Parallel Tracking and Mapping on a Camera Phone

Georg Klein*   David Murray†

Active Vision Laboratory
Department of Engineering Science
University of Oxford

Abstract

Camera phones are a promising platform for hand-held augmented reality. As their computational resources grow, they are becoming increasingly suitable for visual tracking tasks. At the same time, they still offer considerable challenges: Their cameras offer a narrow field-of-view not best suitable for robust tracking; images are often received at less than 15Hz; long exposure times result in significant motion blur; and finally, a rolling shutter causes severe smearing effects. This paper describes an attempt to implement a keyframe-based SLAM system on a camera phone (specifically, the Apple iPhone 3G). We describe a series of adaptations to the Parallel Tracking and Mapping system to mitigate the impact of the device's imaging deficiencies. Early results demonstrate a system capable of generating and augmenting small maps, albeit with reduced accuracy and robustness compared to SLAM on a PC.

1 Introduction

Modern mobile phones have become a compelling platform for mobile Augmented Reality. They contain all the equipment typically required for video-see-though AR: A computer with graphics acceleration, a camera, and a display. Even though the computational power available on phones still lags far behind desktop PCs, some researchers have already demonstrated that it is sufficient to perform real-time visual tracking [8, 10, 14, 15]. In particular, we are encouraged by [15]: Wagner *et al.* demonstrate that by carefully adapting, refining and combining established methods, phones are capable of extracting FAST corners, detecting SIFT / fern features, calculating a camera pose, and rendering augmented graphics at interactive frame-rates. The authors demonstrate their results on known planar targets, but there is little reason to doubt the method could be extended to a larger suitably textured 3D scene – as long as a map of salient points is available *a priori*.

In this paper we consider the task of tracking a phone's position in a previously *unknown* environment. In particular, we attempt to implement a keyframe-based SLAM system on the Apple iPhone 3G. Previous experience with the keyframe-based PTAM system [6, 7] on the PC has shown that it is able to generate room-sized maps and track a moving camera, while (crucially) offering a high level robustness to rapid motion, poor imaging conditions, and inevitable data association failures. We adopt this system as a starting point. The iPhone is selected as representative of the current generation of camera-equipped smart phones. Other devices may have faster CPUs and cameras, but the iPhone's form-factor and widespread deployment make it an interesting platform for AR applications.

As a platform for PTAM, the iPhone offers two main challenges. The lack of processing power is obvious, and particularly significant when PTAM on the PC derives a large part of its robustness from a somewhat brute-force approach to mapping. The second (and perhaps more interesting) challenge comes from the device's camera, which differs in three ways from the cameras commonly used for SLAM. It operates at a low ($<$15Hz) frame-rate; it has a rolling shutter; and most importantly, it has a narrow field-of-view. Each of these effects individually would be problematic, but when all three are present they reinforce one another and conspire to produce video images poorly suited to 3D reconstruction.

After a brief review of related mobile tracking work in Section 2, this paper describes a series of changes to the PTAM algorithm which mitigate the impact of the above imaging and processing limitations. Among the contributions described in Sections 3–6 are new strategies for automatic stereo initialisation, feature detection, rolling shutter compensation, and map pruning. Preliminary results, presented in Section 7, show that the PTAM on the iPhone is capable of creating and tracking small maps. Conclusions and likely areas of future work are presented in Section 8.

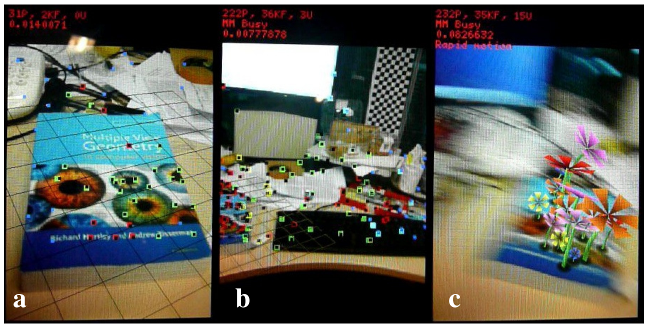

Figure 1: PTAM running on an iPhone 3G. A map is started on a book (a), expanded to circa 220 points (b), and then used to insert virtual flowers into the scene (c).

