Pose Tracking from Natural Features on Mobile Phones

Daniel Wagner[1], Gerhard Reitmayr[2], Alessandro Mulloni[3], Tom Drummond[4], Dieter Schmalstieg[5]

[1 3 5] Graz University of Technology [2 4] University of Cambridge

ABSTRACT

In this paper we present two techniques for natural feature tracking in real-time on mobile phones. We achieve interactive frame rates of up to 20Hz for natural feature tracking from textured planar targets on current-generation phones. We use an approach based on heavily modified state-of-the-art feature descriptors, namely SIFT and Ferns. While SIFT is known to be a strong, but computationally expensive feature descriptor, Ferns classification is fast, but requires large amounts of memory. This renders both original designs unsuitable for mobile phones. We give detailed descriptions on how we modified both approaches to make them suitable for mobile phones. We present evaluations on robustness and performance on various devices and finally discuss their appropriateness for Augmented Reality applications.

KEYWORDS: pose tracking, natural features, mobile phones

INDEX TERMS: H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems – Artificial, augmented, and virtual realities; I.4.8 [Image Processing and Computer Vision]: Scene Analysis – Tracking

1 INTRODUCTION

Tracking from natural features is a complex problem and usually demands high computational power. It is therefore difficult to use natural feature tracking in mobile applications of Augmented Reality (AR), which must run with limited computational resources, such as on Tablet PCs.

Mobile phones are very inexpensive, attractive targets for AR, but have even more limited performance than the aforementioned Tablet PCs. Phones are embedded systems with severe limitations in both the computational facilities (low throughput, no floating point support) and memory bandwidth (limited storage, slow memory, tiny caches). Therefore, natural feature tracking on phones has largely been considered prohibitive and has not been successfully demonstrated to date.

In this paper, we present the first fully self-contained natural feature tracking system capable of tracking full six degrees of freedom (6DOF) at real-time frame rates (20Hz) from natural features using solely the built-in camera of the phone.

To exploit the nature of typical AR applications, our tracking techniques use only textured planar targets, which are known beforehand and can be used to create a training data set. Otherwise the system is completely general and can perform initialization as well as incremental tracking fully automatically.

We have achieved this by examining two leading approaches in feature descriptors, namely SIFT and Ferns. In their original published form, both approaches are unsuitable for low-end embedded platforms such as phones. Some aspects of these techniques are computationally infeasible on current generation phones and must be replaced by different approaches, while other aspects can be simplified to run at the desired level of speed, quality and resource consumption.

The resulting tracking techniques show interesting aspects of convergence, where aspects of SIFT, Ferns and other approaches are combined into a very efficient tracking system. The resulting tracker is 1-2 orders of magnitude faster than naïve approaches towards natural feature tracking and therefore also very suitable for more capable computer platforms such as PCs. We back up our claims by a detailed evaluation of the trackers' properties and limitations that should be instructive for developers of computer vision based tracking systems, irrespective of the target platform.

[1]e-mail: wagner@icg.tugraz.at
[2]e-mail: gr281@cam.ac.uk
[3]e-mail: mulloni@icg.tugraz.at
[4]e-mail: twd20@cam.ac.uk
[5]e-mail: schmalstieg@icg.tugraz.at

2 RELATED WORK

To our best knowledge, there have been no reports so far describing a real-time 6DOF natural feature tracking system on mobile phones. Instead, previous work can be categorized into three main areas: General natural feature tracking on PCs, natural feature tracking on phone outsourcing the actual tracking task to a PC, and marker tracking on phones.

Natural feature tracking approaches differ mostly by the image features that are associated between the video image and a model of the object or environment to be tracked. The dominant trade-off is between the reliability of relocating the features and the computational work required to do so.

Point-based approaches use interest point detectors and matching schemes to associate 2D locations in the video image with 3D locations. The location invariance afforded by interest point detectors is attractive for localization without prior knowledge and wide-base line matching. However, computation of descriptors that are invariant across large view changes is usually expensive. Skrypnyk and Lowe [23] describe a classic system based on the SIFT descriptor [15] for object localization in the context of AR. Features can also be selected online from a model [2] or mapped from the environment at runtime [5][12]. Lepetit et. al [13] recast matching as a classification problem using a decision tree and trade increased memory usage with avoiding expensive computation of descriptors at runtime. A later improvement described by Ozuysal et. al [18] improves the classification rates while further reducing necessary computational work. Our work investigates the applicability of descriptor-based approaches like SIFT and classification like Ferns for use on mobile devices which are typically limited in both computation and memory. Other, potentially more efficient, descriptors such as SURF [1] have been evaluated in the context of mobile devices [4], but also have not attained real-time performance yet. One good survey of local feature descriptors can be found in [16].

To reduce the computational load of searching the whole image for point correspondences, edge-based approaches use prior information about the pose and conduct a local search around the estimated location. To detect an edge, 1-D searches from sample

points along the line are sufficient to establish measurements for pose updates [6]. Various improvements to this scheme were proposed to improve the matching of lines, including statistical appearance models [26] or model-based appearances [19]. Other work combines edge tracking with other sensors in hybrid systems [10][11]. However, edge-based systems cannot stand alone, because the indistinct appearance of edges makes initialization infeasible.

One approach to overcome the resource constraints of mobile devices is to outsource tracking to PCs connected via a wireless connection. All of these approaches suffer from low performance due to restricted bandwidth as well as the imposed infrastructure dependency, which limits scalability in the number of client devices. The AR-PDA project [7] used digital image streaming from and to an application server, outsourcing all processing tasks of the AR application reducing the client device to a pure display plus camera. Shibata's work [22] could adapt how much work it outsourced. The project aimed at load balancing between client and server - the weaker the client, the more tasks are outsourced to a server. Hile reports a SIFT based indoor navigation system [9], which relies on a server to do all computer vision work. The server-based approaches are not real-time; typical response times are reported to be ~10 seconds for processing a single frame.

Naturally, first inroads in tracking on mobile devices themselves focused into fiducial marker tracking. Nevertheless, only few solutions for mobile phones have been reported in literature. In 2003 Wagner et. al ported ARToolKit to Windows CE and thus created the first self-contained AR application [28] on an off-the-shelf embedded device. This port later evolved into the ARToolKitPlus tracking library [27]. In 2005 Henrysson [8] created a Symbian port of ARToolKit, partially based on the ARToolKitPlus source code. In 2004 Möhring [17] created a tracking solution for mobile phones that tracks color-coded 3D marker shapes. Around the same time Rohs created the VisualCodes system for smartphones [20]. Both techniques provide only simple tracking of 2D position on the screen, 1D rotation and a very coarse distance measure. Similarly, TinyMotion [29] tracks in real-time using optical flow, but does not deliver any kind of pose estimation. Takacs et al. recently implemented the SURF algorithm for mobile phones [24]. They do not target real-time 6DOF pose estimation, but maximum detection quality. Hence, their approach is two orders of magnitude slower than the work presented here.

3 Natural Feature Matching

3.1 Scale invariant feature tracking

The SIFT approach is composed of three main steps: keypoint localization, feature description and feature matching. Although SIFT is often associated only with the second step, Lowe's approach specifically combines all three.

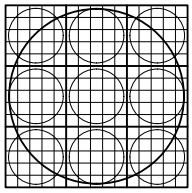

Figure 1. SIFT descriptor layout for 3x3 sub-regions.

