Real-Time Detection and Tracking for Augmented Reality on Mobile Phones

Daniel Wagner, *Member*, *IEEE*, Gerhard Reitmayr, *Member*, *IEEE*,
Alessandro Mulloni, *Student Member*, *IEEE*, Tom Drummond, and
Dieter Schmalstieg, *Member*, *IEEE Computer Society*

Abstract—In this paper, we present three techniques for 6DOF natural feature tracking in real time on mobile phones. We achieve interactive frame rates of up to 30 Hz 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 plus a template-matching-based tracker. 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. The template-based tracker further increases the performance and robustness of the SIFT- and Ferns-based approaches. We present evaluations on robustness and performance and discuss their appropriateness for Augmented Reality applications.

Index Terms—Information interfaces and presentation, multimedia information systems, artificial, augmented, and virtual realities, 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 infeasible and has not been successfully demonstrated till date.

In this paper, we present the first fully self-contained natural feature tracking system capable of tracking full 6 degrees of freedom (6DOF) at real-time frame rates (30 Hz) 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.

We call the resulting tracking techniques PhonySIFT and PhonyFerns in this paper to distinguish them from their original variants. They show interesting aspects of convergence, where aspects of SIFT, Ferns, and other approaches are combined into a very efficient tracking system. Our template-based tracker, which we call PatchTracker, has orthogonal strengths and weaknesses compared to our other two approaches. We therefore combined the approaches into a hybrid tracking system that is more robust and faster.

The resulting tracker is 1-2 orders of magnitude faster than naïve approaches toward 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.

2 Related Work

To the best of our knowledge, our own previous work [20] represents the only published real-time 6DOF natural feature tracking system on mobile phones so far. Previous work can be categorized into three main areas: General natural feature tracking on PCs, natural feature tracking on

- *D. Wagner, G. Reitmayr, A. Mulloni, and D. Schmalstieg are with the Institute for Computer Graphics and Vision, Graz University of Technology, Inffeldgasse 16c, 2nd floor, A-8010 Graz, Austria. E-mail: {wagner, mulloni}@icg.tugraz.at, {reitmayr, schmalstieg}@tugraz.at.*
- *T. Drummond is with the Department of Engineering, University of Cambridge, Trumpington Street, Cambridge, CB2 1PZ, UK. E-mail: twd20@cam.ac.uk.*

Manuscript received 11 Feb. 2009; revised 18 May 2009; accepted 29 July 2009; published online 18 Aug. 2009.
Recommended for acceptance by M.A. Livingston, R.T. Azuma, O. Bimber, and H. Saito.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCGSI-2009-02-0021.
Digital Object Identifier no. 10.1109/TVCG.2009.99.

1077-2626/10/$26.00 © 2010 IEEE Published by the IEEE Computer Society

phone outsourcing the actual tracking task to a PC, and marker tracking on phones.

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 baseline matching. However, computation of descriptors that are invariant across large view changes is usually expensive. Skrypnyk and Lowe [16] describe a classic system based on the SIFT descriptor [12] 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], [9]. Lepetit et al. [10] 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. [14] called Ferns 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 [3], but also have not attained real-time performance yet.

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 [6] 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. Hile and Borriello report a SIFT-based indoor navigation system [8], which relies on a server to do all computer vision work. Typical response times are reported to be $\sim$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 the literature. In 2003, Wagner and Schmalstieg ported ARToolKit to Windows CE, and thus, created the first self-contained AR application [19] on an off-the-shelf embedded device. This port later evolved into the ARToolKitPlus tracking library [18]. In 2005, Henrysson et al. [7] created a Symbian port of ARToolKit, partially based on the ARToolKitPlus source code. TinyMotion [21] 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 [17]. 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 Transform (SIFT)

The SIFT [12] approach from Lowe combines three steps: keypoint localization, feature description, and feature matching. In the first step, Lowe suggests smoothing the input image with Gaussian filters at various scales and then locating keypoints by calculating scale-space extrema (minima and maxima) in the Difference of Gaussians (DoGs). Creating the Gauss convolved images and searching the DoG provide scale invariance but are computationally expensive. The keypoint's rotation has to be estimated separately: Lowe suggests calculating gradient orientations and magnitudes around the keypoint, forming a histogram of orientations. Peaks in the histogram assign one or more orientations to the keypoint. The descriptor is again based on gradients. The region around the keypoint is split into a grid of subregions: Gradients are weighted by distance from the center of the patch as well as by the distance from the center of their subregion. The length of the descriptor depends on the quantization of orientations (usually 4 or 8) as well as the number of subregions (usually 3×3 or 4×4). Most SIFT implementations use eight orientations and 4×4 subregions, which provide the best results but create a large feature vector (128 elements).

3.2 Ferns: Tracking by Classification

Feature classification for tracking [14] learns the distribution of binary features $F(p)$ of a set of model points m_c corresponding to the class C. The binary features are comparisons between image intensities $I(p)$ in the neighborhood of interest points p, parameterized by a pair of offsets (l, r): $F(p)$ is defined as 1 if $I(p+l) < I(p+r)$, and 0 otherwise. At runtime, interest points are detected and their response F to the features is computed. Each point is classified by maximizing the probability of observing the feature value F as $C = argmax_C P(C_i|F)$ and the corresponding model point m_C is used for pose estimation. Different from feature matching, the classification approach is not based on a distance measure, but trained to optimize recognition of features in the original model image.

For a set of N features F_i, the probability of observing it given class C is represented as an empirical distribution stored in a histogram over outcomes for the class C. Many different example views are created by applying changes in scale, rotation, and affine warps, and adding pixel noise, as a local approximation to viewpoint changes. The response for each view is computed and added to the histogram.

To classify an interest point p as a class C, we compute $F(p)$, combining the resulting 0s and 1s into an index number to lookup the probabilities in the empirical distribution. In practice, the size of the full joint distribution is too large and it is approximated by subsets of features ($Ferns$) for which the full distribution is stored. For a fixed Ferns size of $S, M = N/S$, Ferns F_S are created. The probability $\mathrm{P}\left(F_i|C\right)$ is then approximated as $\mathrm{P}(F_i|C) = \prod \mathrm{P}(F_S|C)$. 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

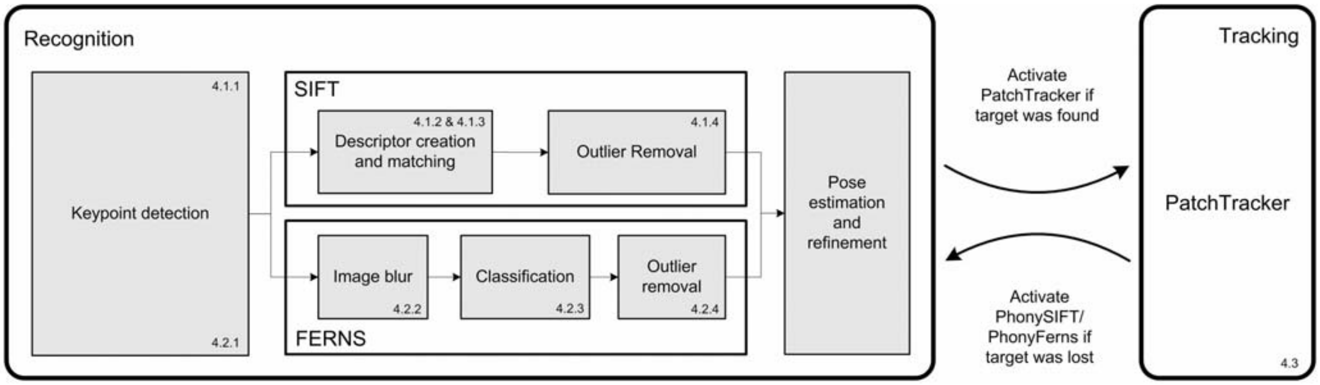

Fig. 1. State chart of combining the PhonySIFT/PhonyFerns trackers and the PatchTracker. The numbers indicate the sections in which the respective techniques are described.

phones. Four major steps make up the pipeline of a feature-based pose tracking system (see Fig. 1) as follows:

1. feature detection,
2. feature description and matching,
3. outlier removal, and
4. pose estimation.

If the PatchTracker is available (details in Section 3), the system can switch to tracking mode until the target is lost and must be redetected.

Our implementations of the SIFT and Ferns techniques share the first and last steps: Both use the FAST [15] 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 PhonySIFT

In the following, we present our modified SIFT algorithm, describing all steps of the runtime pipeline and then presenting the offline target data acquisition.