2 Related Work

Besides the natural feature tracking system of [15], marker-based systems have also been demonstrated on phones. A recent example attempting to reduce the area taken up by the physical marker is [14], but demonstrations of mobile marker tracking go back to [16] in which ARToolkit markers were detected on a PDA and [8] who develop their own marker on an early camera phone. (Today FPUs, OpenGL acceleration and faster CPUs have greatly reduced the engineering effort required to develop AR systems on phones.) A marker/texture hybrid system is demonstrated in [10], where the tracking of a known map texture is assisted by adding a grid of black dots to the map.

3 Managing the cost of Bundle Adjustment

We adopt the basic framework of the PTAM [6] where processing is performed on two threads. A foreground thread handles frame-to-frame tracking and camera registration, while a background thread performs bundle adjustment to optimize the map. This arrangement works despite the iPhone having only a single processor, as its OS respects requested thread priorities — bundle adjustment only runs while the tracking thread is waiting for a new frame.

*e-mail: gk@robots.ox.ac.uk
†e-mail:dwm@robots.ox.ac.uk

IEEE International Symposium on Mixed and Augmented Reality 2009
Science and Technology Proceedings
19 -22 October, Orlando, Florida, USA
978-1-4244-5419-8/09/$25.00 ©2009 IEEE

We find that the 412MHz CPU on the iPhone, although floating-point enabled, is a factor of 15-30$\times$ slower than a single core of an Intel E6700 CPU. Considering that PTAM on the PC dedicates a processing core to bundle adjustment alone, it is clear that some compromises need to be made to perform bundle adjustment on the phone: A numerical reduction in map size is required.

The scalability of bundle adjustment is detailed in [3]. Given a map of N points and M keyframes, with each point observed by T keyframes, different parts of bundle adjustment scale with different quantities: projections, differentials and back substitution cost $O(NT)$; generating the reduced system costs $O(NT^2)$; solving the reduced system costs $O(M^3)$. Here we reduce each of N, M, T.

3.1 Fewer (but multi-scale) map points

PTAM on the PC derives its tracking robustness in part from the use of a large number of point measurements per frame. To generate such a number of points, every possible landmark is inserted into the map: For each of four image pyramid levels, every detected FAST [11] corner which can be triangulated is added. This can be wasteful, as a single physical point may have map duplicates at a number of scales. We use a different approach here, and require that features be corners across multiple scale levels.

Images from the iPhone are received at a resolution of 240$\times$320 pixels, from which five pyramid levels (up to L4 at 15$\times$20 pixels) are generated. We require that new features be valid Shi-Tomasi (ST) corners [12] in at least all of L0,L1,L2. This requirement allows efficient corner extraction, since exhaustive ST-corner extraction is only required in the (relatively small) L2 image. Each maximal corner is projected down to the L1 image, where the immediate local neighborhood is searched for the best ST score. This search proceeds by iterative ascent, with the best pixel in a 3$\times$3 window becoming the next window center, until a maximum is found or three iterations are exceeded. If a good corner is found at L1, the same procedure is repeated in the L0 image, and the L0 position is used as the position of the detected corner.

3.2 Measurement culling

For the map sizes encountered by PTAM, the $O(NT^2)$ outer product cost often dominates bundle adjustment. When running on a PC, operating with large T (measurements per map point) is acceptable; in fact, the background thread actively seeks to increase the density of measurements by re-searching for every map point in every keyframe. On the iPhone we take the opposite approach. Not only do we omit the map-densification steps, we actively *reduce* the number of measurements taken of each point, keeping only the most useful measurements of that point.

For a point $\boldsymbol{p}$, bundle adjustment builds the information matrix V (we adopt the notation of [5], Appendix 6.6):

$$V = \sum_j V_j = \sum_j w_j B_j^T \Sigma_j^{-1} B_j$$

where B_j is the point's 2$\times$3 projection Jacobian (w.r.t. point position) in the jth keyframe, with corresponding measurement noise Σ_j^{-1} and w_j is the weight returned by the Tukey M-Estimator. The determinant $|V|$ is a metric corresponding to the amount of information constraining the position of the point (given known camera poses). We consider for each keyframe the metric $m_j = |V| - |V - V_j|$: This corresponds to information which would be lost were this keyframe's measurement removed. A low value indicates a measurement which is redundant: Either the point is viewed from an angle from which it is already well observed, or measurement noise is high, or the measurement may be classed an outlier. Whenever a point is observed simultaneously by more than 8 keyframes, we discard all measurements beyond the 8th most useful.

