Open access • Proceedings Article • DOI:10.1109/CVPRW.2010.5543248

Server-side object recognition and client-side object tracking for mobile augmented reality — Source link

Stephan Gammeter, Alexander Gassmann, Lukas Bossard, Till Quack ...+1 more authors

Institutions: ETH Zurich

Published on: 13 Jun 2010 - Computer Vision and Pattern Recognition

Topics: Video tracking, Augmented reality, Cognitive neuroscience of visual object recognition, Image retrieval and Mobile device

Server-side object recognition and client-side object tracking for mobile augmented reality

Stephan Gammeter[1] Alexander Gassmann[1] Lukas Bossard[1]
gammeter@vision.ee.ethz.ch agassman@ee.ethz.ch bossard@vision.ethz.ch
Till Quack[1,2] Luc Van Gool[1,3]
quack@kooaba.com vangool@esat.kuleuven.be
[1]ETH Zurich [2]kooaba AG [3]KU Leuven
Zurich, Switzerland Zurich, Switzerland Leuven, Belgium

Abstract

In this paper we present a system for mobile augmented reality (AR) based on visual recognition. We split the tasks of recognizing an object and tracking it on the user's screen into a server-side and a client-side task, respectively. The capabilities of this hybrid client-server approach are demonstrated with a prototype application on the Android platform, which is able to augment both stationary (landmarks) and non stationary (media covers) objects. The database on the server side consists of hundreds of thousands of landmarks, which is crawled using a state of the art mining method for community photo collections. In addition to the landmark images, we also integrate a database of media covers with millions of items. Retrieval from these databases is done using vocabularies of local visual features. In order to fulfill the real-time constraints for AR applications, we introduce a method to speed-up geometric verification of feature matches. The client-side tracking of recognized objects builds on a multi-modal combination of visual features and sensor measurements. Here, we also introduce a motion estimation method, which is more efficient and precise than similar approaches. To the best of our knowledge this is the first system, which demonstrates a complete pipeline for augmented reality on mobile devices with visual object recognition scaled to millions of objects combined with real-time object tracking.

1. Introduction

In recent years, research in Augmented Reality (AR) has made significant progress towards real world consumer applications. One of the main drivers for this development have been the increased processing capabilities and multimedia features of mobile phones. The most advanced

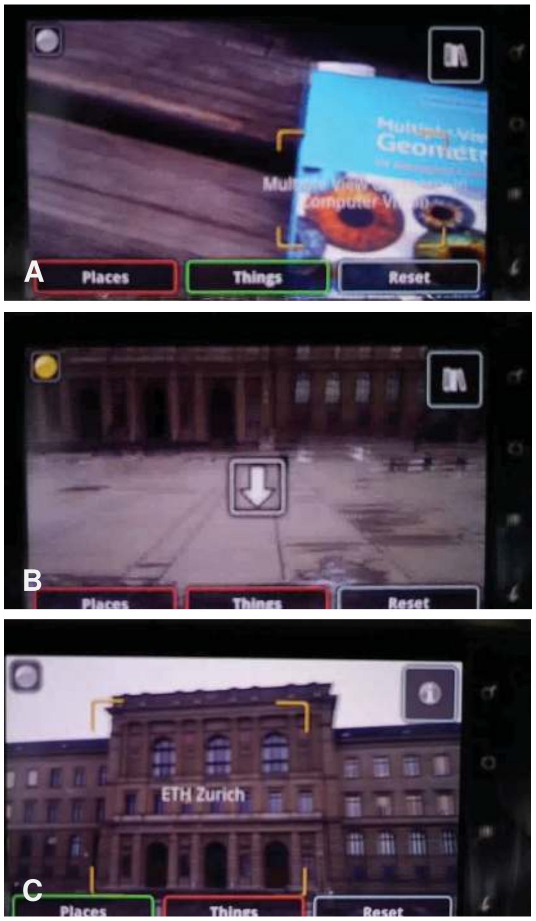

Figure 1. Screenshot of our mobile augmented reality application running on a Google Nexus One phone. The system is able to recognize, track and augment non-stationary objects such as books (A), memorize their position even when out of screen using visual and sensor based cues (B), and seamlessly also recognizes and tracks stationary objects such as landmark buildings (C).

class of devices (so called smartphones) nowadays come equipped with high-resolution touch screens, cameras, accelerometers, GPS, compass, *etc*. Such devices make ideal platforms for consumer-oriented AR applications (video see-through), since they are widely available, cheaper and also more discreet than, say, wearing head mounted displays (HMD, optical see-through).