4.1.1 Feature Detection

The original SIFT uses DoGs for a scale-space search of features. This approach is inherently resource intensive and not suitable for real-time execution on mobile phones. We replaced it with the FAST corner detector with nonmaximum suppression, known to be one of the fastest detectors, still providing high repeatability. Since FAST does not estimate a feature's scale, we reintroduce scale estimation by storing feature descriptors from all meaningful scales (details in Section 4.1.5). By describing the same feature multiple times over various scales, we trade memory for speed to avoid a CPU-intensive scale-space search. This approach is reasonable because of the low memory required for each SIFT descriptor.

4.1.2 Descriptor Creation

Most SIFT implementations adopt 4×4 subregions with eight gradient bins each (128 elements). For performance and memory reasons, we use only 3×3 subregions with four bins each (36 elements) that, as Lowe outlines [12], perform only ~ 10 percent worse than the best variant with 128 elements.

Since we have fixed-scale interest points, we fix the SIFT kernel to 15 pixels. To gain robustness, we blur the patch with a 3×3 Gaussian kernel. Like in the original implementation, we estimate feature orientations by calculating gradient direction and magnitude for all pixels of the kernel. The gradient direction is quantized to 36 bins and the magnitude, weighted using a distance measure, is added to the respective bin. We compensate for each orientation by rotating the patch using subpixel accuracy. For each rotated patch, gradients are reestimated, weighted by distance to the patch center and the subregion center, and finally, written into the four bins of their subregion.

4.1.3 Descriptor Matching

The descriptors for all features in the new camera image are created and matched against the descriptors in the database. The original SIFT uses a k-d Tree with the Best-Bin-First strategy, but our tests showed that some (usually 1-3) entries of the vectors vary strongly from those in the database, tremendously increasing the required tolerance for searching in the k-d Tree, making the approach infeasible on mobile phones. A Spill Tree [11] is a variant of a k-d Tree that uses an overlapping splitting area: Values within a certain threshold are dropped into both branches. Increasing the threshold, a Spill Tree can tolerate more error at the cost of growing larger. Unfortunately, errors of arbitrary amount show up in our SIFT vectors, rendering even a Spill Tree unsuitable. We discovered that multiple trees with randomized dimensions for pivoting allow for a highly robust voting process, similarly to the randomized trees [10]: instead of using a single tree, we combine a number of Spill Trees into a Spill Forest. 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 only visit a single leaf in each tree and merge the resulting candidates. Descriptors that show up in more than one leaf are then matched.

4.1.4 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 outlier removal works in three steps. The first step uses the feature orientations. We correct all relative feature orientations to absolute rotation using the feature orientations in the database. Since the tracker is limited to planar targets, all features should have a similar orientation. We estimate a main orientation and use it to filter out all features that do not support this hypothesis. Since feature orientations are already available, this step is very fast, yet very efficient in removing most of the outliers. The second step uses simple geometric tests. All features are sorted by their matching confidence, and starting with the most confident features, we estimate lines between two of them and test all other features to lie on the same side of the line in both camera and object space. The third step removes final outliers using homographies in an RANSAC fashion allowing a reprojection error of up to 5 pixels. Our tests have shown that such a large error boundary creates a more stable inliers set, while the errors are effectively handled by the M-Estimator during the pose refinement stage.

4.1.5 Target Data Acquisition

SIFT is a model-based approach and requires a feature database to be prepared beforehand. The tracker is currently limited to planar targets, therefore, a single orthographic image of the tracking target is sufficient. Data acquisition starts by building an image pyramid, each level scaled down with a factor of $1/\sqrt{2}$ from the previous one. The largest and smallest pyramid levels define the range of scales that can be detected at runtime. In practice, we usually create 7-8 scale levels that cover the expected scale range at runtime. Different from Lowe, we have clearly quantized steps rather than estimating an exact scale per keypoint. We run the FAST detector on each scale of the pyramid. Features with more than three main orientations are discarded.

4.2 PhonyFerns

This section describes the modifications to the original Ferns [14] to operate on mobile phones.

4.2.1 Feature Detection

The original Ferns approach uses an extrema of Laplacian operator to detect interest points in input images. This was replaced by the FAST detector [15] with nonmaximum suppression on two octaves of the image. At runtime, the FAST threshold is dynamically adjusted to yield a constant number of interest points (300 for a 320×240 input image).

4.2.2 Feature Classification and Training

The runtime classification is straightforward and the original authors provide a simple code template for it. Given an interest point p, the features F_i for each Fern F_S are computed, used to look up log probabilities that are summed to give the final log of probability for each class. The original work used parameters for Fern sizes leading to databases with up to 32 Mb, exceeding by far available application memory on mobile phones. We experimented with smaller Ferns of sizes $S =$ 6-10 with about 200 questions, leading to database sizes of up to 2 Mb.

The original Ferns stored probabilities as 4-byte floating-point values. We found that 8-bit values yield enough numerical precision. We use a linear transformation between the original range and the range [0..255] because it preserves the order of the resulting scores. However, reducing the block size S of the Ferns empirical distribution severely impacts the classification performance. Therefore, we improved the distinctiveness of the classifier by actively making it rotation invariant. For every interest point p, we compute a dominant orientation by evaluating the gradient of the blurred image, quantize it into [0..15], and use a set of prerotated questions associated with each bin to calculate the answer sets. The same procedure is also applied in the training phase to account for errors in the orientation estimation.

FAST typically shows multiple responses for interest points detected with more sophisticated methods. It also does not allow for subpixel 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. Except for this modification, the training phase (running on the PC) is performed exactly as described in [14].

4.2.3 Matching

At runtime, interest points are extracted, their dominant orientation is computed, and the points are classified yielding a class and score as the log probability of being generated by that class.

For each class—and therefore, model point—the top ranking interest point is retained as a putative match. These matches are furthermore culled with a threshold against the matching score to remove potential outlier matches quickly.

The choice of threshold is typically a uniform threshold across all classes, yielding a simple cutoff. However, the probability distributions in the individual classes have different shapes with probability mass concentrated in larger or smaller regions resulting in peak probabilities varying for different classes. Consequently, this leads to different distributions of match scores. A uniform threshold may either penalize classes with broad distributions if too high, or allow more outliers in peaked distributions if too low. In turn, this affects the outlier removal stage, which either receives only a few putative matches or large sets of matches with high outlier rates.

To reduce this effect, we also train a per-class threshold. Running evaluation of the classification rates on artificially warped test images with ground truth, we record the match scores for correct matches and model the resulting distribution as a normal distribution with mean m_c and standard deviation s_c for class c. Then we use the threshold $m_c - ts_c$ as the per-class threshold (the log probabilities are negative, therefore, we shift the threshold toward negative infinity). Fig. 2 shows the average number of inliers versus the inlier rate for recorded video data using either a range of uniform thresholds or a range of per-class thresholds parameterized by $t = [0 \ldots 3]$. Ideally, we want to improve both inlier rate and absolute numbers of inliers. In practice, we chose $t = 2$ as a good compromise.

Depending on the difference in individual class distributions, the per-class thresholds can critically improve the performance of the matching stage. For data with very similar looking model points as in Fig. 2b, per-class thresholds perform not above uniform ones.

4.2.4 Outlier Rejection

The match set returned by the classification still contains a significant fraction of outliers and a robust estimation step is required to compute the correct pose. In the first outlier

