frames. Figure 2d shows an image and some of its detected features, with their associated histograms, represented as geometric shapes.

Indexing and retrieval follow a standard term frequency-inverse document frequency (TF-IDF) weighting scheme on a bag of word model. In simpler words, a table maps dictionary entries to indexed objects. For retrieval, the table is looked up to collect all the objects on which the visual words of the query appear. The candidates are then ranked using an appropriate criterion.

The few best results are selected as candidates for geometric verification. If an object is successfully detected, matches are propagated to the next frame using motion flow estimation. Currently, tracked objects are automatically appended to the list of candidates for next frame's geometric verification stage. Because we eliminate outliers during detection and propagate matches, few outliers remain on the next frame. Geometric verification therefore becomes simpler.

The successfully verified objects then pass the dynamic learning stage in which observed but unmatched features are integrated to the object model. Our algorithm takes care of not integrating features that do not actually belong to the object, and stops adding keypoints when enough are known. As a result, keypoints appearing on new views contribute to tracking and detection, making them robust and reliable.

The remaining of this section details our system's components.

3.1 Low-level: repeatable sparse motion flow

Our method relies on a particular type of motion flow estimation in which the tracked points are detected in a repeatable way. It means that a point that has previously been tracked is supposed to be tracked again when viewed from another angle. Stability and repeatability are the main goals of well-known feature detectors such as SIFT, SURF, or FAST (Lowe 2004; Bay et al. 2006; Rosten and Drummond 2006). In our experiments, we used a GPU implementation of SIFT (Wu 2008).

To compute the motion flow from frame to frame, we first detect the keypoints in both frames. We then compare their local neighborhoods using NCC. This quickly provides the flow of most features. However, when the feature detector detects a point on the first frame but fails to detect it on the following one, the KLT algorithm searches its new location. The decision to turn to KLT is taken when the best found correlation drops below a threshold.

The result of this process is a set of stable tracks. The KLT tracking mitigates the feature detector's failures, while the feature detector ensures repeatability and prevents the KLT tracker from drifting. In our experiments, this approach can track efficiently hundreds of points over hundreds of frames.

The choice of using NCC rather than SIFT descriptor is motivated because the invariance provided by the SIFT is not necessary for computing sparse motion flow. NCC gives reliable result and is consistent with the KLT tracker. The KLT tracker is well adapted to track SIFT features, because a local maximum in the difference of Gaussian implies low autocorrelation in all directions, which is precisely what KLT relies on.

3.2 Describing tracked keypoints

At this stage, our goal is to index target objects and to retrieve the ones visible on the current frame, relying on stable feature tracks. To do so, we aim at constructing a dictionary mapping feature tracks to indexed objects. Our approach has two stages of clustering: at descriptor level and at track level.

3.3 *K*-mean tree

The descriptors extracted during the training sequence are clustered using a recursive *K*-mean, as proposed by Nister and Stewenius (2006). All descriptors are first clustered by a *K*-mean algorithm, with $K = 4$, and using the L^2 norm as a distance measure. The four resulting subsets are then recursively clustered in the same way, in turn. Recursion stops either at depth 8 or when fewer than 64 descriptors remain.

3.4 Learning and exploiting feature variability

Our system tracks points and assigns them to a leaf of the recursive *K*-mean tree, making it possible to observe groups of leaves showing the same physical point. During the second training stage, such groups are collected from a video sequence and accumulated into histograms. Such histograms can capture the descriptor's nonlinear variability that might be caused by viewpoint, illumination, or noise effects.

Straightforward usage of these histograms as index keys is not possible due to the many redundant elements collected during training. We address this ambiguity issue with a second clustering stage. If two histograms are too

difficult to discriminate, they are merged, creating a new, larger histogram. In practice, we compute the dot product between two normalized histograms and use agglomerative clustering. We recursively merge the histogram pair with the highest normalized dot product and stop when it is lower than a threshold (typically 0.1, see Sect. 4). Merging two histograms a and b into a new histogram x gives: $x(l) = a(l) + b(l)$, where $a(l)$ is the number of times leaf l appears in a. At the end of this process, the remaining histograms form a stable visual vocabulary.

Although these histograms seem very large, due to the large number of leafs on the tree, most entries are null. Storing them in a sparse manner is quite compact.