Most of the AR applications for phones have focussed on the visual modality, overlaying virtual objects on the real world seen through the live camera feed on the mobile phone's screen. The basic concept consists of identifying real world objects on the screen, tracking them, and then augmenting the scene with artificial objects. Tracking is often combined with some estimation of the correct 2D or 3D world coordinates for proper placement of augmentations in the scene. Applications can then be further discriminated into those, which build on artificial markers in order to identify objects [22, 9, 28], or those, which use "natural" image features [10, 16, 17, 25]. Both kinds of application have been demonstrated to work in real-time on PC's for quite a while [10, 25], but have gone through a renaissance with their adaption to mobile computing platforms in recent years [16, 17, 5]. Here, especially the discovery of more robust and efficient local features [6, 18, 23] has fueled the advance of markerless approaches.

In a parallel development, the computer vision community has made astonishing progress in visual object recognition, both in terms of precision and scalability [7, 13, 19, 20, 24]. State of the art methods allow for retrieving specific objects (such as products, buildings, *etc*.) from databases with millions of items in a matter of seconds. In fact, much of this advance can also be attributed to the same improved local features, which are responsible for the improved tracking performance on the client.

Combining those two strands of work has the potential for very exciting AR applications. To that end, in this paper we propose a hybrid approach, which delegates the object recognition task to a server, and carries out tracking on the client-side, *i.e*. on the mobile phone. This has several advantages. Firstly, it allows to retrieve objects from very large databases in near real-time, as opposed to keeping the database on the client. Second, it still allows for interactive usage on the client, as opposed to sending only single images with the click of a button. Third, while being interactive, the approach also limits the amount of communication with the server, as opposed to the extreme case of transmitting live video to the server. This is particularly important considering today's mobile data transmission costs, especially when it comes to roaming charges due to the usage of data intensive applications abroad.

Furthermore, in comparison to AR approaches that strongly rely on GPS and sensor input like [1, 15, 3, 27], basing a system on visual recognition has the advantage, that both stationary objects (such as landmark buildings) and non-stationary objects (such as products, billboards, print media *etc*.) can be augmented seamlessly. The advantage over marker-based approaches is also quite obvious, as no markers have to be placed, which allows widespread application. On the other hand it is evident, that the usefulness of such a system is constrained by the size of the object database on the server side. To cope with this challenge, we build on recent research in mining data from community photo collections [8, 11, 12, 21] for landmarks and a commercial API for product recognition [2].

With all these components in place, the research we present in this paper relates to several recent lines of work in both commercial and scientific context. Overall, taking into account the server-side modules our system is built of, Google Goggles [4] is probably the product that is most similar to our system. However, to this date, the client-side of Google Goggles does not allow "live" tracking and augmenting of objects, *i.e*. recognition is triggered with a manual shutter release. Adding tracking adds the important option to incorporate gesture like Human-Computer Interaction, *e.g*. to indicate regions of particular interest. Also, in terms of server-side recognition, besides Goggles our system must be among the currently largest deployments, using roughly 12 million images for landmark recognition (following the approach of [11]) and about 10 million media covers using the public API of kooaba [2].

In terms of client-side tracking this paper relates to several recent advances reported in that field [16, 17, 26, 29]. As we will show, our tracking is somewhat simpler than many of the recent methods (*e.g*. it does not allow for 3D scene estimation). However, this comes with a benefit of efficiency, and furthermore, we argue that for many AR applications 2D augmentation is fully sufficient. A further contribution on the client-side tracking element is the integration with sensors. Opposed to other works, we do not attempt to fuse the information from visual and sensor modalities, but use the sensors to reset the visual tracking when needed.

In summary, the main contributions of this work are:

- An efficient combination of client-side object tracking and server-side object recognition.
- The complete integration of a server-side object database covering millions of objects.
- Memory efficient geometric verification method for to state of the art large-scale object retrieval methods to fulfill near real-time requirements for AR.
- The integration of sensor data in order to reset client-side tracking.
- A fully functional implementation of the system on the Android platform.

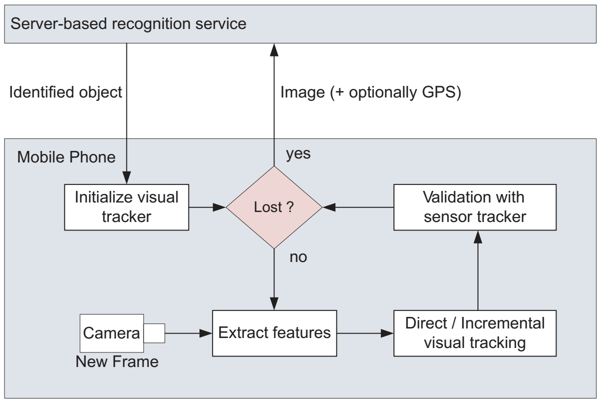
Server-based recognition service
Identified object
Image (+ optionally GPS)
Mobile Phone
yes
Initialize visual tracker
Lost ?
Validation with sensor tracker
no
Camera
New Frame
Extract features
Direct / Incremental visual tracking

Figure 2. The mobile phone sends recognition requests to an external service. Once information on a recognized object is returned, the phone tracks the position of said object using a combination of vision and sensor based trackers. Whenever a tracked object is "lost" a new recognition request is issued to reinitialize tracking.