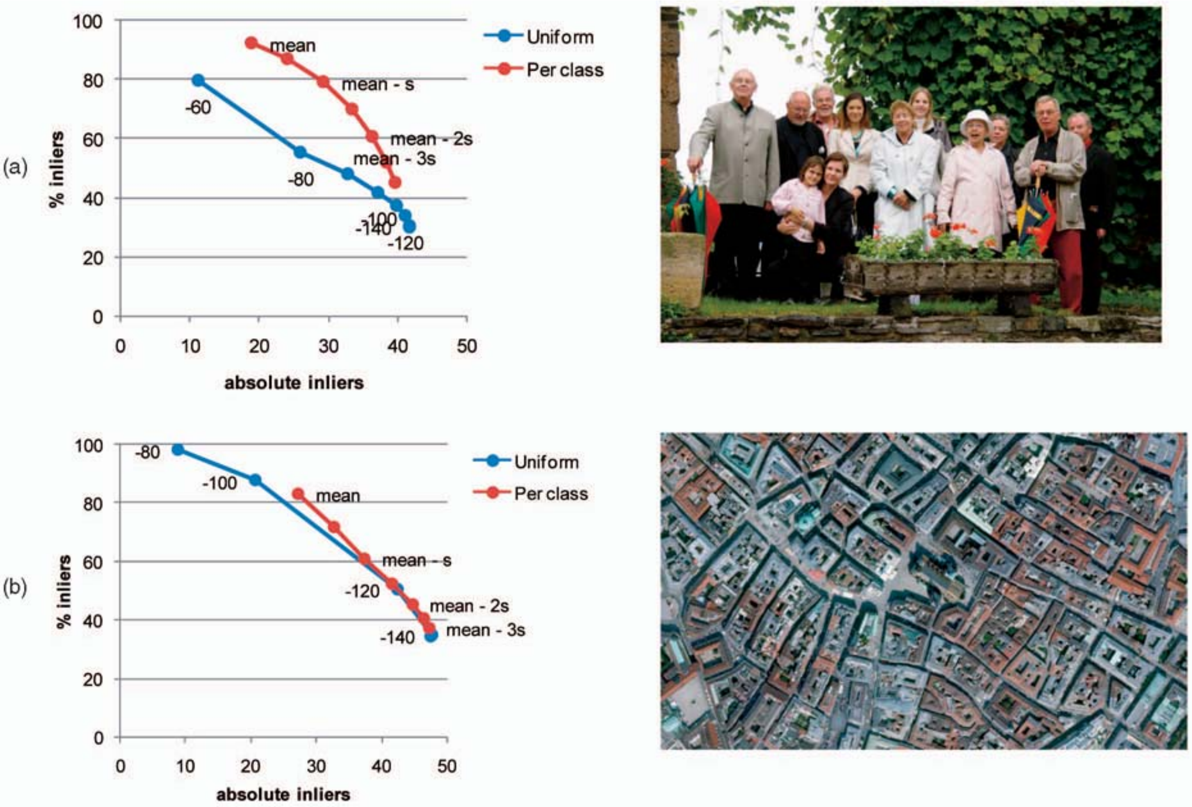

Fig. 2. Improvements in inlier rate and absolute numbers of inliers through per-class thresholds. The data labels show the uniform threshold or the parameter t for per-class thresholds. Image (a) provides different classes and matching performance can be improved significantly. Image (b) has very similar looking model points and little improvement is possible.

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 than a fraction (66 percent) of the peaks.

The remaining matches are used in a PROSAC scheme [4] to estimate a homography between the model points of the planar target and the input image. A simple geometric test quickly eliminates wrong hypotheses including colinear points. Defining a line from two points of the hypothesis set, the remaining two points must lie on their respective sides of the line in the template image as in the current frame. Thus, testing for the same sign in the signed distance from the line in both images is a simple check for a potentially valid hypothesis. The final homography is estimated from the inlier set and used as starting point in a 3D pose refinement.

4.3 PatchTracker

Both the PhonySIFT and the PhonyFerns trackers perform tracking-by-detection: For every image they detect keypoints, match them, and estimate the camera pose. Frame-to-frame coherence is not considered.

Additionally to the PhonySIFT and PhonyFerns tracker, we developed a PatchTracker that purely uses active search: based on a motion model, it estimates exactly what to look for, where to find it, and what locally affine transformation to expect. In contrast to SIFT and Ferns, this method does not try to be invariant to local affine changes, but actively addresses them. Such an approach is more efficient than tracking-by-detection because it makes use of the fact that both the scene and the camera pose change only slightly between two successive frames, and therefore, the feature positions can be successfully predicted.

The PatchTracker uses a reference image as the only data source. No keypoint descriptions are prepared. Keypoints are detected in the reference image during initialization using a corner detector. The image is stored at multiple scales to avoid aliasing effects during large-scale changes.

Starting with a coarsely known camera pose (e.g., from the previous frame), the PatchTracker updates the pose by searching for known features at predicted locations in the camera image. The new feature locations are calculated by projecting the keypoints of the reference image into the camera image using the coarsely known camera pose. We therefore do not require a keypoint detection step. This makes the tracker faster: Its speed is largely independent of the camera resolution and it does not suffer from typical weaknesses of corner detectors such as blur.

After the new feature positions have been estimated, they are searched within a predefined search region of constant size. Using the camera pose, we can create an affinely warped representation of the feature using the reference image as source (a similar approach has been reported in [13]). This warped patch of 8×8 pixels closely resembles the appearance in the camera image and its exact location is estimated using normalized cross correlation (NCC) [22] over a predefined search area. Once a good match is found,

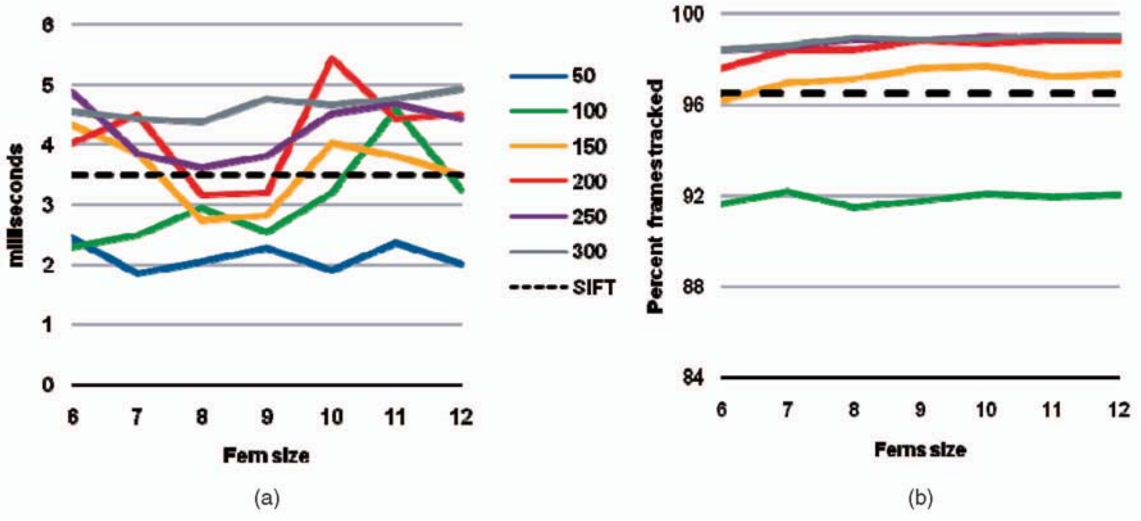

Fig. 3. PhonyFerns (a) runtime per frame and (b) robustness for varying block sizes and number of model points. Dashed black lines represent PhonySIFT reference. $C = 50$ line for robustness (b) is around 50 percent and far below the shown range.

we perform a quadratic fit into the NCC responses of the neighboring pixels to achieve subpixel accuracy.

Template matching over a search window is fast as long as the search window is small enough. However, a small search window limits the speed of the camera motion that can be detected. We employ two methods to track fast moving cameras despite small search regions.

First, we use a multiscale approach. Similar to [9], we estimate the new pose from a camera image of 50 percent size. Only few interest points are searched at this level, but with a large search radius. If a new pose has been found, it is refined from the full resolution camera image using a larger number of interest points, but with a smaller search radius. We typically track 25 points a half resolution with a search radius of 5 pixels and 100 points at full resolution with a search radius of 2 pixels only. Searching at half resolution effectively doubles the search radius.

Second, we use a motion model to predict the camera's pose in the next frame. Our motion model is linear, calculating the difference between the poses of the current and previous frames in order to predict the next pose. This model works well as long as the camera's motion does not change drastically. Since our tracker typically runs at 20 Hz or more, this is rarely the case.

The combination of a keypoint-less detector, affinely warped patches and normalized cross correlation for matching results in unique strengths: Due to using NCC (see above), the PatchTracker is robust to global changes in lighting, while the independent matching of many features increases the chance of obtaining good matches, even under extreme local lighting changes and reflections. Because of the affinely warped patches, it can track under extreme tilts close to 90 degree. The keypoint-less detector makes it robust to blur and its speed is mostly independent of the camera resolution. Finally, it is very fast, requiring only ~ 1 ms on an average PC and ~ 8 ms on a fast mobile phone in typical application scenarios.

4.4 Combined Tracking

Since the PatchTracker requires a previously known coarse pose, it cannot initialize or reinitialize. It therefore requires another tracker to start. The aforementioned strength and weaknesses are orthogonal to the strengths and weaknesses of the PhonyFerns and PhonySIFT trackers. It is therefore natural to combine them to yield a more robust and faster system. In our combined tracker, the PhonySIFT or PhonyFerns tracker is used only for initialization and reinitialization (see Fig. 1). As soon as the PhonySIFT or PhonyFerns tracker detects a target and estimates a valid pose, it hands over tracking to the PatchTracker. The PatchTracker uses the pose estimated by the PhonySIFT or PhonyFerns tracker as starting pose to estimate a pose for the new frame. It then uses its own estimated poses from frame to frame for continuous tracking. In typical application scenarios, the PatchTracker works for hundreds or thousands of frames before it loses the target and requires the PhonySIFT or PhonyFerns tracker for reinitialization.

5 Evaluation

To create comparable results for tracking quality as well as tracking speed over various data sets, tracking approaches, and situations, 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 all three tracking approaches were ported to the mobile phone to also compare the mobile phone and PC platform.

5.1 Ferns Parameters

To explore the performance of the PhonyFerns 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 (defined as finding at least eight inliers), 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.