3.3 Keyframe culling

On the iPhone, the $O(M^3)$ cost of solving the reduced camera system rapidly becomes prohibitive to the point that bundle adjustment does not converge: The number of keyframes in bundle adjustment must be kept low. Unfortunately, the narrow field-of-view of the iPhone's camera, coupled with a portrait form factor which reduces horizontal FOV even further, mandates the use of many keyframes when exploring (so that new features are initialised before old ones go out of view). While we cannot avoid adding keyframes to the map densely, we can instead eliminate keyframes later on.

Once the number of keyframes in the map exceeds a threshold (M>35 on the iPhone), the bundle adjustment thread attempts to find redundant keyframes to erase. This might be a keyframe which observes some portion of the map which is already well-constrained by some other keyframes which provide a better baseline. Keyframes are ranked by a voting system, whereby map points vote for keyframes they consider useful. If a keyframe j is one of only two which observe a map point, it receives a utility score of $u_j = 0.5$ from that point. If more than two keyframes observe a point, a point's score contribution to each keyframe depends on the keyframe's rank according to the information metric m_j above:

$$u_j = \frac{\hat{o} - (rank(j) - 1)}{\sum_{i=1}^{\hat{o}} i}, \quad \hat{o} = min(o, 4)$$

where o is the number of keyframes which observe the point. The keyframes with the lowest sum utility score (which must also be lower than a fixed threshold) are erased. [1]

4 Tracking

The iPhone's combination of narrow field-of-view and low frame-rate (with exposure times approaching 60ms) results in blurry images as soon as the device is moved at any speed. This is somewhat mitigated by the fact that the screen is also on the device, so the user will generally move it gently to be able to see the screen — but even so, motion blur is a frequent occurrence and must be addressed.

Our previous work on the PC [7] proposed two methods for tracking blurred frames: the addition of edges to the map, and full-frame rotation estimation. Edges are more blur-resistant than point features, but the computational cost of adding them to the map is prohibitive on the iPhone. On the other hand full-frame rotation estimation, which aligns successive video frames by direct 3-DOF efficient second-order (ESM [2]) alignment on the coarsest (15$\times$20-pixel) pyramid level, is very fast and we use this method here. This effectively acts as a gyroscope, which the iPhone physically lacks.

PTAM on the PC tracks point features in two stages: Initially, large features are found in coarse levels of the image pyramid; after a pose update calculated from these, more features are searched in lower (finer) levels of the image pyramid, with a tighter search radius. Additionally, feature searches are performed only around FAST corners. This strategy allows a large number ($\sim$1000) of features to be measured, ensuring accurate tracking.

On the iPhone — where the map contains far fewer features, blur is a constant problem, and CPU bandwidth is limited — this does not work. Here we cannot rely on corner responses, and the FAST detector is not used. We omit the fine tracking stage altogether: Feature point searches are performed in the L1 (120$\times$160) image, by exhaustive evaluation of a 4-pixel-radius circle using zero-normalised SSD against an 8$\times$8 template. To compensate the loss of measurement precision at this coarse resolution, each match is

[1] Erased keyframes are retained in memory, and their positions occasionally updated to track changes in the map, for three purposes: First to provide pixels for point features created in them; second to prevent new keyframes being created in the exact same position; and third for relocalisation with the keyframe-based method of [7].

subpixel-refined by 2-DOF ESM alignment. This strategy is made possible by the multi-scale corner detection described in §3.1: Instead of always sourcing template pixels from the same pyramid levels and varying the search level (as in [6]), here we can fix search level and vary the source level (as in [17]) to compensate changes in scale as the camera moves, and still be assured a trackable template.

When the camera is rotating quickly even the L1 image becomes too blurred. When this is predicted by the full-frame initialiser, searches are performed only in the L2 image, and the error threshold for a successful template match is reduced. This strategy allows the system to track moderately fast rotations, but frames tracked in this way are inaccurate, and never used as keyframes.