With these contributions our system makes a significant step ahead from earlier prototype applications, bringing vision-based AR for mobile phones significantly closer to real world applications.

The remainder of this paper is organized as follows: Section 2 gives an overview over the system architecture. Sections 3 and 4 describe the server-side recognition and client-side tracking, respectively. Implementation details are given in Section 5 and experimental results are shown in Section 6.

2. System overview

As motivated in the previous section, we propose a system, which delegates recognition tasks to a server and executes tracking operations on the client. The purpose of this is to provide *large scale* object recognition coupled with a *responsive user interface* for mobile AR. When a relevant object is seen through the mobile phone's screen it should immediately be overlaid with relevant information, ideally without noticing the processing of a search request in the backend. To that end we actively try to minimize the number of requests sent to the server by determining if a new object may have entered the camera's field of view. Figure 2 gives an overview over our system. The tracker running on the client initiates a request to a recognition service, transmitting both a frame grabbed from the camera feed and optionally GPS coordinates over a HTTP connection. The moment for sending a request is chosen based on a heuristic in the tracking algorithm, which will be explained in detail in Section 4. In fact, in our application the client sends the request to two recognition services in parallel, namely one for stationary objects (landmarks) and one for non-stationary objects (products, media covers). Details for this step will be given in Section 3. The server's response consist of XML data containing the information about the recognized object (title,id) and the location of the detection in the query frame (bounding box coordinates). The bounding box coordinates are then used by the client to (re-)initialize the tracker, and the title is used to label the augmentation on the screen. The id of the object can be used to obtain additional information about it on user's request. This operation initiates a request to a dedicated object information web-service, which will not be explained in further detail in this paper. The following sections will describe the individual components of the system in more detail.

Figure 3. Sample images from a typical image cluster.

3. Server-side object recognition

The tasks of the server side recognition module are to accept query images, to identify any objects present including their location in the image, and return the response to the client. To that end we build on state of the art approaches using visual vocabularies of local image features [11, 19, 20]. More precisely, for landmark recognition, we follow very closely our earlier work [11], which in turn merges a data crawling method with the scalable object recognition method of Philbin *et al*. [20]. We then combine our own service for landmark recognition with the media recognition service offered by the company kooaba. Their API gives access to recognition for a couple of million media covers (books, CDs, DVDs) and is available online [2]. We assume that kooaba's system also builds on local features and visual vocabularies, thus we will focus on summarizing the landmark recognition in this section.

3.1. Landmark recognition

A visual recognition service for AR applications, which is able to identify objects such as landmark buildings requires both a database of images, covering each object from several viewpoints, and a scalable and near real-time retrieval method on top of it. In addition, for each object in the database some description such as titles, related weblinks *etc*. should be available.

We have recently proposed such a system [11], in the context of auto-annotation for holiday photos. Ultimately, this application is similar to AR except that the query happens from a mobile device and demands for a close to realtime response. Thus, for this paper, we implemented a similar system as in [11] and then focussed on improving its response times. We briefly summarize [11] and then explain our improvements.

Crawling and clustering of data. In order to collect a sufficient amount of images for a large number of landmarks around the world, we proposed to crawl geotagged images from Flickr. All photos, which are geographically close to each other are then also checked for their visual similarity (based on matching with SURF features [6]) and then clustered.

The outcome of this procedure is a set of clusters, where most of them represent some landmark. An example cluster is shown in Figure 3. Once clusters have been formed based on visual similarity, a post-processing step is executed. Among other things, it determines labels and related content for each object by resorting to the meta data of its cluster. Also, for each image of an object cluster, the bounding box of the object's position is calculated. For a more detailed description of these steps the interested reader is referred to [11]. Following this approach, we collected around 12 million images, which we clustered into roughly 300'000 objects.

Cluster retrieval. For all images in the cluster database their SURF features are quantized using a visual vocabulary of 1 million visual words which was learned using approximate k-means [20]. Every image in the database is then represented as a set of visual words. For every incoming query image the same procedure is applied. The query-set of visual words is matched against all database visual word sets using set intersection as distance measure.