3.5 Feature tracks as index keys

We build an inverted table mapping leaves of the recursive K-mean tree to trained histograms. Given one or several leaves observed when tracking a point, it becomes possible to efficiently fetch the most similar learned histogram. By doing so, our system assigns an index key to a *point track*, as opposed to a single feature. The following equation defines the score of the histogram h for the track t:

$$s(t,h) = \frac{1}{\sum_l t(l)} \frac{1}{\sum_l h(l)} \sum_l t(l)h(l)\mathrm{idf}(l), \qquad (1)$$

where $t(l)$ (respectively $h(l)$) is the number of features assigned to leaf l in the track t (respectively in the trained histogram h). The term $\mathrm{idf}(l) = -\log(\frac{f(l)}{F})$, where $f(l)$ denotes the number of trained histograms involving leaf l, and F the total number of trained histograms. This score gives more importance to rare and discriminative leaves and decreases the weight of frequent ones. If $f(l) = 0$, it means that the observed feature does not match trained data. In this case, the track is ignored.

The complexity of directly computing this score grows linearly with the track size. Therefore, we remember for each track the scores of potential matching histograms. When a new frame is available, the scores are updated regarding only the newly observed leaf. This incremental approach allows our system to efficiently exploit long tracks.

Soft visual word assignment, as suggested by Philbin et al. (2008), can easily be achieved by considering not only the histogram with the highest score, but also the ones at least 90 % as good.

3.6 Object detection

To detect target objects entering the field of view, the database is queried with all point tracks visible on the current frame. As explained, each point track is assigned to a visual word. The histogram of the observed *visual words* in a frame forms a query q. The score of stored object d for the query q is as follows:

$$s(q,d) = \frac{1}{\sum_w q(w)} \frac{1}{\sum_w d(w)} \sum_w q(w)d(w)\mathrm{idf}(w), \qquad (2)$$

where $q(w)$ (respectively $d(w)$) is the number of words w in q (respectively in d), and $\mathrm{idf}(w)$ the negative log of the proportion of the stored frames that contain the visual word w. The few objects with the best scores are kept as candidates for geometric verification.

From a computational point of view, we reduced the complexity of repeating this algorithm at each frame using incremental queries. We keep the scores of objects found in the previous frame and update them with features that appeared or disappeared. The complexity of the query therefore depends on the number of feature addition or subtraction rather than the total number of features present on the current view.

3.7 Geometric verification

Our algorithm builds a list of candidates for geometric verification. It is initialized with the set of objects successfully tracked on the previous frame. Then, the list is extended with at most three other candidates selected by their query score (Eq. 2).

For each object in the list, our system tries to match object and frame features in two different ways: based on the tracks index values (Sect. 3.5), or propagated from previous frame if the candidate has been detected successfully. In the latter case, propagated correspondences contain usually less outliers than the ones resulting from wide-baseline matching. Because the ratio of outliers has an impact on the required number of RANSAC iterations and computation speed, our system uses for geometric verification at most 20 % of new correspondences. In practice, our system first gathers the correspondences from last frame and then finds at most 20 % more matches with the candidate target. When verifying an object that has not been matched on previous frame, only the noisier wide-baseline correspondences are used, requiring more RANSAC iterations.

Once the set of potential correspondences is created, the geometry consistency is verified. Each object has a geometric model, in our implementation either homography or epipolar constraints. For detection, the RANSAC (Fischler and Bolles 1981) algorithm handles the potentially high outlier rate. During tracking, the outlier rate is controlled, and the least median of squares (LMedS) algorithm can optionally replace RANSAC.

3.8 Dynamic model update

Once an object is detected, it can be tracked on views that may strongly differ from the reference one recorded in the database. In that case, some keypoints disappear and others appear. In that case, our system incrementally improves the model by adding appearing features. For this task, the difficulty is to determine whether a feature actually belongs to the object or not.

Our algorithm first eliminates geometrically inconsistent keypoints. It scans all the features matched over more than a fixed amount of frames (typically 5). It also skips keypoints assigned by the RANSAC processes to an object. To determine to which object a keypoint could be attached, the nearest keypoint already assigned to an object in the current frame is considered. If its distance to the candidate keypoint exceeds a threshold (typically 30 pixels), the point is rejected: It probably belongs to the background. Otherwise, the geometric consistency between the corresponding object and the considered keypoint is verified over all the previous frame in which the point has been tracked. To do so amounts to verifying that the inverse transformation of the keypoint by the homography computed for the object remains consistent over the frames. In our implementation, we check that the point on previous frames, once backprojected, falls within 3 pixels of current frame's backprojection.

To prevent the number of keypoints associated to objects from growing too large, we limit it to a constant (typically 3000). Before adding a new keypoint to an object, the system draws a random number between 0 and the limit. If that number is below the current point count, the keypoint is skipped. This approach gives a logarithmic behavior to the system: The rate of feature addition is high if the object is not yet well modeled and slows down as more and more features are integrated. Other strategies to do so could consider the viewpoint to encourage a uniform distribution of learned features over viewpoints or avoiding adding keypoints in regions already well covered. We consider that the optimal strategy depends on the final application. In our case, this simple method is sufficient.