5 Initialisation

5.1 Simplified stereo initialisation from a planar scene

PTAM on the PC must be bootstrapped by the provision of an initial stereo pair, along with point correspondences, from which an initial map is generated either using five-point-pose [13] or, in later versions of the system, a homography decomposition [4]. We have since found that requiring users to provide two initial keyframes is problematic, as some will not understand stereo baseline requirement and attempt to initialise the system using pure rotation. A single-click procedure whereby the user needs only provide the first keyframe would be preferable. Further, [6]'s stereo initialisation is fragile in that it requires long uninterrupted feature tracks (difficult with the iPhone's camera); it has a RANSAC stage for choosing a best-fit homography (slow on the iPhone); and it sometimes fails to disambiguate between two possible plane/motion hypotheses, resulting in an incorrect initialisation (the iPhone's narrow field-of-view exacerbates this effect, as it sees fewer disambiguating points).

Here we propose a more robust, two-stage initialisation method. We retain the assumption that the user is initially viewing a planar scene (this will form the ground-plane for future augmentations). In the first stage, the user presses the screen to initialise the first keyframe; corners are extracted from this, un-projected to the $z=1$-plane and added to the map as features.

In subsequent frames, the user moves the camera, and the map points are tracked in a procedure similar to §4 — but instead of tracking a 3-DOF camera pose $(R,\boldsymbol{t})\in SE(3)$, points are projected into the viewing frustum by a homography $H\in SL(3)$, which is updated from each frame's measurements. The use of a homography to project points allows robust tracking by the use of patch warping and full-frame rotation estimation, and eliminates the need for individually continuous feature tracks. A robust M-estimator reduces the effect of out-of-plane points on this estimation.

During every frame of this first stage, H is decomposed into (usually two) hypotheses for 3D camera pose $(R,\boldsymbol{t})$ and plane normal $\hat{\boldsymbol{n}}$ as per [4]. Initially, when the camera is very close to its start pose, $\hat{\boldsymbol{n}}$ is poorly constrained as any plane normal would satisfy the projection constraints. The map is therefore tracked by H until the camera has moved far enough to constrain the decomposition well. We use the condition number (the ratio of minimum to maximum eigenvalues) of the information matrix J^TJ as a quality metric, where J is the Jacobian matrix of partial derivatives of each point's projection with respect to 8-DOF changes to the decomposition (3+3+2 DOF respectively for changes to R, $\boldsymbol{t}$ and $\hat{\boldsymbol{n}}$). This metric considers not only the value of H but also the image distribution and noise of the feature measurement on which it is based.[2]

Once the decomposition(s) has been suitably constrained, a two-fold ambiguity often remains. If this is the case, a separate 3D map is generated for each hypothesis or $(R,\boldsymbol{t})$, with point positions found by triangulation. In the second stage, the tracker tracks a separate $SE(3)$ camera pose estimate for both 3D maps at every frame, until the objective function (the robustified sum-squared reprojection error) for one case is twice as large as the other: At this point, the hypothesis with the larger error is eliminated, and a single 3D map remains. This now has two (or three, if disambiguation was necessary) keyframes on which bundle adjustment is performed to complete the initial map.

[2]We note that an H-only metric is used successfully in [17] for initialising a per-patch plane orientation EKF.

5.2 Adding points to the map

One of the advantages of the above method on the iPhone is that it spreads computing effort over multiple frames: Instead of a single expensive RANSAC homography fit at the moment the second view is added, the homography is updated incrementally over the intermediate frames. A similar strategy can also be applied to the insertion of new points to the map. We replace the epipolar search of [6] (which delays the appearance of new points on the iPhone) with the concept of partially initialised points which is common in incremental monocular SLAM.

Whenever a keyframe is added to the map, corner detection as described in §3.1 is performed, and up to 25 detected corners which are sufficiently far away from existing features are added to the map. The position of each new point (in the coordinates of the new keyframe $\mathcal{K}$) is

$$\boldsymbol{p}_{\mathcal{K}} = \frac{1}{q}\begin{pmatrix} x & y & 1 \end{pmatrix}^T, \qquad q \sim N(\hat{q},\sigma)$$

where x, y are the un-projected corner coordinates. The inverse depth estimate $\hat{q}$ is initially set to the observed scene's mean inverse depth and assigned an initial uncertainty $\sigma=\infty$.

In subsequent frames, once the tracker has determined camera pose, partially initialised features are projected into the image using their inverse depth estimate $\hat{q}$, and the L1-image is searched for the feature. For any successful measurement, the reprojection error along the epipolar line is used to update $\hat{q}$ and σ by the use of an extended information filter. We consider $\hat{q}$ to be the only free variable, with cameras and x, y assumed perfectly known.