Efficiency for this step is obtained by using an inverted file structure. For the closest 500 candidate images a geometric consistency check is performed. This is done by means of a RANSAC estimation of the homography mapping of feature matches between query and database candidate. The final matching score for each of the 500 candidate images is derived from the number of inliers.

Here, in contrast to many other implementations, we do not take the matching visual words as correspondence pairs for the homography estimation. Instead we reestimate the correspondences based on the second nearest neighbor distance ratio of the original features as proposed in [18]. This approach yields far better retrieval accuracy. A drawback is that for each query 500 SURF feature files need to be loaded into memory from disk.

The final step is a voting procedure, where the retrieved database images vote for their clusters and the cluster which receives the most votes is selected as the final result.

Optimized response times with compressed features. The geometric verification step turned out to be the bottleneck for achieving AR compatible response times. Assuming a single file access takes roughly 10 ms, then geometric verification alone would take at least 5 seconds. So, instead of loading features from disk, we compress them using a product quantizer [14] in order to be able to keep the features in memory.

The main idea of [14] is to decompose the feature space into a Cartesian product of low dimensional subspaces and to quantize each subspace separately. A feature vector is then represented by a short code composed of its subspace quantization indices. In our implementation first the entire 64-dimensional feature space is quantized using only 8 centroids (3 bits). Then, for every feature vector in the training set we subtract it's closest centroid and the marginalized features are used to train a product quantizer. To that end, we divide the 64-dimensional feature space into 16 4-dimensional subspaces and quantize features in each subspace using k-means with 1024 centroids (*i.e.* 16×10 bits). Thus, a feature vector is now represented by a compact $160 + 3$ bit code instead of 64 float or integer values.

For one million images we require now approximately 30 GB of storage instead of 300 GB. This training procedure was carried out on the features of 5000 images. During feature correspondence search in the geometric verification step, we only consider features as matching candidates if their 3 bit coarse quantization indices match. We also found that in contrast to image retrieval with large vocabularies it doesn't matter whether the quantizer is learned on the database images themselves or on an independent set.

With this adaptation, we gain a significant speed improvement over [11]. In the original implementation, geometric verification is performed using uncompressed feature files that are (due to their size) loaded from disk. The resulting recognition times would be on the order of 10 seconds, which is not sufficient for an AR application. Using our in-memory quantized features we achieve recognition times strictly under 2 seconds.

4. Client-side object tracking

Once an object has been recognized by the server and the response has been received by the mobile phone, a virtual label is attached to the object. The label's on screen position has to be updated for every frame, *i.e.* we need to track the object. Thanks to the equipment of modern smartphones with camera, accelerometer and a magnetic compass, we have several options for tracking at our disposal. Tracking based on sensors has received significant attention due to its simplicity. From the sensors alone the phone's pose can be estimated, but since signals from the sensors are often noisy the resulting AR experience is often not optimal. However, sensors are not subject to any drift which is an important property we can leverage to make overall tracking more robust.

The other option for tracking is visual tracking using image features. This is much more accurate than sensor based tracking, however it can easily fail during rapid movement and in some cases it can be subject to drift. Our strategy is to combine sensor based tracking and visual tracking to keep the good bits of both methods. Throughout this section, we make the simplifying assumption that a tracked object's movement is orthogonal to the cameras viewing direction, so that we can model an object's pose merely using a translation and a rotation. This assumption holds true for most objects that are at a reasonable distance from the camera, which applies to most AR scenarios. Furthermore, for the placement of the label we even discard the rotational component of the movement, since we focus on text labels as augmentation, which should not change orientation to ensure good readability. Thus, ultimately we are only interested in the 2D screen location of a tracked object. We found that for our purposes this simple model works very well in practice. In cases where the motion model is not able to capture an object's pose correctly (*e.g.* during scale change) the tracker can be reinitialized by sending a new request to the recognition service. Note, that this also helps to overcome drift, *i.e.* re-initiating the client-side tracker with a request to the recognition server increases robustness even more.

In the following two sections we will first describe visual tracking, and immediately after the integration of sensor tracking with the visual trackers.

4.1. Visual feature tracking

For tracking we use FAST [23] corners in conjunction with 8×8 pixel image patches as feature descriptors. Whenever a recognition request is sent to the server the extracted features of the current frame are kept as reference features.