Dynamically, updating the database breaks the incremental computation of the objects scores described in Sect. 3.6. Score computation starts from scratch everytime the database is updated. The impact of this effect is minor compared to the benefit of remembering new views for later detection.

4 Results

We present in this section the experiments we conducted to evaluate our system. We focused on retrieval and tracking capabilities. We finally present a toy application demonstrating that our system fits augmented reality requirements.

4.1 Retrieval evaluation

The goal of this experiment is to evaluate our system's ability to retrieve objects visible in a video stream. To do so, we ignore the geometric verification stage and concentrate on the best ranked candidate returned by the query.

We differentiate two scenarios. If the time to enter the objects in the database is not an issue, it is possible to include their views in the two clustering stages. This yields good performance, at the cost of a longer and less flexible procedure to add the objects to the database. Because some applications cannot tolerate such a long process, we tested our method with and without including the target objects in the learning data. We call the tested scenarios trained and untrained.

Evaluation was conducted on video sequences showing 13 test objects, three of which are depicted by Fig. 3. The image resolution used is 640×480. The first sequence does not show the test objects. It is used to build the visual vocabulary. The second sequence shows the objects in turn. Thirteen objects are manually added to the database. The third sequence is a test sequence. It also shows the objects, at most one at a time. Ground truth reference is obtained by manually noting the object visible on each frame. To test a scenario, we use every frame of the test sequence as a query to the database. We count the number of frames in which the first result matches the ground truth.

In the trained scenario, the visual vocabulary is built using both the first and the second sequences. It implies that the resulting K-mean tree is more complete. The untrained scenario does not use the second sequence for building the visual vocabulary. In this case, the system has to deal with previously unseen features.

Each scenario is tested twice. Once using directly the leaves of the K-mean tree as index keys, and once using our approach.

Table 1 presents the evaluation results. In both scenarios, our method improves retrieval. The results obtained in the trained scenario show that even when quantization errors are avoided by using the target objects to create the visual vocabulary, our method can still learn some remaining variability and provide a performance gain of about 4.5 %.

4.2 Tracking objects

For this experiment, we printed 325 photographs. We entered the pictures one by one into the system, by capturing them on a uniform background with a handheld