The SIFT descriptor itself is actually neither rotation nor scale invariant. To overcome this both parameters are provided by the keypoint detector. In the first step, keypoint localization, Lowe suggests smoothing the input image with Gaussian filters at various scales and then calculating the Difference of Gaussians (DoG), which presents a fast approximation of the Laplacian operator. Keypoints are finally located by searching for scale-space extrema (minima and maxima in the DoG pyramid). Naturally the creation of the Gauss convolved image scales plus the min/max search is computationally very expensive.

While the keypoint localization step already provides a scale estimate for making the descriptor scale invariant (by increasing the kernel correspondingly), the feature's rotation has to be estimated separately. Lowe suggests calculating gradient orientations and magnitudes around the keypoint, which then form a histogram of orientations. Searching for peaks in the histogram finally assigns one or more orientations to the keypoint.

The actual descriptor is again based on gradients. The region around the keypoint is split into sub-regions that define parts of the describing feature vector (see Figure 1). The gradients are weighted by distance from the center of the patch (indicated by the large circle in Figure 1) as well as by the distance from the center of the corresponding sub-region (indicated by the 9 small circles in Figure 1). The length of the descriptor depends on the quantization of orientations (usually 4 or 8) as well as the number of sub-regions (usually 3x3 or 4x4). Although Lowe describes and analyzes several combinations of these parameters, most SIFT implementations use 8 orientations and 4x4 sub-regions, which provide best results, but create a rather large feature vector of a size of 128 dimensions.

3.2 Ferns: Tracking by classification

Contrary to descriptor-based matching as described in the last section, feature classification for tracking [18] works by learning the distribution of some features F of a set of classes C corresponding to model points m_C in a model image. At runtime, interest points are detected using some interest point detector, the value of feature F for an interest point is computed and the point is classified by maximizing the probability of observing the feature value F

$$C = \text{argmax } \mathrm{P}(C_i \mid F) \text{ over } C_i.$$

The model point m_C corresponding to the class C of an interest point is then used as the 3D correspondence for subsequent pose estimation. Different to feature matching approaches, the classification scheme is not based on a distance measure, but trained to optimize recognition of the feature points in the original model image. The classification scheme can be less computationally intensive, depending on the basic features used.

The Ferns [18] classification uses binary features that compare image intensities $I(p)$ in the neighborhood of interest points p. A binary feature F is a function $F(p)$ parameterized by a pair of offsets (l,r) such that

$$F(p) = \begin{cases} 1 & \text{if } I(p+l) < I(p+r) \\ 0 & \text{otherwise.} \end{cases}$$

For a set of N features F_i the probability of observing class C can be computed using Bayes theorem as

$$\mathrm{P}(C \mid \{F_i\}) = \mathrm{P}(\{F_i\} \mid C)\ \mathrm{P}(C) \,/\, \mathrm{P}(\{F_i\}).$$

The denominator is only a scaling constant, and the prior $\mathrm{P}(C)$ is assumed to be uniform. The probability $\mathrm{P}(\{F_i\} \mid C)$ is learned in a training phase by counting the occurrence of $\{F_i\}$ for many different examples of the same model point and thus corresponding class C.

Example views are created by applying changes in scale, rotation and affine warps and added pixel noise. These

modifications provide a local approximation to the appearance changes that are created by different view points of the model feature. The different results of computing $F_1,F_2,...,F_N$ are counted in a 2^N sized histogram describing an empirical distribution of $P(\{F_i\}|C)$ (see Figure 2).

To classify an interest point p as a class C corresponding to model feature m_C, we evaluate the probability of observing the features $\{F_1,F_2,...,F_N\}$ given class C by computing the result string of 0 and 1s, combining it into an index number and using the index to lookup the probabilities in the empirical distribution. The class C yielding the highest probability is then the resulting classification. Both training and classification operate on images smoothed with a Gaussian filter.

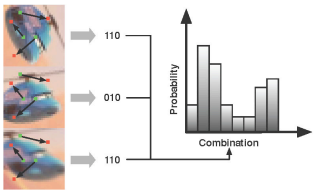

Figure 2. Learning distribution of features F from several examples and storing the occurrence of outcomes.

For practical numbers of N, the size of the full joint distribution is too large to be fully represented. Instead it is approximated by subsets of features, so called *Ferns*, for which the full distribution is stored. For a fixed Ferns size S, $M = N/S$ such Ferns F_S are created. The probability $P(\{F_i\}|C)$ is then approximated as

$$P(\{F_i\}|C) = \prod P(F_S \mid C).$$

In practice, probability values are computed as log probabilities and the product in the last equation is replaced with a sum.

4 Making Natural Feature Tracking Feasible on Phones

In the following we describe our modified approaches of the SIFT and Ferns techniques. Since the previous section already gave an overview on the original design, we concentrate on changes that made them suitable for mobile phones.

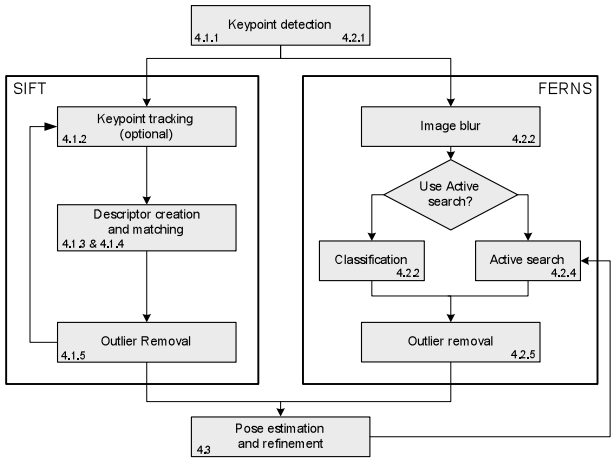

Figure 3. Overview of the SIFT and Ferns pipelines.

Four major steps make up the pipeline of a feature based pose tracking system (see Figure 3): (1) Feature detection (and tracking), (2) Feature description and matching, (3) Outlier removal and (4) Pose estimation.

Our implementations of the SIFT and Ferns techniques share the first and last steps: Both use the FAST [21] corner detector to detect feature points in the camera image, as well as Gauss-Newton iteration to refine the pose originally estimated from a homography.

4.1 Modified SIFT for phones

In the following we describe in detail how we modified the SIFT algorithm to achieve real-time performance on mobile phones. We begin with describing all steps of the run-time pipeline and finish with the offline target data acquisition, since it relies on the same techniques as used at runtime.

4.1.1 Feature detection

The original SIFT algorithm uses Difference-of-Gaussians (DoG) to perform a scale-space search that not only detects features but also estimates their scale. Although several faster implementations of Lowe's approach have been proposed, the approach is inherently resource intensive and therefore not suitable for real-time execution on mobile phones. We therefore replaced the DoG with the FAST corner detector with non-maximum suppression that is known to be one of the fastest corner detectors, but still provides a high repeatability.

Since our approach does not estimate a feature's scale anymore, the resulting descriptor is not scale invariant in the sense of the original SIFT implementation. To reintroduce scale estimation, the descriptor database contains features from all meaningful scales (see more details in section 4.1.6). Consequently, we trade memory for speed: at the cost of potentially describing the same feature multiple times over various scales we can avoid the CPU intensive scale-space search. Due to the low memory requirements per SIFT descriptor, this approach turns out to be reasonable.