We make a distinction between two different modes of visual tracking, *direct tracking* and *incremental tracking*. In direct tracking mode we attempt to match the features of the current frame against the reference features. If this succeeds then the recognized object can usually be very accurately located, with localization error typically in the order of a single pixel. Direct tracking can fail for a number of reasons, the most obvious of which is when the recognized object moves outside the field of view of the camera. Even when a recognized object is visible on the screen, direct tracking can fail since for objects very close to the camera the assumption of Euclidean motion between frames might not hold true and the simple image patch features we use are neither rotation nor scale invariant.

In these situations incremental tracking jumps in, which matches the features of the current frame against the features of the previous frame, in order to estimate the inter-frame motion. The location of a tracked object is then updated solely based on the incremental movements estimated between successive frames. Due to the limited accuracy of the feature detection ($\pm$ 0.5 pixel) and image noise an accumulation of small localization errors may happen, which means incremental tracking is subject to drift. Thus, if the tracker remains in this mode for several successive frames, a new recognition request is initiated.

In both tracking scenarios feature correspondences are calculated using simple neighbor search with the L_2 norm on the patch descriptors. Several heuristics are put in place to further speed-up the matching process. For instance we a-priori reject features with different FAST corner polarities and under the assumption that the inter-frame motion is small we can even further reduce the set of correspondence candidates.

4.2. Motion estimation

Once the basic feature correspondences between frames are estimated, for robust motion estimation we then use a procedure similar to what [28] refers to as "incremental tracking using pixel flow". The idea is that each inter-frame correspondence yields a translation vector which is inserted into a two dimensional histogram. The total translation is then estimated by taking the weighted sum of all pixel flows in the neighborhood of the histogram's primary mode. Since the motion model of [28] only consists of a pure translation without any rotational component, this procedure fails in the presence of rotation around the camera's viewing direction. This is because no dominant mode can be found in the translation histogram. The situation is illustrated in Figure 5.

In order to account for translation *and* rotation we need to consider at least two feature correspondence pairs at the same time. We first take into account all 780 possible pairings of the 40 best matching features correspondences (*i.e.* the ones with smallest L_2 distance). We then reject any pair of correspondences where the distance of the two points in the reference frame and the current frame changes by more than 5 pixels. This is due to the assumption of planar Euclidean motion without scale change, where the distance between two matched features is expected to remain the same in every frame. Furthermore, if the two features in either the first or second frame are less than 5 pixels apart, then the correspondence pair is also rejected.

The remaining correspondence pairs are used for motion estimation. As in [28] we use a two dimensional histogram. However, instead of considering the translation of individual correspondences, we consider the translation *and* rotation of correspondence pairs. For each considered correspondence pair $\left((\vec{A}, \vec{A}'), (\vec{B}, \vec{B}')\right)$ we now first rotate the two points $\vec{A}$ and $\vec{B}$ in the reference frame around the labels position such that the difference vectors $\vec{d} = \vec{A} - \vec{B}$ in the reference frame and the difference vector $\vec{d'} = \vec{A}' - \vec{B}'$ in the current frame become co-linear. This is illustrated in Figure 4. If $\vec{A}''$ and $\vec{B}''$ denote the rotated points in the reference frame, then a translation estimate for this feature

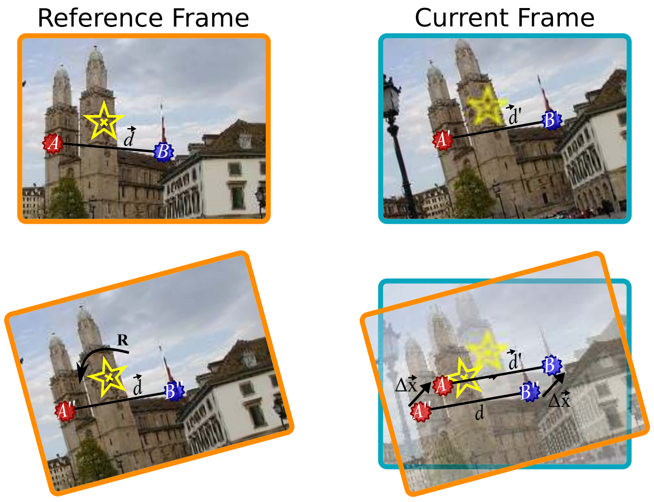

Figure 4. Before a translation estimate is calculated we compensate for any rotation around an objects label in the reference frame.

pair is given by

$$\Delta\vec{x} = \frac{\vec{A}' - \vec{A}''}{2} + \frac{\vec{B}' - \vec{B}''}{2}$$

Each resulting estimate $\Delta\vec{x}$ is then inserted into a two dimensional histogram. If the inter-frame motion can be properly approximated by only a translation and rotation, we expect to find one single mode in the histogram as demonstrated on the bottom right of Figure 5. Finally, we determine whether tracking across the two frames was successful by checking if the following conditions are met

- At least 120 votes are cast
- The histogram bin with the highest number of votes together with it's 8 neighboring bins must contain at least $\frac{1}{4}$ of all cast votes
- The second highest mode of the histogram must either be very close to the first mode or contain less than half the amount of votes

If all these conditions are satisfied, then we assume that tracking has been successful and keep a weighted sum of all translation estimates in the neighborhood of the histogram's primary mode as translation estimate. This voting scheme offers robustness against potential remaining outliers, as isolated votes will not affect the final result.