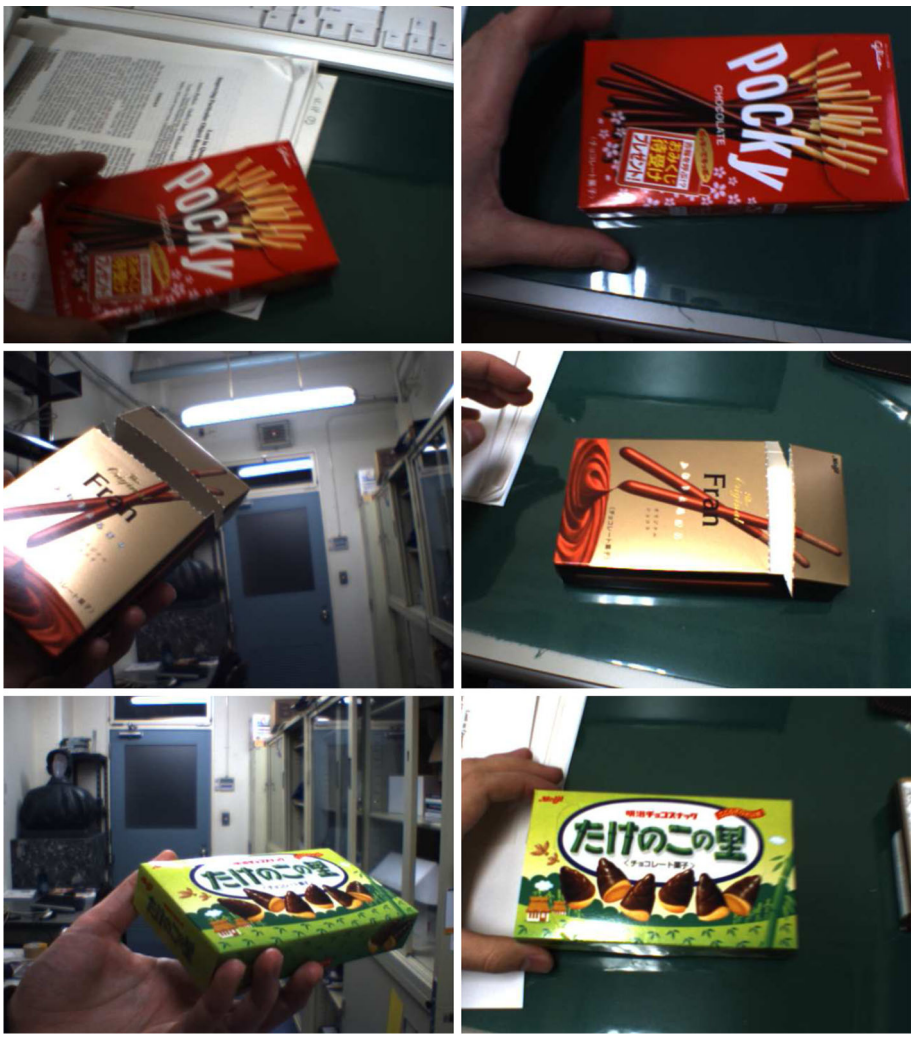

Fig. 3 Pairs of query (*left*) and retrieved (*right*) images that our approach made possible to retrieve, without any geometric verification. The direct tree indexing approach failed to handle these frames, as opposed to our method. In the untrained scenario, our method improves the number of successfully retrieved frames of about 5 %, as detailed by Sect. 4.1 and Table 1. These three pairs are selected among these 5 %. In the case of the first row, motion blur causes SIFT detector instability. In the second case, specular reflections alter the object appearance. In the third case, the perspective distortion and the moving specular reflection perturb retrieval if no tracking information is used

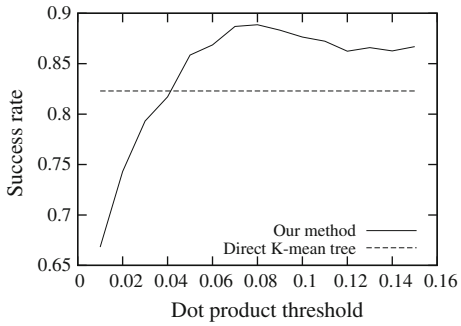

Fig. 4 Changing the threshold that stops histogram merging during the second clustering phase. If the threshold is chosen above 0.06, our method improves retrieval rate by 4–6 %. This graph was obtained in the untrained scenario, as described in Sect. 4.1

video camera. It is important to note that adding a new picture to the system is perceptually instantaneous, because the *K*-mean tree and histogram clusters remain constant. The user points the target to the camera, clicks, and tracking can start.

Table 1 To evaluate performance, we count the number of frames correctly retrieved for a test sequence.

	Success/direct	Success/stabilized
Untrained	2,240	2,351 (+5.0 %)
Trained	2,389	2,497 (+4.5 %)

The rows correspond to the scenarios described in Sect. 4.1. The column Success/direct contains results obtained without using our method. The column Success/stabilized presents the results when using our approach. This table clearly shows that, for both scenarios, our approach improves performance

Based on the populated database, the system is able to recognize and track randomly chosen pictures. The recognition delay is short, typically 2–3 frames. Once detected, the photographs are tracked, subject to neither drift nor jittering, as depicted by Fig. 5.

During the training stage, we transformed 125 pictures in their digital form with random homographies to generate synthetic views, out of which about 5 million features were extracted. We recursively applied *K*-mean clustering with

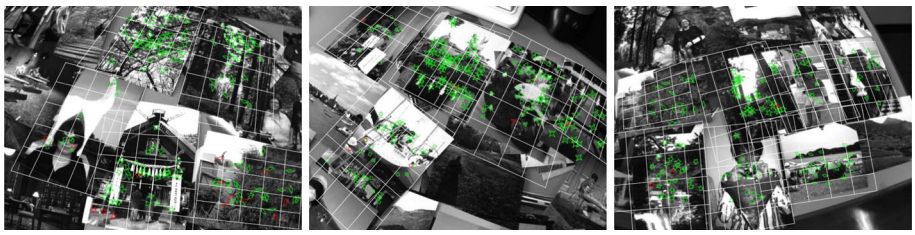

Fig. 5 Tracking photographs on a desk. The frames are selected from a test sequence in which the user moves a camera above a desk on which lies several tens of photographs. The system recognizes and track the pictures when they enter the field of view. The *white grid* represents the detected homography. The *red marks* show the newly established correspondences and the *green marks* the ones propagated from previous frame