Fig. 3 shows the speed and robustness for different values of S and C for the Cars data set. To compare the behavior of the Ferns approach to the SIFT implementation,

		Original SIFT	PhonySIFT without outlier removal	PhonySIFT with outlier removal	Original Ferns	PhonyFerns without outlier removal	PhonyFerns with outlier removal
Scale/Rotate "Boat"	Image 1	1970/1980 (99%)	316/1432 (22%)	262/319 (82%)	73/73 (100%)	128/192 (67%)	111/114 (97%)
	Image 2	1596/1603 (100%)	400/1546 (26%)	192/200 (96%)	62/64 (97%)	106/183 (58%)	66/67 (99%)
	Image 3	534/544 (98%)	298/1472 (20%)	273/295 (93%)	4/198 (2%)	25/169 (15%)	15/23 (65%)
	Image 4	362/365 (99%)	244/1277 (19%)	220/222 (99%)	1/198 (1%)	27/173 (16%)	19/23 (83%)
	Image 5	45/63 (71%)	89/1083 (8%)	31/36 (86%)	1/193 (1%)	10/166 (6%)	7/32 (22%)

		Original SIFT	PhonySIFT without outlier removal	PhonySIFT with outlier removal	Original Ferns	PhonyFerns without outlier removal	PhonyFerns with outlier removal
Tilt "Graffiti"	Image 1	1017/1030 (99%)	410/1493 (27%)	331/332 (100%)	54/55 (98%)	153/195 (78%)	90/93 (97%)
	Image 2	184/228 (81%)	256/1524 (17%)	199/201 (99%)	21/22 (95%)	77/179 (43%)	73/79 (92%)
	Image 3	14/21 (67%)	58/936 (6%)	4/10 (40%)	12/13 (92%)	35/162 (22%)	22/27 (81%)
	Image 4	0/14 (0%)	27/1109 (2%)	0/0 (0%)	3/199 (2%)	15/159 (9%)	14/21 (67%)
	Image 5	0/9 (0%)	4/1232 (0%)	0/21 (0%)	1/194 (1%)	0/144 (0%)	0/11 (0%)

		Original SIFT	PhonySIFT without outlier removal	PhonySIFT with outlier removal	Original Ferns	PhonyFerns without outlier removal	PhonyFerns with outlier removal
Brightness "Cars"	Image 1	1197/1212 (99%)	589/1515 (39%)	354/375 (94%)	124/126 (98%)	163/197 (83%)	112/117 (96%)
	Image 2	895/910 (98%)	478/1410 (34%)	347/350 (99%)	116/117 (99%)	161/192 (84%)	120/121 (99%)
	Image 3	651/661 (98%)	463/1449 (32%)	75/83 (90%)	102/104 (98%)	147/194 (76%)	101/106 (95%)
	Image 4	503/514 (98%)	305/1043 (29%)	243/254 (96%)	89/91 (98%)	143/193 (74%)	104/108 (96%)
	Image 5	336/349 (96%)	184/663 (28%)	151/158 (96%)	85/85 (100%)	136/190 (72%)	90/96 (94%)

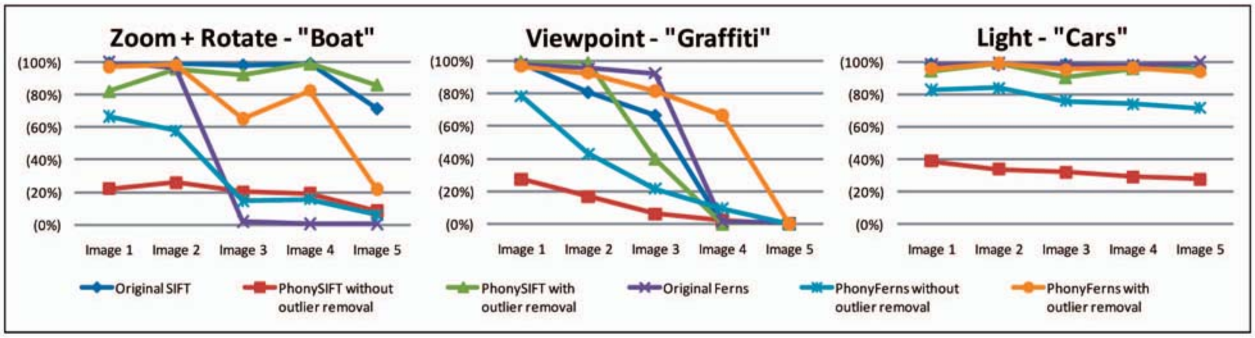

Fig. 4. Matching results for the three image sets of the Mikolajczyk framework that we used. For each test, the absolute number of inliers and matches as well as the percentages is reported.

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 in Fig. 3. The runtime performance seems the best for the middle configurations, while small S appears to suffer from the larger value of M, whereas for large S, the bad cache coherence of large histogram tables seems to impact performance.

5.2 Matching Rates

To estimate how our modifications affected the matching rates, we compared PhonySIFT and PhonyFerns against their original counterparts using images from the Mikolajczyk and Schmid framework.[1] We tested all four methods on three data sets (Zoom+rotation, Viewpoint, and Light) with one reference image and five test images each. The homographies provided with the data sets were used as ground truth. We allowed a maximum reprojection error of 5 pixels for correspondences to count as inliers. Although 5 pixels is a seemingly large error, our tests show that these errors can be handled effectively using an M-Estimator, while at the same time, the pose jitter is reduced due to a more stable set of inliers.

For each data set, we report the percentage of inliers of the original approach without any outlier removal, our approach without outlier removal, and our approach with outlier removal (see Fig. 4).

In the first data set, the original SIFT works very well for the first four images, while the matching rate suffers clearly in the fifth image. Although the matching rate of the PhonySIFT without outlier removal is rather low, with outlier removal, it is above 80 percent for all images and even surpasses the original SIFT for the final image. The matching rate of the original Ferns works very well on the first two images, but quickly becomes worse after that, while PhonyFerns works well except for the last image, where it breaks, because our training set was not created to allow for such high scale changes.

The second data set mostly tests tolerance to affine changes. Both the original and the modified versions (with outlier removal) work well for the first two images. The performance decreases considerably with the third image and only PhonyFerns is able to detect the fourth image. The third data set tests robustness to changes in lighting. All methods work very well on this data set.

The matching tests show a clear trend: The outlier rates of the modified methods are considerably higher than those of the original approaches. Yet, even very high numbers of outliers can be successfully filtered using our outlier removal techniques so that the modified approaches work at similar performance levels like the original approaches.

[1] http://www.robots.ox.ac.uk/~vgg/research/affine.

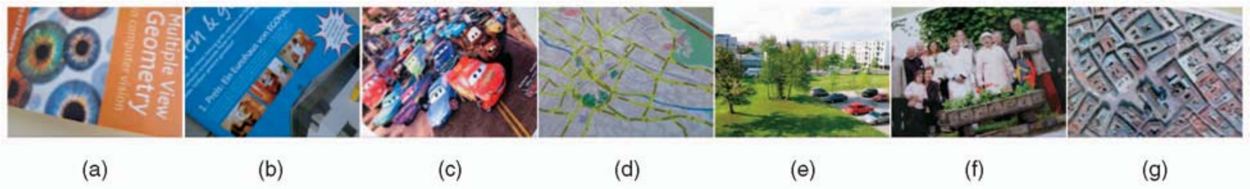

Fig. 5. The seven test sets (a)-(g): book cover, advertisement, cars movie poster, printed map, panorama picture, photo, and Vienna satellite image.

5.3 Tracking Targets

The optimized configurations for both PhonySIFT and PhonyFerns from the last sections were used to test robustness on seven different tracking targets (see Fig. 5) in stand-alone mode as well as in combination with the PatchTracker. The targets were selected to cover a range of different objects that might be of interest in real applications. We created test sequences for all targets at a resolution of 320×240 pixels. The sequences have a length of 501-1,081 frames. We applied all four combinations to all test sequences and measured the number of frames in which the pose was estimated successfully. 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.

As can be seen in Fig. 6, the Book and Cars data sets (first and third pictures in Fig. 5) performed worst. The Book cover consists of few, large characters and a low contrast, blurred image, making it hard for the keypoint detector to find keypoints over large areas. In the Cars data set, the sky and road are of low contrast, therefore, also respond badly to corner detection. Same as for the Book data set, these areas are hard to track with our current approaches.

The Advertisement, Map, and Panorama data sets show better suitability for tracking. Both the Advertisement and the Panorama consist of areas with few features, but they are better distributed over the whole target than in the Cars or Book targets. The Map target clearly has well-distributed features, but robustness suffers from the high frequency of these features, which create problems when searching at multiple scales. The Photo and Vienna data sets work noticeably better than the other targets because the features are well distributed, of high contrast and more unique than the features of the other data sets.