For each frame the visual tracker always first tries to perform direct tracking and uses the aforementioned criteria to determine if direct tracking was successful or not. If direct tracking fails, tracking switches to incremental mode, using the same criteria to determine if tracking was successful or not. If this also fails, we can still resort to sensor tracking, which is described in the next section.

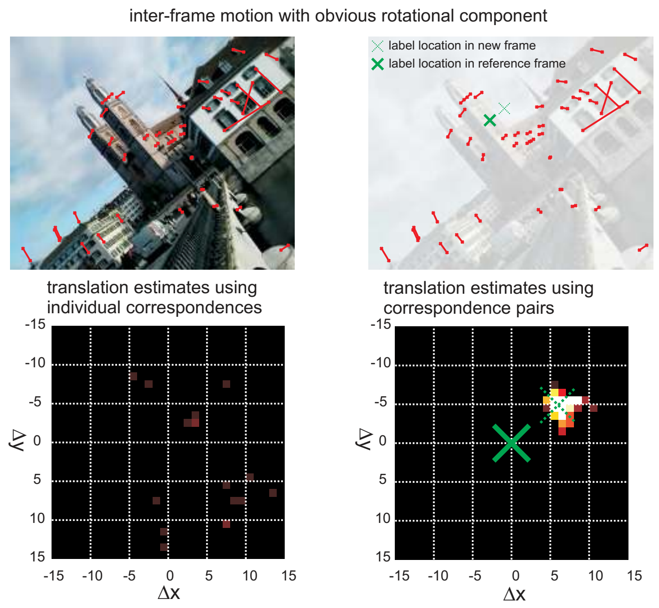

Figure 5. Bottom left: When considering the translation vectors of individual correspondences in the presence of a rotation, then no distinct translation direction can be determined. Bottom right: If however we compensate for the rotational component around an object label in the reference frame, then a clear consensus on the labels translation vector is found.

4.3. Sensor tracking

We now want to combine our visual tracking with sensor tracking. Instead of trying to fuse the cues from both trackers, we use the input from the sensors as watchdog for the visual tracking system. As mentioned earlier, the pose of a mobile phone relative to the earth surface can in principle be fully determined from the sensors. When the camera parameters are also known, every camera pixel can be mapped to a point on a sphere with a fixed radius around the mobile phone[1]. Because the accelerometer and magnetic sensors are used to measure the earth's gravitational and magnetic field directly, the resulting pose estimates are not subject to drift. However, the sensors built into consumer devices provide only very noisy signals and we also found that on Android phones the sensors' outputs are provided at irregular time intervals. Thus in order to accurately place a label on the phone's screen these signals need to undergo heavy filtering.

We let the visual and the sensor tracker run independently. The watchdog mechanism then works as follows. For every frame where direct visual tracking succeeded we assume that the tracked object has been accurately localized, thus we update the sensor tracker's view of the world,

[1]This is exploited in AR applications like Layar [1] or Wikitude [3] where based on the user's position (obtained using GPS) markers for nearby objects like buildings or other landmarks are displayed on the mobile phone's screen.

so that it matches the visual tracker. Whenever direct tracking fails and the visual tracker is in incremental tracking mode we compare the proposed label locations of the visual and sensor tracker. When over 10 successive frames the proposed label locations do not coincide, we assume that the incremental tracker suffered from drift and assume the tracked object has been lost. In this case we submit a new recognition request to the recognition service and reset the visual tracker to match the sensor tracker's view of the world.

5. Implementation details

For the client-side implementation of our system we chose the Android mobile platform. This is due to the easy access to advanced features such as camera stream, GPS, sensor data *etc*. and the modern programming environment, which makes development more productive than on other mobile platforms. The user interface, network handling and sensor handling were implemented in Java. However, for the visual tracking we had to resort to native code in order to allow for interactive frame rates. Frames are passed from Java to native C code, which handles feature extraction, feature matching and motion estimation. Frame size is set at 480*320 pixels. With these settings we reach about 12 fps on a Google Nexus One phone. The Android platform is not yet fully optimized for vision based AR applications. There is a substantial overhead due to unnecessary memory allocation and garbage collection when grabbing frames from the camera. Additionally it is not trivial to synchronize the tracking results with the displayed frames resulting in a slightly offset label position.