By varying the threshold of the FAST corner detector we can dynamically adjust the number of corners found. Optimizing for ~150 features per frame turns out to be good balance between finding enough features for matching and processing speed.

4.1.2 Feature tracking

After new features have been detected, they can optionally be tracked by cross correlation. While feature tracking is not mandatory since the match step is independent of frame-to-frame coherence, it provides two benefits: Most obviously, tracking of features gives a speed up, since features that were tracked reliably don't have to be described and matched against the SIFT database. At the same time feature tracking also improves the overall robustness since features that passed all outlier tests are forwarded with highest confidence values into the next frame, which improves outlier removal.

Our feature tracker follows both "good" and "bad" features from frame to frame. Good features passed all tests and finally contributed to the pose estimation in the previous frame. Hence, they provide a good basis for the next camera frame. Bad features could either not be matched or were filtered out by the outlier removal step. Since a bad feature is likely to be re-detected in the next frame, much processing time can be saved by forwarding this information to the next frame. Forwarding both good and bad features removes the need to describe and match them, resulting in a considerable speedup.

To track features we extract patches of a size of 8x8 pixels that are blurred using a 3x3 Gaussian kernel. The blurring step makes the features more robust against any kind of affine transformation and slightly incorrect feature coordinates. A Sum of Absolute Difference (SAD) measure is used to estimate patch similarity.

We allow an average difference of up to 8% (empirically determined) per pixel to treat a feature as correctly matched.

Features are only tracked in a search radius of 25 pixels. To speed-up the search for neighboring features, all new coordinates are entered into a 2D grid that provides almost constant search time per feature.

4.1.3 Descriptor creation

Although Lowe describes several versions of his SIFT descriptor [15], most people associate SIFT only with its most complex variant, which is built from 4x4 sub-regions with 8 gradient bins each, resulting in a 128 dimensional vector. For performance and memory reasons we decided to use a variant using only 3x3 sub-regions with 4 bins each (resulting in a 36 dimensional vector). As Lowe outlines, this variant performs only ~10 percent worse than the best variant.

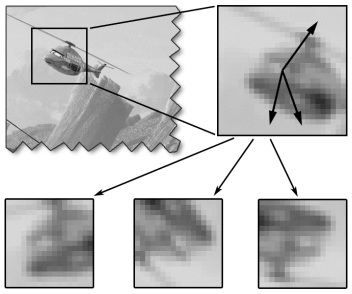

Figure 4. Extraction of SIFT features.

Since our approach is not based on interest points providing scale information, the SIFT kernel is always 15 pixels wide (3 sub-regions with a size of 5 pixels each, see Figure 1). To gain more robustness, we again blur the patch with a 3x3 Gaussian kernel (see Figure 4). Like in the original implementation we first estimate the main orientations from the patch's gradients: for all pixels of the kernel, we calculate the gradient direction and magnitude. The gradient direction is quantized to [0..35] to select the corresponding target bin. The gradient magnitude is weighted using a distance measure and is added to the respective bin. The resulting histogram is then searched for peaks. If more than 3 peaks are found, the feature is discarded.

For each detected orientation, the patch is rotated using sub-pixel accuracy to compensate that orientation. Based on the rotated patches, descriptor vectors are created: Gradient magnitudes and orientations are estimated again and weighted by their distance to the patch center as well as to the sub-region center. The weighted magnitudes are then written into the 4 bins corresponding to the sub-region.

Using gradients makes the approach invariant to constant brightness changes. Furthermore the vector is normalized to compensate for linear brightness changes. Finally any entries that are longer than 25% of the overall length are cropped to reduce too strong influence of single values.

4.1.4 Descriptor matching

After the descriptors for all features detected in the new camera image (except for those tracked from the previous frame) have been created, they are matched against the descriptors in the database. Brute force matching is not an option, since for each frame ~50-100 features have to be matched against ~5000 features in the database. Since each feature is described by a 36 dimensional vector this would result in multiplying and summing up 18 million vector entries, which is infeasible to perform in real-time due to computational constraints as well as memory throughput limits. The original SIFT implementation uses a k-d Tree together with a Best-Bin-First strategy. Yet, our tests showed that even with this approach far too many vectors have to be compared for real-time performance on mobile phones.

Looking in more detail into why the k-d tree is ineffective for our purposes, we discovered that some (usually 1-3) entries of the 36 dimensional vectors vary strongly from their respective vectors in the database. These errors increase the required tolerance for searching in the tree tremendously. Hence, we decided to use a different approach. A Spill Tree [14] is a variant of a k-d Tree that uses an overlapping splitting area. Values that are within a certain threshold are dropped into both branches. With an increasing threshold, a Spill Tree is capable of tolerating more and more error at the cost of growing larger. Unfortunately errors of arbitrary amount can show up in our SIFT vectors, which renders even a Spill Tree unsuitable.

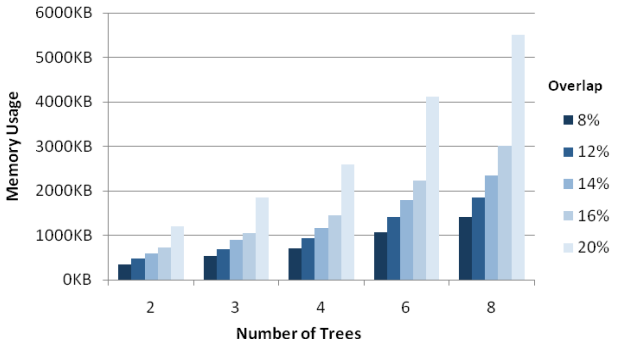

Figure 5. Spill forest memory requirements as function of number of trees and spill tree overlap percentage.

While a single Spill Tree turns out to be insufficient, we discovered that multiple trees with randomized dimensions for pivoting allow for a highly robust voting process, similar to the idea of randomized trees [13]: instead of using a single tree, we combine a number of Spill Trees into a Spill Forest. Each Spill Tree is built up to such a size, that it holds ~50-80 entries in each leaf. While trees with more levels reduce search time, more levels also increase the chance of testing a faulty dimension. Since only a few values of a vector are expected to be wrong, a vector has a high probability of showing up in the "best" leaf of each tree. We therefore only visit a single leaf in each tree and merge the resulting candidates. Descriptors that show up in only one leaf are discarded. All others are matched using Sum of Squared Difference (SSD).

Naturally, using multiple trees increases the memory requirements. Figure 5 shows the memory footprint of various Spill Forest sizes for a typical dataset, ranging from 2-8 trees and 8%-20% threshold (overlap). Our tests (see section 5.4.1) have shown that this approach finds the correct descriptor in more than 95% with reasonable memory usage while the processing time is reduced such that descriptor matching is not a bottleneck.

4.1.5 Outlier removal

Although SIFT is known to be a very strong descriptor, it still produces outliers that have to be removed before doing pose estimation. Our version of SIFT does not estimate the scale of features, so we experimented with verifying features on two successive scale levels. This approach removes nearly all outliers, but also many inliers, and it is very computationally intensive.