We therefore conclude that drawings and text are less suitable for our tracking approaches. They suffer from high frequencies, repetitive features, and typically few colors (shades). Probably, a contour-based approach is more suitable in such cases. Real objects or photos, on the other hand, have often features that are more distinct, but can suffer from poorly distributed features creating areas that are hard to track.

5.4 Tracking Robustness

Based on the Vienna data set, we created five different test sequences with varying number of frames at a resolution of 320×240 pixels, each showcasing a different practical situation: Sequence 1 resembles a smooth camera path, always pointing at the target (602 frames); Sequence 2 tests partial occlusion of a user interacting with the tracking target (1,134 frames). Sequence 3 checks how well the trackers work under strong tilt (782 frames). Sequence 4 imitates a user with fast camera movement as it is typical for mobile phone usage (928 frames). Finally, sequence 5 checks how well the trackers cope with pointing the camera away from and back to the target (601 frames).

All five sequences were tested with four different trackers: PhonySIFT, PhonyFerns, PatchTracker in combination with PhonySIFT (only for re/initialization), and PatchTracker in combination with PhonyFerns (only for re/initialization). The results of all tests are shown in Fig. 7. For each sequence and tracker, we coded the tracking success (defined as finding at least eight correspondences) as a horizontal line. The line is broken at those points in time, where tracking failed.

All four trackers are able to work very well with the "simple sequence." While PhonySIFT and PhonyFerns lose tracking for a few frames during the sequence, the PatchTracker takes over after the first frame and never loses it.

The four variants perform differently at the occlusion sequence, where large parts of the tracking target are covered by the user's hand. Here, both the PhonySIFT and the PhonyFerns tracker break. The PhonySIFT tracker works better because the PhonySIFT data set for this target contains more features, and it is therefore able to better find features in the small uncovered regions. The PatchTracker again takes over after the first frame and does not lose track over the complete sequence.

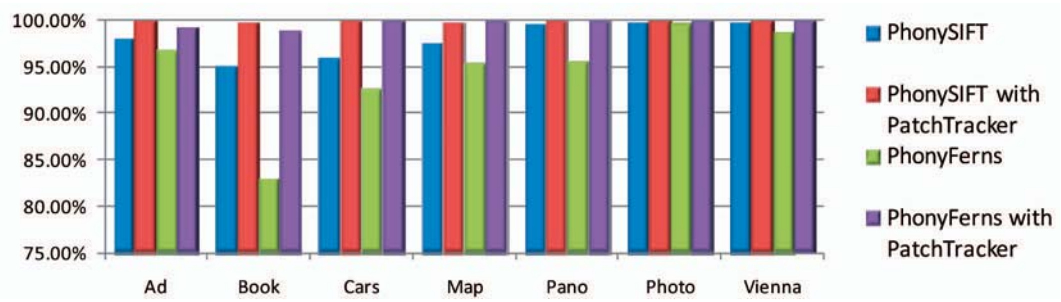

Fig. 6. Robustness results over different tracking targets.

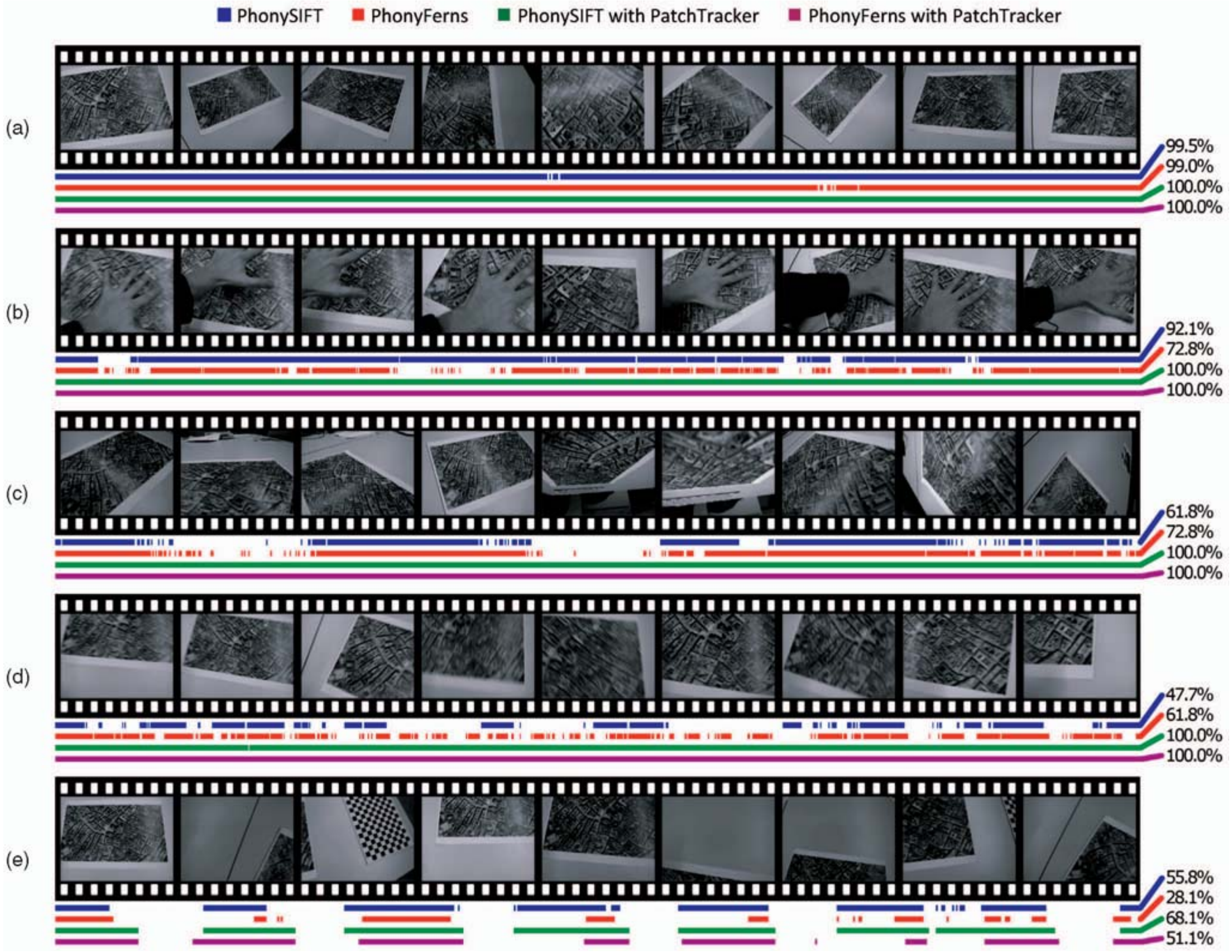

Fig. 7. Robustness tests of the four trackers on five test cases (a)-(e): (a) simple, (b) occlusion, (c) tilt, (d) fast movement, and (e) loss of target. The horizontal bars encode tracking success over time, defined as estimating a pose from at least eight keypoints. The reference image and test sequences can be downloaded from http://studierstube.org/handheld_ar/vienna_dataset.

Both PhonySIFT and PhonyFerns are known to have problems with strong tilts, which results from the fact that they were designed to tolerate tilt, but not actively take it into account. Generally, the PhonyFerns tracker does better than the PhonySIFT, which fits the expectations on these two methods. Since the PatchTracker directly copes with tilt, it does not run into any problems with this sequence.

The fast camera movements, and hence, strong motion blur of the fourth sequence create a severe problem for the FAST corner tracker used for both the PhonySIFT and the PhonyFerns trackers. The PhonyFerns tracker performs better because it automatically updates the threshold for corner detection, while the PhonySIFT tracker uses a constant threshold. By lowering the threshold, the PhonyFerns tracker is able to find more keypoints in the blurred frames than the PhonySIFT tracker does. The PatchTracker has no problems even with strong blur.

The last sequence tests coping with a target moving out of the camera's view and coming back in, hence, testing for tracking from small regions as well as fast reinitialization from an incomplete tracking target. In this sequence, the dynamic corner threshold becomes a weakness for the PhonyFerns tracker: The empty table has only very few features, making the PhonyFerns tracker to strongly decrease the threshold and requiring many frames to increase it again until it can successfully track a frame. Consequently, it takes the PhonyFerns tracker longer to find the target again than it does for the PhonySIFT tracker. The PatchTracker loses the target much later than PhonySIFT and PhonyFerns. The combined PatchTracker/PhonySIFT reinitializes exactly at the same time as the stand-alone PhonySIFT tracker. The PatchTracker/PhonyFerns combination behaves differently: Since the PatchTracker loses the target much later than PhonyFerns only does, the PhonyFerns part of the combined tracker has less frames for lowering the corner threshold too much, and therefore, reinitializes faster than when working alone.