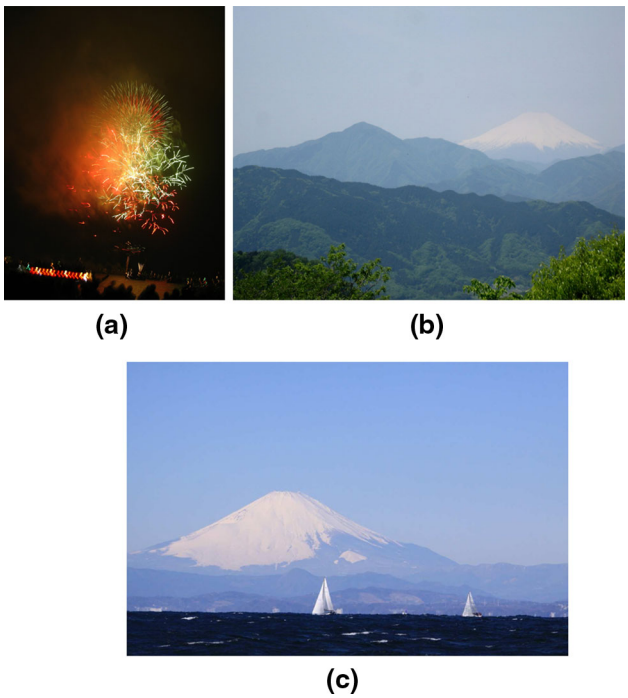

Fig. 6 Cases of failure. The system fails to handle these images, due to the specific nature of the texture (**a**), low contrast due to atmospheric effects (**b** and **c**), and poorly discriminative texture (**b** and **c**). In total, only 10 pictures out of the 325 ones cannot be detected effectively. Half of them are not recognized at all, while the three others are detected only on unoccluded, sharp, and frontal views

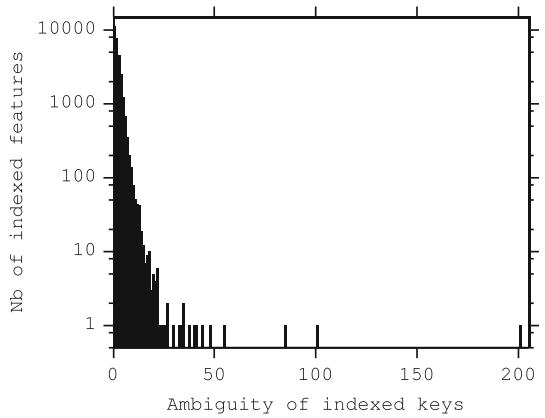

Fig. 7 This *histogram* shows the ambiguity of the index dictionary. The *horizontal axis* shows the number of occurrences. The *vertical one* shows the number of index keys. For example, 11213 features have their own index key. The most ambiguous feature appears at 200 places. This figure clearly shows that the indexing scheme is discriminative because most features are assigned to a unique index key, and there is only a small number of ambiguous features that appear often

$K = 4$, stopping at a maximum depth of 8 or when less than 32 descriptors remained in a branch. The resulting tree has 85,434 nodes, 63,955 of which are leafs. During the second training phase, 655,970 histograms were extracted from new synthesized views. The agglomerative clustering process produced 39,749 histograms.

The tree and cluster set produced during training allow our system to efficiently establish correspondences between an input view and the objects stored in the database. It is interesting to observe that the system can deal with objects that have not been used for training. We verified this behavior by populating the database with 200 unseen objects, in addition to the 125 ones used for training. Our system kept its performance and could successfully detect and track almost all of the 325 targets, except a few.

Out of 325 pictures, the system fails to detect only 10: 6 among the learned 125 image set and 4 among the 200 other ones. A few of these pathological pictures are illustrated by Fig. 6.

When several pictures look similar, the system might be confused. For example, we took two pictures of the same building, taken from a slightly different points of view. In this case, the system tends to recognize the building rather than the pictures themselves. At detection, it sometimes mixes up both pictures. The tracking is stable in the sense that, once the system selected one picture, it will stick to that choice as long as the target can be tracked. This behavior is acceptable for augmented reality applications.

Figure 7 depicts the ambiguity of keypoints. The histogram shows that, within our database, most of the keypoints have a quantized appearance that occurs only a few times. Therefore, they provide a strong support to the