The outlier removal that was finally adopted therefore operates on a single-scale and works in three steps. The first step uses the orientations that have already been estimated for the descriptor

creation (see section 4.1.3). The relative orientations of all matched features are corrected to absolute rotation using the feature orientations stored in the database. Since the tracker is limited to planar targets, all features should have a similar orientation. Entering all orientations into a histogram and searching for a peak, a main orientation is quickly estimated and used to filter out those features which do not support this hypothesis. Since the feature orientations are already available, this step is very fast; at the same time it is very efficient in removing most of the outliers.

The second outlier removal step uses simple geometric tests. All features are sorted in linear time by their matching confidence using distribution sort. Then, starting with the most confident candidates, lines are estimated from two features. All other features are then tested to lie on the same side of the line in camera as well as object space. If too many features (>50%) fail the test for a single line, the test is canceled, since one of the two reference features is probably faulty. Up to 30 lines are tested. Features which fail line tests are removed.

The third outlier removal step creates homographies to remove final outliers. Features that passed the previous two tests obviously have a correct orientation and coarsely lie at the right spot in the camera image. Hence, only few outliers remain. Since a homography must be computed for the initial pose anyway, it can be used for strong final test without introducing overhead. A main problem with creating homographies is that features must not only be correct but also well-placed to be suitable: Features must not be co-linear, and their convex hull should cover a large region of the camera image.

To find good candidates, we first estimate the main direction of the point cloud using perpendicular regression. We then select features that lie at extremal positions in both directions of the line, as well as furthest perpendicular to the line. For higher robustness, we select two candidates for each direction. These 8 features are then combined into sets of 4 (one from each direction) and used to calculate 2^4=16 candidate homographies from object into camera space. All homographies are then used to test the projection of the features. The homography with the smallest number of outliers and all respective inliers are finally selected for pose refinement.

4.1.6 Target data acquisition

The SIFT tracker uses a model-based approach and hence requires a feature database that has to be prepared beforehand. Since the tracker is currently limited to planar targets, a single orthographic image of the tracking target is sufficient. Data acquisition starts by building an image pyramid. Successive pyramid levels are created by scaling down with a factor of 1/sqrt(2) from the previous level. Features will later be detected at similar scales as available in the image pyramid. Hence the range from largest to smallest pyramid level defines the range of scales that can be detected at runtime. In practice we usually create 7-8 scale levels sized from ~1MPixel down to 80KPixel. This approach varies from the one described by Lowe in that we have clearly quantized steps rather than estimating an exact scale per keypoint.

After the image pyramid has been created, we run the FAST corner detector to search for features at all scales. To restrict the creation of feature descriptors to the most stable features, we require corners to show up at two successive scales. For features passing this test, descriptors are created. Again, features with more than three main orientations are discarded. All feature coordinates plus descriptors are then stored in a feature set file.

The discretization of scale-space in this way leads to a multi-scale feature description similar to that described by Chekhlov et al. [3]. However, we compute descriptors only from scales where an interest point was found to avoid descriptors that cannot be detected.

4.2 Modified Ferns for phones

This section describes the modifications to the original Ferns [18] tracking work to operate on mobile phone devices.

4.2.1 Feature detection

The original Ferns implementation uses an extrema of Laplacian operator to detect interest points in the input image. This was replaced by the FAST detector [21] as described in section 4. Interest points are computed on 2 octaves of the image and subjected to non maximum suppression. These interest points are then classified using the Ferns classifier to yield matches with the points from the model.

4.2.2 Feature classification

The implementation of the runtime classification is straightforward and the original authors provide a simple code template to highlight this fact. Given an interest point p, the features F_i for each Fern F_S are computed and used as indexes into the histograms. The histograms store the log of the probabilities and a summation over all Ferns yields the final log of probability for each class.

The original work used Fern sizes S = 10-14 and M = 20-30 Ferns, requiring storage of more than $3 \cdot 10^5$ entries per model feature. Even if a single log probability is stored as a byte, realistic databases of at least 100 model features grow to 32Mb, exceeding by far available application memory on mobile phones.

The selection of M and S for fixed N provides a convenient way of trading off memory use and classification performance.

The overall memory usage is given by $M2^S$, so that reducing S yields more significant memory savings. For a fixed number of questions $N = SM$, Figure 6 shows the memory usage as a function of S. We experimented with S between 6 - 12 and a total N of 200 which yielded elements numbers of around between 2000 and 64000. In total, the number of entries of all histograms for 100 features then varies between $2 \cdot 10^5$ and $6.4 \cdot 10^6$.

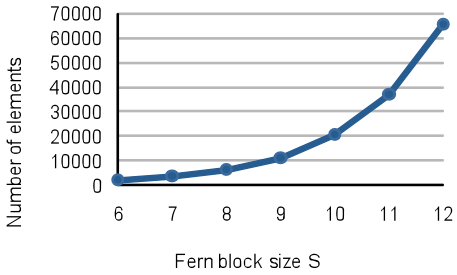

Figure 6. Memory requirements per model point.

The original work stored log probabilities as floating point values using 4 bytes per element. We found that representing the log probabilities as 8-bit bytes yields enough numerical precision to avoid any degradation in performance. A linear transformation between the range of the original log probabilities and the range [0..255] for unsigned 8-bit numbers is used, because it preserves the order of the resulting scores.

However, reducing the block size S of the FERNs empirical distribution severely impacts the classification and matching performance. Therefore we found it necessary to improve the distinctiveness of the classifier by actively making it rotation invariant. Thus, for every interest point p, we compute a dominant orientation by evaluating the gradient of the blurred image centered on the interest point. The orientation is then quantized

into [0..15] and a set of pre-rotated questions associated with each bin is used to calculate the answer sets. The same procedure is also applied in the training phase, for training the empirical distributions to compensate inaccuracy in the rotation estimation, quantization noise from the rotation quantization, and the alignment of rotated questions to pixels.

Figure 7 compares the classification performance of the different schemes using a fixed number of 200 questions. We compared using different block sizes with a fixed block size and different number of rotation bins. The graph shows both classification rate, defined as correct classification of a model point given its location, and match rate, defined as the rate at which the top log probability interest point is within a fixed small neighborhood of the true model point. Both rates were computed for artificially warped and transformed images to have ground truth. The block size was varied as S = 6, 8, 10, 12 and the rotation bins as 1, 4, 8, 16 with a fixed size S = 8.

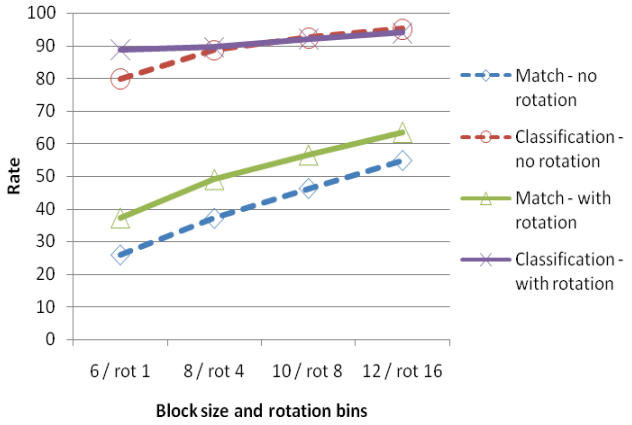

Figure 7. Classification rate (top two lines) and match rate (bottom two lines) for different Fern sizes S without rotation (dashed line) and for S = 8 with different rotation bins (solid line).