Fig. 8 analyzes in depth, how well each tracker operates on the five test sequences. The left column of charts shows the distribution of reprojection errors in pixels for each tracker on successfully tracked frames, while the right column of charts shows the distribution of inliers per frames—including failed frames with 0 inliers. The reprojection error distribution shows that the PatchTracker combinations have the smallest reprojection errors with only the "Fast Movement" sequence producing significantly larger errors. However, on this sequence, the PatchTracker

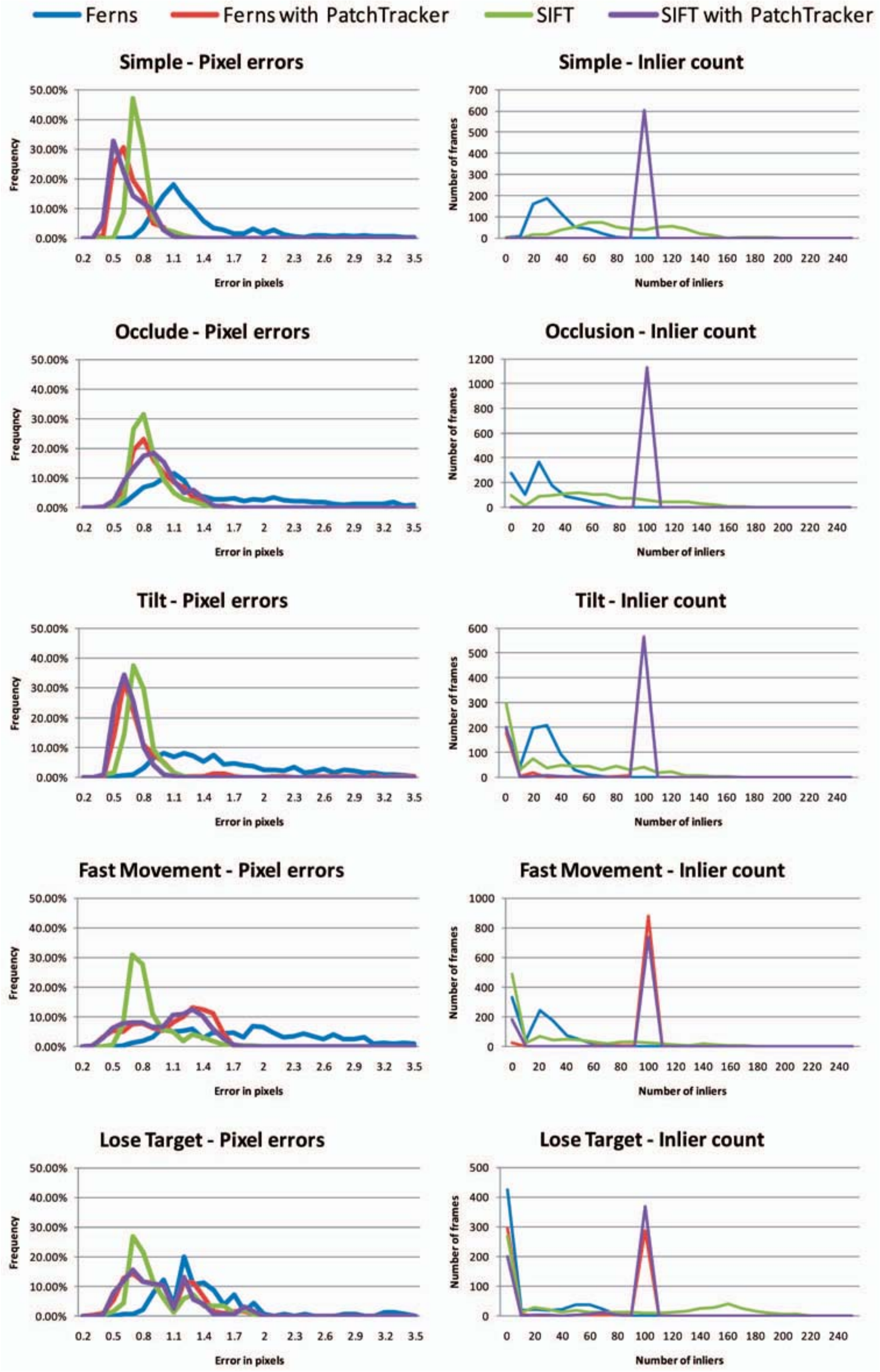

Fig. 8. Analysis of reprojection errors and inliers count for the five test sequences.

tracks many more frames successfully, even with reduced accuracy than the pure localization-based approaches as seen in the inlier distribution. The seemingly better behavior of the SIFT tracker comes from the fact that it did not track the difficult frames of this sequence, whereas the PatchTracker combinations continued to track at lower quality.

The Inlier count charts show that the PatchTracker combinations usually track at either full keypoint count (defined to be a maximum of 100) or not at all. Hence, for the "Simple," "Occlusion," and "Fast Movement" sequences, there is only a single peak at 100 inliers, whereas in the "Tilt" and "Lose Target," there is another peak at 0. Naturally, the maximum keypoint count per frame could be increased for the PatchTracker but would not change the picture drastically. The Ferns and SIFT trackers show different performances. Ferns tends to track much less points than SIFT, mostly due to its smaller data set, which was reduced to save memory. The larger number of

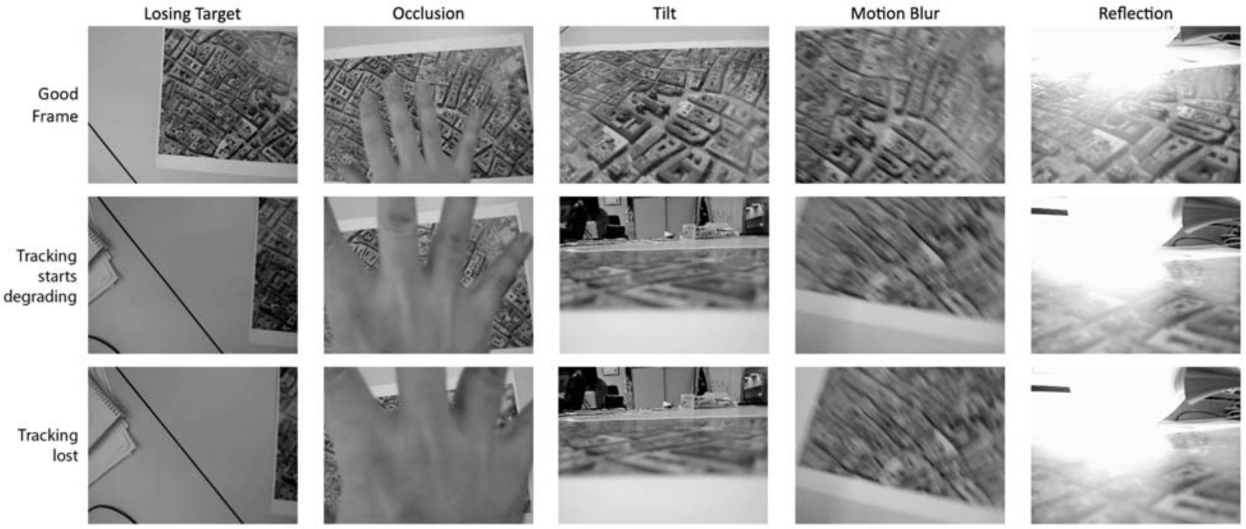

Fig. 9. Testing the PatchTracker against losing target, occlusion, tilt, motion blur, and reflections. The first image of each column shows a typical frame of the test sequence that was tracked well. The second image shows when the tracking quality starts to degrade. The third image shows the first frame that breaks tracking.

keypoints in the SIFT case lead to more stable poses, which was confirmed visually.

5.5 Breaking the PatchTracker

The robustness tests in the previous section showed that the PatchTracker strongly improves the overall tracking robustness of the other systems, resulting in almost 100 percent tracked frames in most of the tests. We therefore created an extra series of tests that was not designed to resemble practical situations, but instead, each image sequence makes the tracking increasingly more difficult allowing us to analyze when exactly the tracking breaks.

We created five different test sequences at a resolution of 320×240 pixels and a varying number of frames, each one looking at a different aspect. All tests were run with the PhonySIFT/PatchTracker combination. The sequences were designed so that the PhonySIFT tracker detects the pose in the first frame and then hands over tracking to the PatchTracker until it loses the target.

Fig. 9 shows a typical good frame for each test, the first frame when tracking degrades and the first frame that made the PatchTracker break. As before, the PatchTracker was configured to find 100 keypoints per frame. Degrading was defined as finding fewer than 100 keypoints.