6. Experiments and results

In order to perform reproducible tracking experiments we recorded video sequences together with sensor data, so that all experiments could be done offline. A groundtruth was generated by manually annotating every frame in the recorded videos. We found that even though sensor tracking is quite inaccurate with typical localization errors on the order of 50 pixels it's very effective in detecting failures caused by drift during incremental tracking. To demonstrate this we recorded a special sequence, where a rapid movement cause incremental tracking to fail. We observed that within less than 2 seconds, the sensor tracker detects this failure and resets the visual tracker (see supplemental material). Figure 6 shows an experiment for the visual part of the tracking pipeline. It can be observed, that for direct tracking the localization error remains bounded and reaches at most 4 pixels. For incremental tracking we found that even after 50 frames the accumulated error stays below 8 pixels.

Figure 7 shows frames from a sequence recorded under live operating conditions on a Google Nexus One. In the beginning the users points his phone at a book which is recognized by the kooaba [2] recognition service. Next he looks at the building in front of him which is recognized a landmark building by our own recognition service. Thanks to the integration of the server-side system proposed in [11], our augmented reality service covers hundreds of thousands of landmarks around the world.

7. Conclusions

We have demonstrated a fully functional and complete AR system which recognizes and tracks stationary as well as mobile objects. It combines a client-side implementation on an Android powered mobile phone together with a server-side object recognition service. The client-side application allows for real-time tracking and interactive usage on state of the art smart phones. Our server-side provides an AR compatible object recognition service for 300000 object clusters with response times strictly under 2 seconds using a memory efficient geometric verification method. In our whole set-up we do not use any markers and do not require GPS information for object recognition and tracking.

References

[1] http://layar.com.

[2] http://www.kooaba.com/developers/.

[3] http://www.wikitude.org.

[4] lhttp://www.google.com/mobile/goggles/.

[5] A. Adams, N. Gelfand, and K. Pulli. Viewfinder alignment. *Comput. Graph. Forum*, 27(2):597–606, 2008.

[6] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *ECCV'06*, 2006.

[7] O. Chum and J.Matas. Web scale image clustering. Technical report, Czech Technical University Prague, 2008.

[8] D. Crandall, L. Backstrom, D. Huttenlocher, and J. Kleinberg. Mapping the world's photos. In *WWW '09*, 2009.

[9] D.Wagner and D. Schmalstieg. First steps towards handheld augmented reality. In *ISWC'03*, 2003.

[10] V. Ferrari, T. Tuytelaars, and L. V. Gool. Markerless augmented reality with a real-time affine region tracker. *IEEE ISAR'01*, 2001.

[11] S. Gammeter, L. Bossard, T. Quack, and L. Van Gool. I know what you did last summer: object level auto-annotation of holiday snaps. In *ICCV '09*, 2009.

[12] J. Hays and A. A. Efros. Im2gps: estimating geographic information from a single image. In *CVPR08*, 2008.

[13] H. Jegou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *ECCV08*, 2008.

[14] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 2010. to appear.

[15] M. Kähäri and D. Murphy. Mara: Sensor based augmented reality system for mobile imaging device. In *ISMAR'06*, 2006.