Notably, while the classification rate does not suffer much from less complex Ferns, the match rate deteriorates dramatically and is as low as 25% for the S = 6 case. Similarly the S = 8 case without rotation is just below 40% but improves to over 60% percent for 16 rotations. This is a higher rate than for the S = 12 case without rotation, while its memory requirements are only a fraction of the latter. Moreover, the match rate is the crucial parameter for successful outlier removal using RANSAC, as it determines the number of trials necessary to obtain a robust estimate.

Robust classification also requires smoothing of the input images with Gaussian blur to reduce the influence of noise and to make the differences more stable. To improve the speed of this rather expensive operation, the kernel size was chosen to be 5 pixels only and the coefficients were modified to match powers of 2 to make use of specific strengths (free barrel shifting) of the ARM processors. The resulting smoothing is not a true Gaussian blur anymore, but its use in both training and runtime results only in little degradation of the classification performance.

4.2.3 Training

FAST typically shows multiple responses for interest points detected with more sophisticated methods. It also does not allow for sub-pixel accurate or scale-space localization. These deficiencies are counteracted by modifying the training scheme to use all FAST responses within the 8-neighborhood of the model point as training examples.

Testing the resulting feature classification also allows for pruning the classification database to improve the matching performance. We remove all model points that have a matching rate below 50% to improve the overall matching rate.

4.2.4 Active search

Feature classification does not require any prior knowledge about camera pose and allows full localization at every frame. However, using information about the expected location of model points in the frame can reduce the required computational effort to establish matches and improve the inlier rate.

Using a motion model, the camera pose is predicted for the next frame. All model points are projected into the current image. For all detected interest points, the model points within a certain search radius are selected and the classification is restricted to these model points only. To speed up the search for nearby model points, a 2D grid is used, reducing the search overhead to be almost linear.

The restricted classification is more efficient as it needs to compute the probabilities only for a subset of all classes. Furthermore, as each class is matched against fewer points, the likelihood of a false match is reduced, assuming the true match is in the set of detected interest points. Finally, model points that do not project into the current view are also excluded from matching, avoiding spurious matches against invisible model points.

4.2.5 Outlier rejection

The match set returned by the classification still contains a significant fraction of outliers. Therefore a robust estimation step is required to compute the correct pose. In a first outlier removal step, we use the orientation estimated for each interest point and compute the difference to the stored orientation of the matched model point. Differences are binned in a histogram and the peaks in the histogram are detected. As differences should agree across inlier matches, we remove all matches in bins with less matches then a fraction of the peaks. We use 66% as threshold, chosen experimentally by evaluating the distributions of differences.

The remaining matches are used in an MLESAC [25] scheme to estimate a homography between the model points of the planar target and the input image. The final homography is estimated from the inlier set and used as starting point in a 3D pose refinement scheme described below.

4.3 Pose Refinement

Pose refinement estimates a 6DOF camera pose from the 2D-3D point correspondences between observed feature points and original model points using a Gauss-Newton iteration scheme to minimize the re-projection error under a standard camera model. Given a camera pose C as a rigid transformation from a world co-ordinate system into the camera coordinate system, a point x = $(x,y,z,1)$ is projected into the view to the point p = (u,v) with the following observation function

$$\boldsymbol{p} = \mathrm{proj}(C\boldsymbol{x}). \tag{1}$$

The function proj(.) models the projection from camera frame to image coordinates as

$$\begin{pmatrix} u \\ v \end{pmatrix} = \mathrm{proj}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} f_u & 0 & c_u \\ 0 & f_v & c_v \end{pmatrix} \begin{pmatrix} r' \frac{x}{z} \\ r' \frac{y}{z} \\ 1 \end{pmatrix}, \text{ where}$$
$$r = \sqrt{(\tfrac{x}{z})^2 + (\tfrac{y}{z})^2} \quad \text{and}$$
$$r' = 1 + \alpha r^2 + \beta r^4. \tag{2}$$

The term r' compensates for radial lens distortion and the parameters f_u, f_v, c_u, c_v, α and β model the intrinsic camera

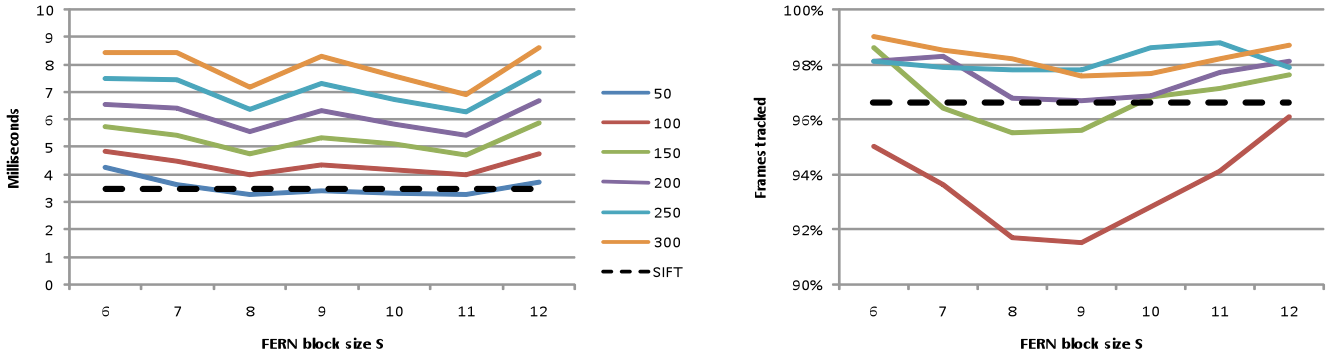

Figure 8: FERN runtime per frame (left) and robustness (right) for varying block size and number of model points. Dashed black lines represent SIFT reference. The N = 50 line for robustness (right) is around 50% and far below the shown range.

parameters, which are determined in an off-line camera calibration step.

For a set of N observations p_i of model points x_i, the observation error $e(C)$ is the sum of the individual errors d_i squared

$$e(C) = \sum_{i=1}^{N} d_i^2 = \sum_{i=1}^{N} (\boldsymbol{p}_i - \mathrm{proj}(\boldsymbol{x}_i))^2. \quad (3)$$

Minimizing $e(C)$ with respect to C provides the least-squared error estimate of the camera pose. We parameterize C by applying a small motion M to it as $C^+ = MC$. The motion M is parameterized by the six-vector μ corresponding to the exponential map parameterization of the Lie group $SE(3)$. Minimizing the error $e(MC)$ with respect to μ requires solving the over determined system of linear equations $J\,\mu = d$ where J is the Jacobian of (1) with respect to μ for every observation i

$$J_i = \frac{\partial d_i}{\partial \boldsymbol{\mu}} = \begin{pmatrix} \frac{\partial u}{\partial \mu_j} \\ \frac{\partial v}{\partial \mu_j} \end{pmatrix}. \quad (4)$$

Then the following equation gives the minimizing parameter vector μ

$$(J^T J)^{-1} J^T \boldsymbol{d} = \boldsymbol{\mu}. \quad (5)$$

The Cholesky decomposition is used to efficiently compute the pseudo inverse $(J^T J)^{-1} J^T$.