The first sequence tests at which point the tracker loses the target that moves out of the screen. Tracking begins degrading when the target covers only 17 percent of the camera image ($\sim 63 \times 206$ pixels) and breaks when it goes down to 8 percent ($\sim 30 \times 204$ pixels). The second sequence tests partial occlusion of the tracking target. As can be seen in the 2nd column, tracking works well even under large occlusions and breaks only when the target is hardly visible anymore ($\sim$6 percent of the field of view).

The tilt test checks how strong the camera can be tilted with respect to the tracking target. Due to the affine warping of patches, very strong tilts can be tolerated and tracking works without degradation until it suddenly breaks. The motion blur sequence tests how fast the camera can be moved despite long exposure times, which is a typical problem of mobile phones, which are not very tolerant to low lighting conditions. Tracking degrades only when there is severe blur and breaks at a point when the target can be hardly recognized anymore (last image in fourth column). Finally, we estimated how well the NCC is able to compensate for reflections on the tracking target. The second image in the fifth column of Fig. 9 shows that tracking starts degrading at a point, where the camera image is already poor in contrast, and then loses the target quickly (last image in fifth column).

5.6 Performance

Finally, the overarching challenge of natural feature tracking on mobile phones is speed. To explore the operational speed of our approaches, we evaluated PhonySIFT, PhonyFerns, and both in combination with the PatchTracker on a mobile phone and a PC. As reported in [20], performance scales linearly with the CPU clock rate on mobile phones and is independent of the operating system. We therefore benchmarked on a single mobile phone only. The mobile we used is an Asus P552W, which has a Marvell PXA930 running at 624 MHz and a screen resolution of 240×320. The P552W does not have a floating point unit or hardware 3D acceleration. Hence, the trackers ran in fixed-point mode on this device. The PC is a Dell Notebook with an Intel Core 2 running at 2.5 GHz. Although the notebook's CPU has multiple cores, the trackers ran single-threaded only. On the PC, we activated the floating-point mode for all methods.

We tested all trackers against the "simple" sequence 1 of the robustness tests of Section 5.4, since we wanted to prevent tracking failures as much as possible to measure full frames only. During the first frame, several lookup tables are created. Some of these tables are for our fixed-point mathematics implementation (e.g., sine/cosine tables); others are specific to PhonySIFT. The PhonyFerns and the PatchTracker do not use custom lookup tables. Creating the fixed-point math tables takes 38 ms on the mobile phone and far below 0.1 ms on the PC (it is not possible to measure this reliably, since it is executed only once per application call). The PhonySIFT lookup tables require 121 ms to fill on the

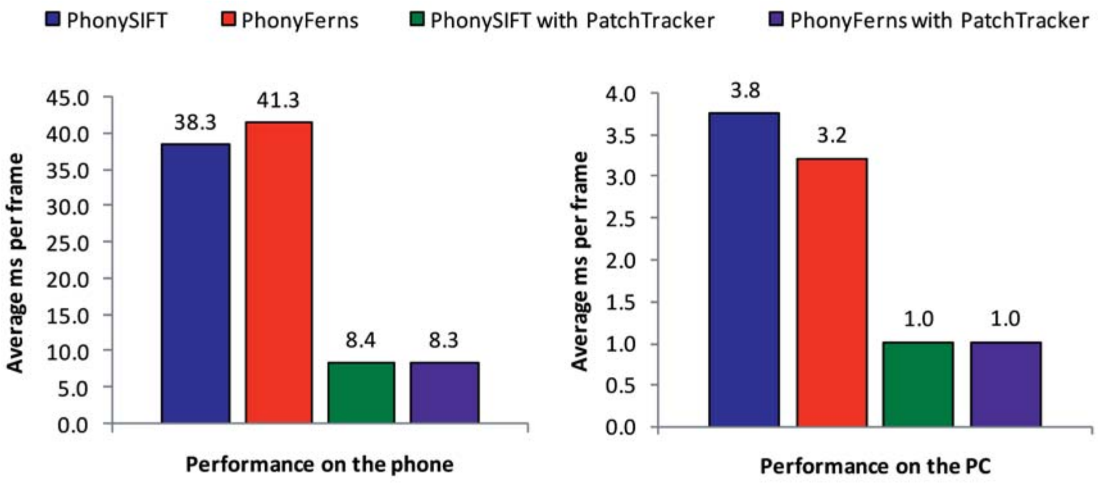

Fig. 10. Performance measurements on mobile phone and PC.

mobile phone and 0.3 ms on the PC. Fig. 10 shows the performance results on both mobile phone and PC. These measurements do not include the timings for the first frame.

The mobile phone runs both stand-alone versions of PhonySIFT and PhonyFerns in roughly 40 ms per frame. Adding typical overhead of AR applications (camera image retrieval, rendering, application overhead), this performance allows running an AR application at about 15 frames per second. Considering that the mobile phone we ran on the benchmarks represents the top of what is currently available, faster tracking is required to be sufficient for average devices. When activating the PatchTracker, the per frame time decreases to ~ 8 ms, reducing the performance requirements to a level that even average smart phones are capable of tracking at high frame rates. On the PC, PhonySIFT and PhonyFerns require 3.8 ms and 3.2 ms per frame when running in stand-alone mode. In combination with the PatchTracker, the tracking time goes down to 1 ms.

Comparing the per frame times of the mobile phone and the PC shows that even software that is highly optimized for mobile phones runs about 10 times slower on a high-end mobile phone than on an average single-core PC. Since all four methods could be made to run in multiple threads, the real performance gap between current mobile phones and PCs is even higher.

5.6.1 Detailed Speed Analysis for SIFT

Looking in more detail into what PhonySIFT spends its computation time for (see Fig. 11) on the mobile phone shows that the relatively simple task of corner detection requires ~ 14 percent of the overall time. This is not surprising since the corner detector has to look at almost all 76,800 pixels (320×240, except for those pixels close to the image border, where features cannot be described).

Feature describing and matching together require 74 percent of the overall time. It starts with describing the features, which includes blurring a patch, estimating its orientation, rotating to compensate for the orientation, and finally, creating one or more description vectors.

Outlier removal costs ~ 9 percent of the overall time per image. Most of the time is spent for creating and testing homographies, which is the last inlier test after orientation checks, removing duplicates and line tests. Finally, pose refinement takes ~ 3 percent frame time.

The timings on the PC are similar: Corner detection requires 20 percent. Same as on the mobile phone, describing and matching require 73 percent. Yet, outlier removal and pose refinement benefit from native floating point, requiring 4 and 3 percent only.

5.6.2 Detailed Speed Analysis for Ferns

The Ferns algorithm is simpler than the SIFT algorithm and consequently consists of only a few blocks (see Fig. 12). A set of operations is performed on the whole image consisting of corner detection (22 percent), downsampling (2 percent) to create a second octave, and blurring the input octave images (15 percent). The remaining time is spent in the classification (59 percent) which is linear both in number of interest points and classes, and finally, outlier detection (2 percent).

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.

5.6.3 Detailed Speed Analysis for PatchTracker

The PatchTracker is made up of three main blocks: downsampling, level 1, and level 0 estimations. Downsampling (using averaging) from 320×240 to 160×120 is a very fast operation and requires only 6 percent (0.5 ms) of the overall time on the mobile phone. Searching at level 1 takes 21 percent (1.6 ms) and pose estimation of these search

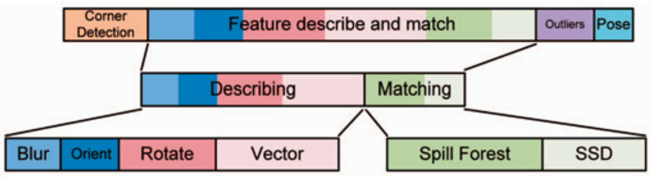

Fig. 11. Relative timings of the SIFT tracker on the mobile phone.

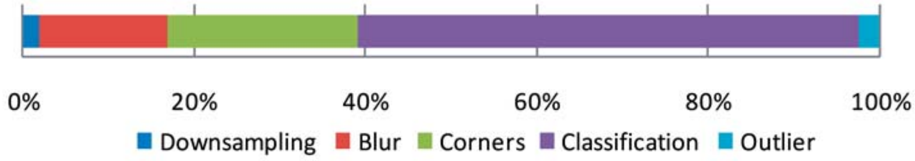

Fig. 12. Relative timings of the Ferns component.

results 22 percent (1.7 ms). The resulting pose is then used as starting point for level 0 search (34 percent, 2.6 ms) and pose estimation (17 percent, 1.3 ms). The level 1 pose is estimated from 20 points only compared to the 100 points at level 0. Still pose estimation at level 0 is faster, which is probably a result of the good starting pose forwarded from level 1.

Timings on the PC differ a bit from those on the mobile phone: Downsampling takes only 3 percent (0.03 ms), probably due to larger caches. Searching level 1 and level 0 takes the majority of time: 28 percent (0.27 ms) and 48 percent (0.46 ms), which is partially due to fixed-point usage on the PC. Pose estimation is very fast requiring only 7 percent (0.07 ms) and 13 percent (0.13 ms) again taking advantage of native floating point. Reasoning why the percentages for pose estimation are opposite to those on the mobile phone would require further investigation.