When any subsequent keyframe is added, all partially initialised points are projected into the image. Any point successfully measured is re-triangulated against the original keyframe and immediately inserted into the map with standard Cartesian coordinates. This means that in contrast to most MonoSLAM systems (such as [9], which this approach resembles) the inverse-depth information filter is used purely for data association: Any information gathered from the intermediate frames is discarded once a second keyframe observes the point — as is any bias introduced by the false assumptions of independence and perfect camera poses, which make the method computationally cheap.

6 Rolling Shutter Compensation

The iPhone uses a rolling shutter to the effect that the left edge of the video image records the scene ∼60ms after the right edge. This unfortunate byproduct of the CMOS sensor introduces bias into almost every step of a SLAM system (where, typically, the assumption of a global shutter is made). While some authors have demonstrated that a rolling shutter can be exploited to directly measure velocity from a single image [1], this was done in the context of fast high-res cameras, small accelerations, and known models; here, we merely aim to reduce the negative impact of the rolling shutter on map accuracy. This is done by applying a rolling shutter correction to all image measurements used in bundle adjustment.

Each keyframe has a list of 2D image measurements $\boldsymbol{m} = (u\, v)^T$. We store also a list of measurements made in the preceding and following video frames, from which a set 2D image velocities $\dot{\boldsymbol{m}} = (\dot{u}\, \dot{v})^T$ can be derived by assuming constant image velocity. These velocities are noisy and incomplete (as some points may not be measured in all three successive frames) so they are not used directly as corrections; instead, the instantaneous camera velocity $\dot{\boldsymbol{\mu}}$

during the keyframe is estimated as the least-squares solution of

$$\begin{bmatrix} J_1 \\ \vdots \\ J_n \end{bmatrix} \dot{\boldsymbol{\mu}} = \begin{pmatrix} \dot{\boldsymbol{m}}_1 \\ \vdots \\ \dot{\boldsymbol{m}}_n \end{pmatrix}$$

where J_i is the 2×6 image projection Jacobian of the ith point with respect to keyframe camera pose. The estimated camera velocity is then used to generate the corrected point measurements $\hat{\boldsymbol{m}}_i = \boldsymbol{m}_i - J_i\dot{\boldsymbol{\mu}}\delta t_i$, where $\delta t_i = f(u_i)$ is the time offset (relative to the center of the image) at which image column u_i was imaged.

The above method is crude in that it uses a constant velocity model and keeps velocity estimation outside of bundle adjustment — however it is only intended to provide a rough, first-order correction of measurements, and any more advanced technique would carry a substantially higher computational cost. As the map changes, measurements are occasionally re-corrected, but this is done at a far lower frequency than bundle adjustment.

7 Results and Limitations

Video results (also shown in Figure 1) of the system described above are included as an attachment, and demonstrate PTAM on the iPhone generating, tracking, and augmenting small maps in both indoor and outdoor settings. The system is mostly able to cope with the difficult images received (especially indoors).

A typical video frame is processed in 30ms. 5ms are required for BGRA-to-Y conversion and image pyramid generation; 1.5ms for rotation estimation; 17.5ms to warp and search for 50 point features; 2.5ms to calculate a pose update; and 3.5ms to find and update 12 partially initialised features. The remaining 36ms per frame are used for image capture, rendering overheads and bundle adjustment. Corner detection requires 10ms when a keyframe is added.

A single iteration of bundle adjustment with 35 keyframes and a map of 300 points completes after approximately 750ms — however the number of iterations required for convergence varies from a few to dozens. The culling procedure outlined in §3 requires circa 50ms to decide which measurements and keyframes to erase, but is successful in keeping the cost of bundle adjustment in check. It is possible for this method to erase a keyframe such that the map is split in half and corrupted — but this is rare, and has only been observed when actively trying to map too-large areas.

In our tests, rolling shutter compensation consistently reduces the total objective function (robustified sum of squared reprojection errors) of bundle adjustment. Error can be reduced by as much as 50% when observing test patterns such as checkerboards, but in more general scenes a reduction by 10% to 20% is the norm.