Most mobile phones today don't have floating-point units, which requires relying on fixed-point numerics. Most fixed-point algorithms tend to become unstable if too much data is involved due to the limited numerical stability. We therefore limit the pose refinement step to a maximum of 20 features. These features are selected to cover a large area of the camera image. For performance reasons we use another 2D grid for this purpose.

5 Evaluation

To create comparable results for tracking quality as well as tracking speed over various datasets (see Figure 10), tracking approaches and devices, we implemented a frame server that loads uncompressed raw images from the file system rather than from a live camera view. The frame server and both tracking approaches were ported to Windows CE and Symbian and tested in a large number of combinations, resulting in more than 1 million recorded measurements.

5.1 Ferns parameters

To explore the performance of the Ferns classification approach under different Fern sizes, we trained a set of Ferns on three data sets and compared robustness, defined to be the number of frames tracked successfully, and speed. The total number of binary features was fixed to $N = 200$ and the size of Ferns was varied between $S = 6$-12. The corresponding number of blocks was taken as $M = [N/S]$. The number of model points was also varied between $C = 50$ - 300 in steps of 50.

Figure 8 shows the robustness and speed for different values of S and C for the Cars data set. To compare the behavior of the Ferns approach to the SIFT implementation, we ran the SIFT with optimized parameters on the same data sets. The resulting SIFT performance is given as black dashed line in the graphs of Figure 8. The runtime performance seems best for the middle configurations, while small S appears to suffer from the larger value of M, while for large S the bad cache coherence of large histogram tables seems to impact performance. The lower robustness values for the mid S configurations is due to a less optimized match threshold derived by linear interpolation from thresholds for the border values. Additionally, Table 1 shows the relative memory usage of the different Fern sets versus the SIFT database.

Optimizing the Fern parameters requires both achieving similar speed and robustness as the SIFT method while not using substantially more memory. As final parameters for further evaluation we selected $S = 8$ and $M = 25$ with $C = 150$ classes.

FERN block size S \ Number of model points	50	100	150	200	250	300
6	9%	17%	26%	35%	44%	52%
7	15%	30%	44%	59%	74%	89%
8	26%	53%	79%	106%	132%	158%
9	46%	93%	139%	186%	232%	279%
10	84%	169%	253%	338%	422%	507%
11	152%	304%	456%	608%	760%	912%
12	270%	541%	811%	1081%	1351%	1622%

Table 1. Fern memory usage relative to SIFT (for Cars dataset). The black line indicates configurations with similar memory usage.

5.2 Robustness

The optimized configurations for both SIFT and Ferns from the last section were used to test robustness on different targets. We defined a pose to be found successfully, if the number of inliers is 8 or greater. This definition of robustness is used for all tests in the paper. Furthermore we also compared the pure localization at every frame with the corner tracking and active search option of the two systems.

As can be seen in the middle chart of Figure 9, the Book and

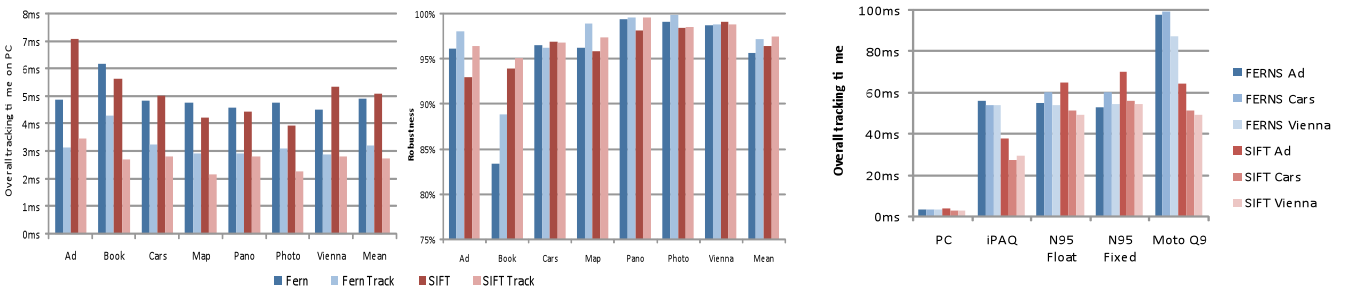

Figure 9: Left: timings of SIFT and Ferns on all 7 datasets on a PC with 2GHz. Middle: corresponding tracking robustness. Right: Timings on 5 different platforms and 3 datasets; grouped by platform and algorithm.

Figure 10: The 7 test sets (from left to right): book cover, advertisement, movie poster, printed map, panorama picture, photo, satellite image.

Advertisement datasets (first two pictures in Figure 10) performed worst. Compared to the other datasets, both targets consist of large, untextured areas that provide only few trackable features. In both cases active search and corner tracking where able to improve tracking robustness considerably, while at the same time improving tracking speed by 30-100%. Analyzing the reason for tracking failure we noticed that tracking failed a few times over longer sequences of frames, when the camera image showed mostly untextured areas.

The Pano, Photo and Vienna datasets work noticeably better, since theses images are uniformly covered with features including many different brightness levels. The Cars and Map datasets perform in medium quality for different reasons: while the Cars poster is covered with many highly detectable features in the middle area, its top and bottom areas are very weak in contrast and hence only few corners are detected there. The Map dataset on the other hand has many perfectly distributed features, but of low average quality of being tracked: there are many small areas on the map, that are uniformly colored at similar gray levels, a condition that is equally unsuitable for SIFT and Ferns.

On average, both approaches performed very similar: Both were able to track the targets in ~96% of all frames without active search or corner tracker and performed ~1-2% better with. Also the timings (on the PC) were similar at 3ms with active search and corner tracking, and 5ms without.

5.3 Performance

Finally, the overarching challenge of natural feature tracking on mobile phones is speed. To explore the operational speed of the two approaches, we evaluated the Ad, Cars and Vienna sequences from section 5.2 on the following list of devices:

- Notebook with a 2GHz Intel Core Duo (Windows XP)
- HP iPAQ 614c with a 624MHz Intel XScale PXA270 (Windows Mobile)
- Nokia N95 with a 330MHz TI OMAP 2420 (Symbian)
- Motorola Q9 with a 330MHz TI OMAP 2420 (Windows Mobile)

Both the Motorola Q9 (Windows Mobile) and the Nokia N95 (Symbian) use the same CPU, hence we expected similar benchmarking results. Although most of our code is integer or fixed-point based, we also benchmarked the N95 (who's OMAP 2420 CPU has a hardware floating point unit) with hardware floating point enabled (replacing all fixed-point code with floating-point variants). The IPAQ 614c (Windows Mobile) runs an XScale CPU with 624MHz and was therefore expected to clearly outperform the other phones. For reference, we ran all test series on a notebook with a 2GHz Core Duo. Since the code is optimized for phones it does not take advantage of the PC's 2nd CPU core.

Three different test series were run on all devices: Advertisement (bad tracking quality), the Cars poster (medium tracking quality) and the Vienna satellite image (best tracking quality).

In general, the timings (see Figure 9) show that we could achieve our goal of natural feature tracking on mobile phones operating at interactive speeds. The average runtime per frame is around 60ms for the slowest devices, resulting in a potential frame rate of 15 fps. After adding typical overheads of acquiring frames and rendering output, we can still expect to build AR systems operating at 10 fps on mobile phones.