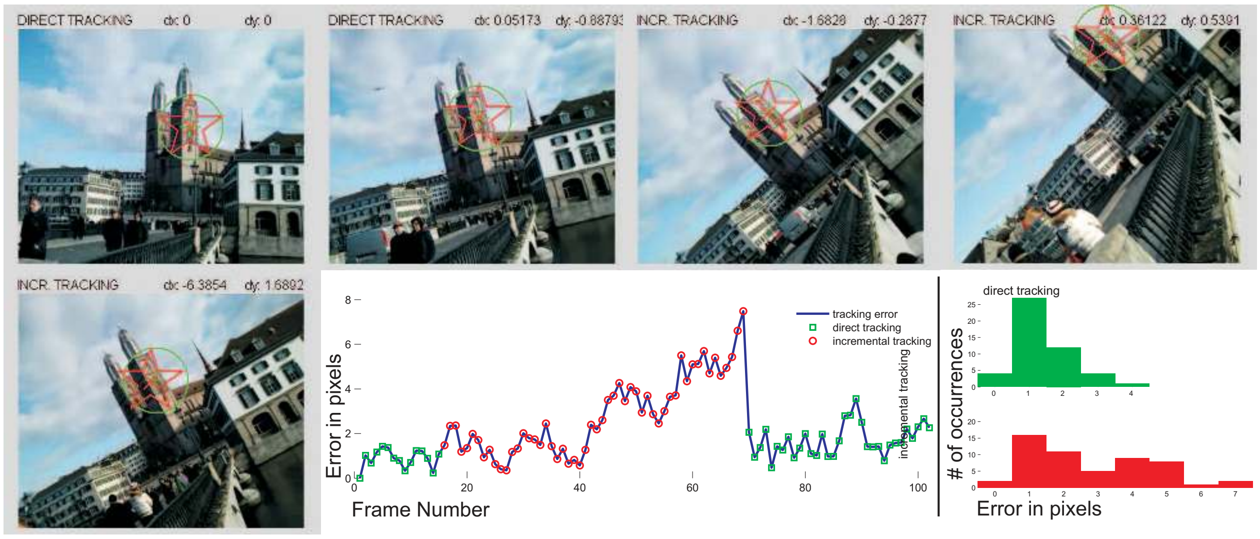

Figure 6. Top: Manually annotated test sequence. The green circle indicates the groundtruth, the red circle indicates the tracking result. Bottom: absolute tracking error in pixels. Note that incremental tracking clearly shows signs of drifting. Note how at around frame 70 the system switches back to precise direct tracking, as the original object features were found again as rotation came back towards 0.

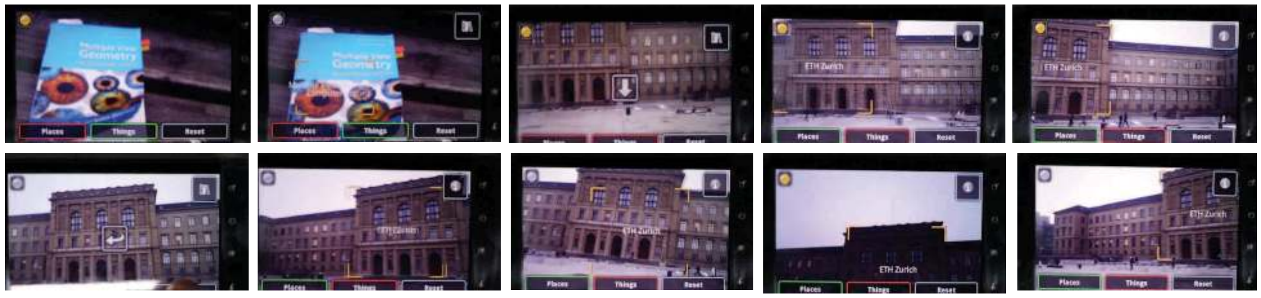

Figure 7. A real life demonstration of our application under a typical usage scenario. Note how both stationary and non stationary objects are augmented. Also note how sensor tracking kicks in once the object is out of the screen (indicated by arrows).

[16] G. Klein and D. Murray. Improving the agility of keyframe-based SLAM. In *ECCV08*, 2008.

[17] G. Klein and D. Murray. Parallel tracking and mapping on a camera phone. In *ISMAR'09*, 2009.

[18] D. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2), 2004.

[19] D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. In *CVPR'06*, 2006.

[20] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *CVPR'07*, 2007.

[21] T. Quack, B. Leibe, and L. Van Gool. World-scale mining of objects and events from community photo collections. In *CIVR '08*, 2008.

[22] M. Rohs and B. Gfeller. Using camera-equipped mobile phones for interacting with real-world objects. In *Advances in Pervasive Computing*, pages 265–271, 2004.

[23] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *ECCV06*, volume 1, pages 430–443, May 2006.

[24] J. Sivic and A. Zisserman. Video google: a text retrieval approach to object matching in videos. In *ICCV'03*, 2003.

[25] I. Skrypnyk and D. Lowe. Scene modelling, recognition and tracking with invariant image features. In *ISMAR'04*, 2004.

[26] D. Ta, W. Chen, N. Gelfand, and K. Pulli. Surftrac: Efficient tracking and continuous object recognition using local feature descriptors. In *CVPR09*, 2009.

[27] G. Takacs, V. Chandrasekhar, N. Gelfand, Y. Xiong, W.-C. Chen, T. Bismpigiannis, R. Grzeszczuk, K. Pulli, and B. Girod. Outdoors augmented reality on mobile phone using loxel-based visual feature organization. In *MIR '08*, pages 427–434, New York, NY, USA, 2008. ACM.

[28] D. Wagner, T. Langlotz, and D. Schmalstieg. Robust and unobtrusive marker tracking on mobile phones. In *ISMAR'08*, 2008.

[29] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg. Pose tracking from natural features on mobile phones. In *ISMAR'08*, 2008.