In a direct subjective comparison to PTAM on the PC with a wide-angle camera, the phone-based system is far less accurate and robust. Pose jitter for identically placed stationary cameras is a factor of 30 higher on the phone. The difference in speed at which a map can be expanded is also notable; the system will refuse to insert new keyframes if camera pose is poorly constrained, which happens frequently with the sparser maps generated on the phone. Finally, the system here requires scenes with substantial quantities of texture which satisfies the multi-scale corner detector, whereas PTAM on the PC is more flexible in this regard.

8 Conclusions and Further Work

This paper has demonstrated that keyframe-based SLAM can operate on mobile phones. Although the accuracy and scope of the system may be limited compared to what is possible on a PC, the iPhone is nevertheless able to generate and augment small maps in real time and at full frame-rate. Tracking will likely become more accurate on future phones with faster CPUs and 30Hz cameras — we may hope that wider fields-of-view become available too.

A number of interesting avenues for further research have emerged in the course of this work. Here information is discarded to keep bundle adjustment computationally manageable — but there are many alternatives to this. One would be to simply declare the map full and stop adding new information; another would be to marginalize parameters out of the estimation, or declare some map points well enough constrained that they can be regarded as fixed. Perhaps a work-flow by which the user initially maps out a workspace and then locks the map for tracking would be acceptable.

Finally, this work has discussed the iPhone's limitations vis-à-vis the PC, but has not attempted to exploit its extra capabilities. The iPhone's camera can take high-resolution stills, the phone contains an accelerometer and a GPS receiver, and future devices will contain magnetic compasses and gyroscopes. These extra capabilities could surely be used to benefit visual tracking and AR.

References

[1] O. Ait-Aider, N. Andreff, J.-M. Lavest, and P. Martinet. Simultaneous object pose and velocity computation using a single view from a rolling shutter camera. In *9th European Conference on Computer Vision (ECCV 2006)*, pages 56–68, May 2006.

[2] S. Benhimane and E. Malis. Homography-based 2d visual tracking and servoing. *Special Joint Issue on Robotics and Vision. Journal of Robotics Research*, 26(7):661–676, July 2007.

[3] C. Engels, H. Stewénius, and D. Nistér. Bundle adjustment rules. In *Photogrammetric Computer Vision (PCV'06)*, 2006.

[4] O. Faugeras and F. Lustman. Motion and structure from motion in a piecewise planar environment. *International Journal of Pattern Recognition and Artificial Intelligence*, 2(3):485–508, 1988.

[5] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[6] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *Proc 6th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, October 2007.

[7] G. Klein and D. Murray. Improving the agility of keyframe-based SLAM. In *Proc. 10th European Conference on Computer Vision (ECCV'08)*, 2008.

[8] M. Möhring, C. Lessig, and O. Bimber. Video see-through AR on consumer cell-phones. In *Proc 3rd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'04)*, pages 252–253, Arlington, VA, November 2004.

[9] T. Pietzsch. Efficient feature parameterisation for visual slam using inverse depth bundles. In *Proc. British Machine Vision Conf.*, 2008.

[10] M. Rohs, J. Schoening, A. Krueger, and B. Hecht. Towards real-time markerless tracking of magic lenses on paper maps. In *Adjunct Proc. 5th International Conference on Pervasive Computing*, 2007.

[11] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *Proc 9th European Conference on Computer Vision (ECCV'06)*, May 2006.

[12] J. Shi and C. Tomasi. Good features to track. In *Proc IEEE Intl. Conference on Computer Vision and Pattern Recognition (CVPR '94)*, pages 593–600. IEEE Computer Society, 1994.

[13] H. Stewénius, C. Engels, and D. Nistér. Recent developments on direct relative orientation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 60:284–294, June 2006.

[14] D. Wagner, T. Langlotz, and D. Schmalstieg. Robust and unobtrusive marker tracking on mobile phones. In *Proc 7th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'08)*, Sept. 15–18 2008.

[15] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg. Pose tracking from natural features on mobile phones. In *Proc 7th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'08)*, Sept. 15–18 2008.

[16] D. Wagner and D. Schmalstieg. First steps towards handheld augmented reality. In *7th Intl. Symposium on Wearable Computers (ISWC'03)*, pages 127–137, White Plains, NY, October 2003.

[17] H. Wuest, F. Wientapper, and D. Stricker. Acquisition of high quality planar patch features. In *Advances in Visual Computing, 4th International Symposium (ISVC'08)*, 2008.