However, our measurements did not fully reflect our expectations because the Ferns code performed much slower (~2x slower than it should) on the Windows Mobile devices, whereas it works as expected on the Symbian platform it was originally written for. The slow-down is specific to the log probability summation code which is a simple memory lookup and summation routine. Nevertheless, we were not able to identify the reason for the slow down.

Ignoring this misbehavior we notice that both Ferns and SIFT run at reasonable speed on all tested devices. The SIFT implementation, which was developed on the Windows CE platform is able to take full advantage of the 624MHz and runs at ~30 milliseconds on all test sets. Due to the problems mentioned above, the Ferns code is not able to run at full speed on this device.

On the N95 the Ferns implementation performs very similar to the SIFT tracker: both algorithms run at 50-70 milliseconds for all test sequences. Timings with fixed-point and floating-point are very similar due to the low usage of these data types across the tracking pipelines of both algorithms. As expected, the SIFT tracker runs at almost identical speed on the Motorola Q9 as on the Nokia N95, whereas the Ferns tracker again performs at only half the speed it should.

Compared to all benchmarked phones, the PC executed both approaches about 15-20 times faster. Even though its clock rate is only 3.2 times higher than the clockrate of the XScale, its much larger caches and multiple ALUs provide a clear advantage.

5.3.1 Detailed speed analysis for SIFT

Looking in more detail into what SIFT spends its computation time for (see Figure 11) shows that the most time consuming single action is corner detection, requiring ~23% of the overall time. This is not surprising since the corner detector has to look at almost all 76800 (320x240) pixels (except for those pixel close to the image border, where features can not be described).

Feature describing and matching together requires 50% of the overall time. It starts with describing the features, which includes blurring a patch, estimating its orientation, rotating to compensate orientation and finally creating the description vector. Together these 4 steps require ~21% of the overall time.

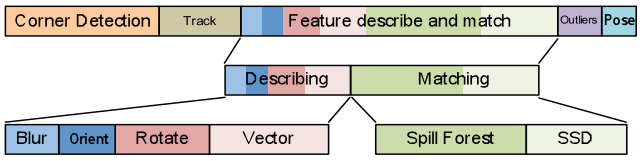

Figure 11. Relative timings of the SIFT tracker.

Matching of descriptors starts by dropping down the trees to the very first leaves, which is fast, but consecutively requires merging the lists of results. Although this merge operation requires only linear time (the candidate lists are presorted) the pure number of 200-400 candidates makes this an expensive operation. Calculating sum of squared difference (SSD) for the resulting best candidates then takes only ~11% of the overall time.

Descriptor creation and matching is computationally expensive, but is supported by corner tracking, which helps reducing the number of descriptors to create, while at the same time introducing only a small additional cost.

Outlier removal costs ~7% of the overall time per image. Most of the time is spent for creating and testing homographies for the last inlier test. Finally, pose refinement takes ~4% frame time.

5.3.2 Detailed speed analysis for Ferns

The Ferns algorithm is simpler then the SIFT algorithm and consequently consists of only a few blocks (see Figure 12). A set of operations is performed on the whole image consisting of corner detection, down sampling to create a second octave and blurring the input octave images. The remaining time is spent in the classification which is linear both in number of interest points and classes, and finally, outlier detection.

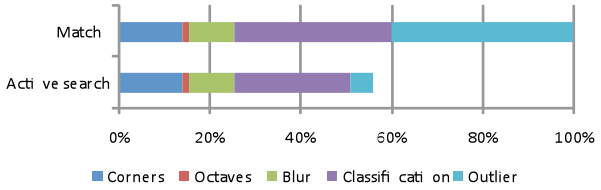

Figure 12. Relative timings of the Ferns component, normalized to the full localization mode.

The impact of active search can be observed both in the reduction of time spent in classification as fewer classes are visited, as well as in the dramatically reduced time spent in the RANSAC outlier detection stage. Here, the increased inlier rate pays off as a large set of inliers can be established quickly.

5.4 Memory tradeoffs

Both SIFT and Ferns have to trade memory for speed and/or robustness. Due to the limits of today's mobile phones, both approaches have to run with much less memory than originally designed for. In this section we discuss where most of the memory is spent and how reducing the overall memory footprint influences tracking behavior.

5.4.1 Memory for SIFT matching

SIFT descriptor datasets as tested in this paper are typically in a size from 50-100KB, which is highly suitable for mobile phones, as long as only a small number of targets shall be tracked at a time. Since brute force matching is too slow, we developed Spill Forests as described in section 4.1.4. Due to their highly redundant nature, Spill Forests can easily require several megabytes of memory for the aforementioned datasets.

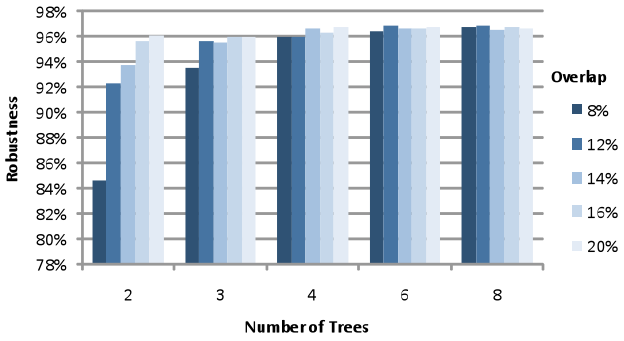

Figure 13. Robustness for SIFT descriptor searching as function of number of Spill trees and Spill tree overlap.

We experimented with various Spill Forest sizes in terms of memory usage (see Figure 5) and robustness (see Figure 13). Memory requirements increase linear with the number of trees and more than linear with overlap (threshold). Our tests show that using only 2 Spill trees, combined with a low threshold (<14%), does not yield robust matching. As can be seen in Figure 13, the first combination (sorted from lowest to highest memory usage) that allows tracking of more than 96% of all frames, uses 4 trees and a threshold of 14% and was therefore used for all tests in this publication.

5.4.2 Memory for Ferns datasets

Memory requirements for Ferns were already extensively discussed in section 4.2.2 and 5.1. The exponential dependency on the block size favors the use of more classes, but smaller blocks to achieve a desired level of robustness. As Figure 8 shows, the robustness does not depend greatly on the Ferns block size, but rather on the number of classes trained. Thus, at the expense of some computational overhead, smaller block sizes may be chosen to reduce memory footprint.

6 Conclusions and Future Work

We have presented two approaches for natural feature trackers that allow robust pose estimation from planar targets in real time on mobile phones.

The two original techniques, SIFT and Ferns, are very different in their approach – while SIFT is engineered around a highly sophisticated feature descriptor, Ferns recasts detection as classification, and relies on Bayesian statistics of large quantities of simple binary tests.

We originally assumed that the simplicity of Ferns would let it outperform the more complex SIFT on a constrained platform such as a phone. However, it turned out that in order to deliver a high level of quality Ferns requires significant amounts of memory (for a phone) and computational bandwidth to use the consumed memory. Moreover, the very simple structure of Ferns

descriptors requires more sophisticated outlier management, which consumes further computational resources.

The approach finally adopted for both shows interesting aspects of convergence: In both approaches, Laplacian/Gaussian feature detection was replaced by simple FAST detector at the expense of losing scale independence. Ferns adopted a regularization using the dominant orientation from SIFT, while SIFT adopted a search forest approach from Ferns. Two of the three steps of outlier management, namely orientation check and homography check, are shared by both approaches. A major weakness of both approaches is the rather limited tilt angle they can tolerate. While artificial fiducial markers can be detected close to 90° tilt, both SIFT and Ferns do usually not permit more than 40-50° of tilt.