6 Conclusions and Future Work

We have presented three 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 PhonyFerns would let it outperform the more complex PhonySIFT 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. PhonyFerns adopted a regularization using the dominant orientation from SIFT, while PhonySIFT 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 (~40-50 degree) they can tolerate. This limit is strongly reduced by combining PhonySIFT and PhonyFerns with the PatchTracker, which can still track at close to 90 degree tilt.

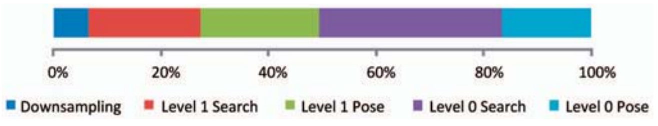

Fig. 13. Relative timings of the PatchTracker.

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 multicore phone CPUs very soon. This may enable more expensive per-pixel processing, allowing to reintroduce 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.

Acknowledgments

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, the European Union under contract FP6-2004-IST-4-27571 and the Christian Doppler Research Association.

References

[1] H. Bay, T. Tuytelaars, and L.V. Gool, “Surf: Speeded up Robust Features,” *Proc. European Conf. Computer Vision (ECCV)*, 2006.

[2] G. Bleser and D. Stricker, “Advanced Tracking through Efficient Image Processing and Visual-Inertial Sensor Fusion,” *Proc. IEEE Virtual Reality Conf. (VR ’08)*, pp. 137-144, 2008.

[3] W.-C. Chen, Y. Xiong, J. Gao, N. Gelfand, and R. Grzeszczuk, “Efficient Extraction of Robust Image Features on Mobile Devices,” *Proc. Int’l Symp. Mixed and Augmented Reality (ISMAR)*, 2007.

[4] O. Chum and J. Matas, “Matching with PROSAC—Progressive Sample Consensus,” *Proc. Conf. Computer Vision and Pattern Recognition (CVPR)*, 2005.

[5] A.J. Davison, W.W. Mayol, and D.W. Murray, “Real-Time Localisation and Mapping with Wearable Active Vision” *Proc. Int’l Symp. Mixed and Augmented Reality (ISMAR ’03)*, pp. 18-27, 2003.

[6] J. Gausemeier, J. Fruend, C. Matysczok, B. Bruederlin, and D. Beier, “Development of a Real Time Image Based Object Recognition Method for Mobile AR-Devices,” *Proc. Second Int’l Conf. Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa (Afrigraph ’03)*, pp. 133-139, 2003.

[7] A. Henrysson, M. Billinghurst, and M. Ollila, “Face to Face Collaborative AR on Mobile Phones,” *Proc. Int’l Symp. Mixed and Augmented Reality (ISMAR ’05)*, pp. 80-89, 2005.

[8] H. Hile and G. Borriello, "Information Overlay for Camera Phones in Indoor Environments," *Proc. Third Int'l Symp. Location- and Context-Awareness (LoCA '07),* pp. 68-84, 2007.

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

[10] V. Lepetit, P. Lagger, and P. Fua, "Randomized Trees for Real-Time Keypoint Recognition," *Proc. Conf. Computer Vision and Pattern Recognition (CVPR '05),* pp. 775-781, 2005.

[11] T. Liu, A.W. Moore, A. Gray, and K. Yang, "An Investigation of Practical Approximate Nearest Neighbor Algorithms" *Advances in Neural Information Processing Systems,* pp. 825-832, MIT Press, 2005.

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

[13] N. Molton, A. Davison, and I. Reid, "Locally Planar Patch Features for Real-Time Structure from Motion," *Proc. British Machine Vision Conf. (BMVC),* 2004.

[14] M. Ozuysal, P. Fua, and V. Lepetit, "Fast Keypoint Recognition in Ten Lines of Code," *Proc. Conf. Computer Vision and Pattern Recognition (CVPR '07),* pp. 1-8, 2007.

[15] E. Rosten and T. Drummond, "Machine Learning for High-Speed Corner Detection," *Proc. European Conf. Computer Vision (ECCV '06),* pp. 430-443, 2006.

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

[17] 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," *IEEE Trans. Pattern Analysis and Machine Intelligence,* 2008.

[18] D. Wagner and D. Schmalstieg, "ARToolKitPlus for Pose Tracking on Mobile Devices," *Proc. 12th Computer Vision Winter Workshop (CVWW '07),* pp. 139-146, 2007.

[19] D. Wagner and D. Schmalstieg, "First Steps Towards Handheld Augmented Reality," *Proc. Seventh Int'l Conf. Wearable Computers (ISWC '03),* pp. 127-135, 2003.

[20] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg, "Pose Tracking from Natural Features on Mobile Phones," *Proc. Int'l Symp. Mixed and Augmented Reality (ISMAR '08),* pp. 125-134, 2008.

[21] J. Wang, S. Zhai, and J. Canny, "Camera Phone Based Motion Sensing: Interaction Techniques, Applications and Performance Study," *Proc. ACM Symp. User Interface Software and Technology (UIST '06),* pp. 101-110, 2006.

[22] B. Zitova and J. Flusser, "Image Registration Methods: A Survey," *Image and Vision Computing,* vol. 21, no. 11, pp. 977-1000, Oct. 2003.

Daniel Wagner received the MSc degree from the Vienna University of Technology and the PhD degree from the Graz University of Technology. He is a postdoctoral researcher at the Graz University of Technology. During his studies, he worked as a software developer and joined Reality2, developing Virtual Reality software. After finishing his computer science studies, he was lead developer at Vienna-based game company BinaryBee, working on high-quality multiuser Internet games, as a software developer for Tisc Media, doing 3D engine development for a TV-show game, and as a consultant for Greentube's "Ski Challenge 2005." In 2006, he was a visiting researcher at HITLab, New Zealand. In October 2007, he finished his PhD thesis on handheld augmented reality. His current interest as a researcher at the Graz University of Technology is on mobile-augmented reality technology, including real-time graphics and computer vision for mobile phones. He is the leader of the Handheld Augmented Reality Group at the Graz University of Technology and deputy director of the Christian Doppler Laboratory for Handheld Augmented Reality. He is a member of the IEEE.

Gerhard Reitmayr received the Dipl-Ing degree in 2000 and Dr Techn degree in 2004 from the Vienna University of Technology. He is a professor of augmented reality at the Graz University of Technology. He worked as a research associate in the Department of Engineering at the University of Cambridge, United Kingdom, until May 2009, where he was a researcher in an industry-funded project and the principal investigator of two EC-funded projects, IPCity and Hydrosys. He is a member of the IEEE and the IEEE Computer Society.

Alessandro Mulloni received the BSc and MSc degrees in computer science, respectively, from the University of Milan and the University of Udine. He is working toward the PhD degree at the Graz University of Technology. His studies focused on 3D real-time graphics on handheld devices and human-computer interaction. During his studies, he worked as a researcher on various projects at the ITIA-CNR in Milan, inside the HCILab in Udine, and the Handheld Augmented Reality Group at the Graz University of Technology. He is currently working in the Christian Doppler Laboratory for handheld augmented reality, focusing the PhD research on user-centric design of interaction and visualization methods for handheld augmented reality. He is a student member of the IEEE.

Tom Drummond received the degree in mathematics from the University of Cambridge in 1988 and the PhD degree in computer science from the Curtin University of Technology, Perth, Western Australia, in 1998. He is currently a senior lecturer in the Department of Engineering at the University of Cambridge. His research there over the last 10 years has included work on real-time visual tracking, robust methods, reconstruction, SLAM, visually guided robotics, and augmented reality.

Dieter Schmalstieg received the Dipl-Ing, Dr Techn, and habilitation degrees from the Vienna University of Technology in 1993, 1997, and 2001, respectively. He is a full professor of virtual reality and computer graphics at the Graz University of Technology, Austria, where he directs the "Studierstube" research project on augmented reality. His current research interests include augmented reality, virtual reality, distributed graphics, 3D user interfaces, and ubiquitous computing. He is an author and coauthor of more than 100 reviewed scientific publications, a member of the editorial advisory board of computers and graphics, member of the steering committee of the IEEE International Symposium on Mixed and Augmented Reality, chair of the EUROGRAPHICS working group on Virtual Environments, advisor of the K-Plus Competence Center for Virtual Reality and Visualization in Vienna, deputy director of the Doctoral College for Confluence of Graphics and Vision, director of the Christian Doppler Laboratory for Handheld Augmented Reality, and member of the Austrian Academy of Science. In 2002, he received the START Career Award presented by the Austrian Science Fund. He is a member of the IEEE Computer Society.