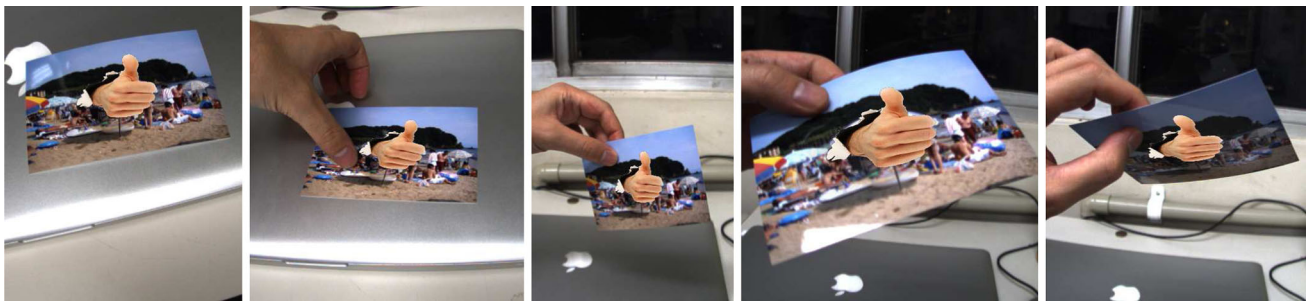

Fig. 8 In these frames selected from a longer sequence, our system automatically augments the picture with a virtual hole through which a hand passes. The virtual hand appears stable on the moving photograph, despite illumination changes, partial occlusion, camera defocus, and specular reflection

retrieval process. However, Fig. 7 also shows that a few descriptors appear very often. These points are less discriminative but can still bring information for registration.

4.3 Dynamic update evaluation

Dynamically inserting new keypoints in the database as they are discovered under new viewpoints allows the system to match more features while detecting the target in a similar viewpoint. To evaluate our system's robustness to viewpoint changes, and to quantify the improvement given by the dynamic update, we conducted the following experiment. We first randomly picked a picture that we mounted on the rotating support depicted by Fig. 9. We then measured the number of matches validated by the RANSAC process in four different cases:

1. Tracking, prior to learning;
2. Detection, prior to learning;
3. Tracking with dynamic learning enabled;
4. Detection, after learning.

In the case of tracking, we slowly rotate the support starting from 0° to 90°. Every 15°, we count the number of correct matches. In the case of detection, we orient the support, completely occlude the object to break tracking, show the object again, and inspect the matches. Tracking with dynamic learning enabled enriches the database with the keypoints visible on new views. Repeating the detection experiment after that learning process yields much better result because the model to match with is more complete.

Figure 10 shows the result of this experiment conducted on four different target pictures. When no dynamic learning is used, the tracking and detection curves start from the same point. Tracking is then more performant, as expected. The number of matches for 0° varies from one picture to the other due to their different appearance. Picture #3 is clearly the most textured one.

While tracking and learning, many points are added to the model, yielding a high number of matches. However, some of these matches might not be reliable. The keypoint could have been detected outside of the object, or it might have been created by a specular reflection. The detection after learning is more relevant, because only the features that can actually be matched are counted. The number of matches after learning is consistently and significantly higher than before learning. By automatically completing the target representation, dynamic learning allows the system to recognize the target in more difficult situation. In the case of picture 1, straight detection fails as early as 30°. After updating the model, detection is still reliable at 60°.

4.4 Application to augmented reality

To demonstrate our system's suitability to AR applications, we used it to augment pictures with virtual drawings. The database of target objects contains the 325 photographs

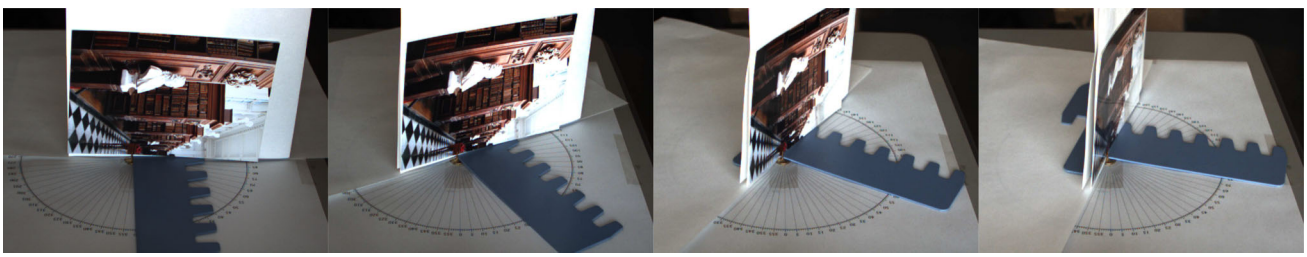

Fig. 9 Some of the images used for the experiment described in Sect. 4.3. The target is attached to a rotating support. The rotation angles are, from *left* to *right*, 0°, 30°, 60°, and 75°. These angles are represented on the horizontal axes of Fig.10