The final performance of both techniques is very comparable given an equal amount of CPU performance and memory. At the lower end, SIFT may be more stable than Ferns, while Ferns may be able to outperform SIFT given an increased amount of memory and CPU bandwidth currently not afforded by phones.

The SIFT descriptor database can be limited to as little as 50K, but the Spill Forest created at runtime can require several megabytes. The Ferns datasets are much larger and have to be downloaded to the phone and stored in advance. Yet, in order to track larger or many targets concurrently, memory requirements of both approaches have to be reduced.

We observe that the level of CPU performance on phones has not increased very much in the last three years, probably because of a certain market saturation and the very tight power budget afforded by cell phone batteries. Instead, it is very likely that programmable GPUs will be embedded in multi-core phone CPUs very soon. This may enable more expensive per-pixel processing, allowing to re-introduce operations such as Laplacian/Gaussian transforms again. Depending on whether CPU or GPU enhancements become available, the choice of next generation of tracking technique may be different.

A natural future step is to extend the presented work in order to support 3D tracking targets. In the case of 3D targets, estimating a homography would not suffice anymore. Furthermore, it would be necessary to cope with self-occlusions of the tracking target.

7 Acknowledgements

The authors thank Vincent Lepetit and the computer vision group at EPFL for sample code and discussions regarding the Ferns implementation. This research was funded by the Austrian Science Fund FWF under contracts Y193 and W1209-N15, and the European Union under contract FP6-2004-IST-4-27571.

References

[1] Bay, H., Tuytelaars, T., Gool, L. V., Surf: Speeded up robust features, In Proc. ECCV 2006, 2006.

[2] Bleser, G., Stricker, D., Advanced tracking through efficient image processing and visual-inertial sensor fusion. In Proc. of IEEE VR 2008, pp. 137-144 2008

[3] Chekhlov, D., Pupilli, M., Mayol-Cuevas, W., Calway, A., Real-time and robust monocular slam using predictive multi-resolution descriptors, In Proc. ISVC 2006, pp. 276–285, 2006.

[4] Chen, W.-C., Xiong, Y., Gao, J., Gelfand, N., Grzeszczuk, R., Efficient extraction of robust image features on mobile devices, In Proc. ISMAR 2007, 2007.

[5] Davison, A.J., Mayol, W.W., Murray, D.W., Real-time localisation and mapping with wearable active vision. In Proc. of ISMAR 2003, pp. 18-27, 2003

[6] Drummond, T.W., Cipolla, R., Visual tracking and control using lie algebras. In Proc. IEEE Conf. on Computer Vision and Pattern Recognition 1999, pp. 652-659, 1999

[7] Gausemeier, J., Fruend, J., Matysczok, C., Bruederlin, B., Beier, D., Development of a real time image based object recognition method for mobile AR-devices, Proceedings of the 2nd International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa (Afrigraph 2003), pp. 133-1392003

[8] Henrysson, A., Billinghurst, M, Ollila, M.. Face to Face Collaborative AR on Mobile Phones. Proceedings International Symposium on Augmented and Mixed Reality (ISMAR'05), pp. 80-89, 2005

[9] Hile, H., Borriello, G., Information Overlay for Camera Phones in Indoor Environments, 3rd International Symposium onLocation- and Context-Awareness (LoCA 2007), pp. 68-84, 2007

[10] Jiang, B., Neumann, U., You, S., A robust hybrid tracking system for outdoor augmented reality. In Proc. of VR 2004, pp. 3-10, 2004

[11] Klein, G., Drummond, T.W.. Robust visual tracking for non-instrumented augmented reality. In Proc. of ISMAR 2003, pp. 113-122, 2003

[12] Klein, G., Murray, D., Parallel tracking and mapping for small ar workspaces. In Proc. of ISMAR 2007, pp. 225–234, 2007

[13] Lepetit, V., Lagger, P., Fua, P., Randomized trees for real-time keypoint recognition. In Proc. CVPR 2005, pp. 775-781, 2005

[14] Liu, T., Moore, A.W., Gray, A., Yang, K., An investigation of practical approximate nearest neighbor algorithms. In Advances in Neural Information Processing Systems, MIT Press, pp. 825-832

[15] Lowe, D., Distinctive image features from scale-invariant keypoints. Int. Journal of Computer Vision, Volume 60, Issue 2, pp. 91-110, 2004

[16] Mikolajczyk, K. and Schmid, C. 2005. A Performance Evaluation of Local Descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.* 27, 10 (Oct. 2005), 1615-1630. 2005.

[17] Möhring, M., Lessig, C., Bimber, C.. Video See-Through AR on Consumer Cell Phones. Proceedings of International Symposium on Augmented and Mixed Reality (ISMAR'04), pp. 252-253, 2004

[18] Ozuysal, M., Fua, P., Lepetit, V., Fast keypoint recognition in ten lines of code. In of Proc. CVPR 2007, pp. 1-8, 2007

[19] Reitmayr, G., Drummond, T.W., Going out: Robust tracking for outdoor augmented reality. In Proc. of ISMAR 2006, pp. 109-118, 2007

[20] Rohs, M., Gfeller, B., Using Camera-Equipped Mobile Phones for Interacting with Real-World Objects. Advances in Pervasive Computing, Austrian Computer Society (OCG), pp. 265-271, 2004

[21] Rosten, E. Drummond, T., Machine learning for high-speed corner detection. In Proc. of ECCV 2006, pp. 430-443, 2006

[22] Shibata, F., Mobile Computing Laboratory, Department of Computer Science, Ritsumeikan University, Japan, http://www.mclab.ics.ritsumei.ac.jp/research.html

[23] Skrypnyk, I., Lowe, D., Scene modeling, recognition and tracking with invariant image features. In Proc. of ISMAR 2004, pp. 110-119, 2004

[24] Takacs, G., Chandrasekhar, V., Gelfand, N., Xiong, Y., Chen, W.-C., Bismpigiannis, T., Grzeszczuk, R., Pulli, K., and Girod, B., Outdoors Augmented Reality on Mobile Phone using Loxel-Based Visual Feature Organization, to appear in IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 2008

[25] Tordoff, B.J., Murray, D., Guided-MLESAC: Faster image transform estimation by using matching priors. IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 27, Issue 10 pp. 1523-1536, 2005

[26] Wuest, H., Vial, F., Stricker, D., Adaptive line tracking with multiple hypotheses for augmented reality. In Proc. of ISMAR 2005, pp. 62-69, 2005

[27] Wagner, D., Schmalstieg, D., ARToolKitPlus for Pose Tracking on Mobile Devices, Proceedings of 12th Computer Vision Winter Workshop (CVWW'07), pp. 139-146, 2007

[28] Wagner, D., Schmalstieg, D.. First Steps Towards Handheld Augmented Reality. Proceedings of the 7th International Conference on Wearable Computers (ISWC 2003), pp. 127-135, 2003

[29] Wang, J. Zhai, S., Canny, J., Camera Phone Based Motion Sensing: Interaction Techniques, Applications and Performance Study, In ACM UIST 2006, pp. 101-110, 2006