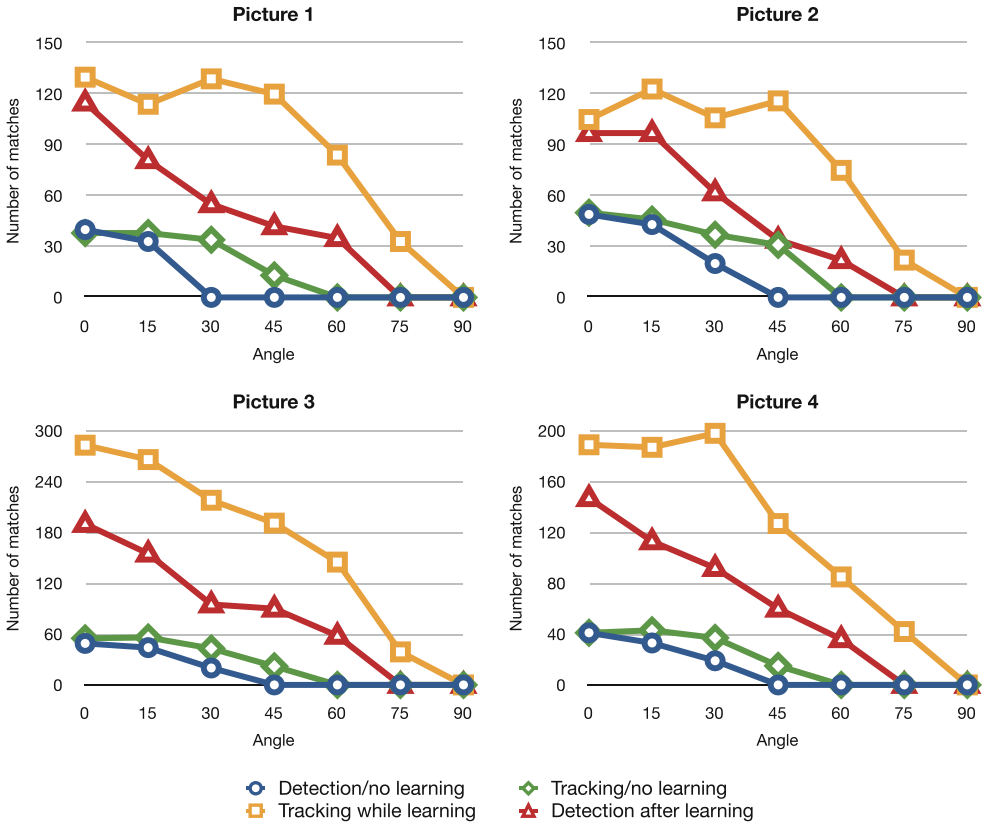

Fig. 10 These *charts* show the number of matches as a function of rotation angle, as explained in Sect. 4.3. The *four charts* correspond to *four different pictures*, the last of which is visible on Fig. 9. These *four cases* show that the dynamic learning process significantly improves the system's detection abilities

mentioned in the previous section. When the system recognizes a known picture, it overlays it with its virtual counterpart warped with the appropriate homography.

As depicted by Fig. 8, our method's stable tracking yields convincing augmented reality, despite hazards such as viewpoint changes, illumination changes, partial occlusion, camera defocus, and specular reflection. The frames of Figs. 8 and 11 were produced directly and in real time by our system fed with a live video stream (except cropping).

As illustrated by Fig. 11, when several photographs appear in the field of view, our system augments them as long as they appear large enough on the input image. Since an homography is computed for each target picture, the system can augment them even if they move independently.

The frame rate of our system is typically 6–8 frames per second. The computationally heaviest component in our implementation is the SIFT feature detector, despite its implementation on the GPU.

5 Conclusion

In this paper, we presented an image retrieval approach to multiple object tracking. We demonstrated its effectiveness and scalability by running experiments on more than 300 target objects. Our system is user friendly because it is responsive, fully automated, and reliable. For example, augmenting a new object simply amounts to pointing the camera at it and clicking. The augmentation starts immediately, and the internal object representation is automatically improved as new views become available. Detection for tracking initialization is 100 % automatic and has a low delay. Our system can process live video streams and is robust to partial occlusion, viewpoint changes, illumination effects, and other hazards.

These properties make our approach ideal for augmented reality applications that overlay virtual elements on real objects. Possible applications include:

- animating pages of books,
- tagging virtual messages on real walls,
- virtually annotating objects for maintenance tasks,
- augmenting card games with virtual scenes,
- augmenting panorama views with landmark names.

To encourage further development of such AR applications, we provide the source code of our system in a packaged named Polyora.[2]

We demonstrated the validity of our approach with planar objects. We believe it could be quite adapted to handle more complex shapes, because of its multi-stage

[2] https://github.com/jpilet/polyora.

Fig. 11 Augmenting multiple objects simultaneously. Our system retrieves, tracks, and augments the pictures lying on the desk. On the *first row*, all the objects are augmented with the same virtual content. On the *second row*, the virtual elements vary from object to object. In these examples, 3–6 pictures are tracked and augmented simultaneously

learning process, which could handle the varying appearance of features lying on nonplanar objects. In future work, we plan to replace the homography by a full 3-D transformation to extend the proposed system to any type of textured object. We also aim to reduce our system's dependency on texture. Currently, only very textured objects can be detected easily. Taking into account the geometric relationship of keypoints could extend indexing and retrieval to printed text or uniform objects with a specific 3-D shape, such as a chair or a tripod.

References

Baker S, Matthews I (2004) Lucas-kanade 20 years on: a unifying framework. Int J Comp Vis 56(3):221–255

Bay H, Tuytelaars T, Gool LV (2006) SURF: speeded up robust features. In: European conference on computer vision

Fiala M (2005) ARTag, a fiducial marker system using digital techniques. In: Conference on computer vision and pattern recognition, pp 590–596

Fischler M, Bolles R (1981) Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun ACM 24(6):381–395

Harris C, Stephens M (1988) A combined corner and edge detector. In: Fourth alvey vision conference, Manchester

Jégou H, Douze M, Schmid C (2008) Hamming embedding and weak geometric consistency for large scale image search. In: European conference on computer vision, LNCS, vol 1, pp 304–317

Kato H, Billinghurst M, Poupyrev I, Imamoto K, Tachibana K (2000) Virtual object manipulation on a table-top AR environment. In: International symposium on augmented reality, pp 111–119

Lepetit V, Fua P (2005) Monocular model-based 3d tracking of rigid objects: a survey. Found Trends Comp Graph Vis 1(1):1–89

Lepetit V, Pilet J, Fua P (2004) Point matching as a classification problem for fast and robust object pose estimation. In: Conference on computer vision and pattern recognition, Washington, DC

Lowe D (2004) Distinctive image features from scale-invariant keypoints. Int J Comp Vis 20(2):91–110

Lucas B, Kanade T (1981) An Iterative Image Registration Technique with an Application to Stereo Vision. In: International joint conference on artificial intelligence, pp 674–679

Matas J, Chum O, Martin U, Pajdla T (2002) Robust wide baseline stereo from maximally stable extremal regions. In: British machine vision conference, London, pp 384–393

Nister D, Stewenius H (2006) Scalable recognition with a vocabulary tree. In: Conference on computer vision and pattern recognition

Obdržálek Š, Matas J (2005) Sub-linear indexing for large scale object recognition. In: British machine vision conference

Ozuysal M, Lepetit V, Fleuret F, Fua P (2006) Feature harvesting for tracking-by-detection. In: European conference on computer vision, Graz

Ozuysal M, Fua P, Lepetit V (2007) Fast keypoint recognition in ten lines of code. In: Conference on computer vision and pattern recognition, Minneapolis, MI

Park Y, Lepetit V, Woo W (2008) Multiple 3d object tracking for augmented reality. In: International symposium on mixed and augmented reality, pp 117–120

Philbin J, Chum O, Isard M, Sivic J, Zisserman A (2007) Object retrieval with large vocabularies and fast spatial matching. In: Conference on computer vision and pattern recognition

Philbin J, Chum O, Isard M, Sivic J, Zisserman A (2008) Lost in quantization: Improving particular object retrieval in large scale image databases. In: Conference on computer vision and pattern recognition

Rosten E, Drummond T (2006) Machine learning for high-speed corner detection. In: European conference on computer vision

Shi J, Tomasi C (1994) Good features to track. In: Conference on computer vision and pattern recognition, Seattle

Sivic J, Zisserman A (2003) Video google: a text retrieval approach to object matching in videos. In: Proceedings of the international conference on computer vision, vol 2, pp 1470–1477

Taylor S, Rosten E, Drummond T (2009) Robust feature matching in 2.3μs. In: IEEE CVPR workshop on feature detectors and descriptors: the state of the art and beyond

Uchiyama H, Saito H (2009) Augmenting text document by on-line learning of local arrangement of keypoints. In: International symposium on mixed and augmented reality, pp 95–98

Wagner D, Reitmayr G, Mulloni A, Drummond T, Schmalstieg D (2008) Pose tracking from natural features on mobile phones. In: International symposium on mixed and augmented reality, Cambridge

Wagner D, Schmalstieg D, Bischof H (2009) Multiple target detection and tracking with guaranteed framerates on mobile phones. In: International symposium on mixed and augmented reality, Orlando

Wu C (2008) A GPU implementation of David Lowe's scale invariant feature transform