Robust and Unobtrusive Marker Tracking on Mobile Phones

Daniel Wagner[1], Tobias Langlotz[2], Dieter Schmalstieg[3]

Graz University of Technology

ABSTRACT

Marker tracking has revolutionized Augmented Reality about a decade ago. However, this revolution came at the expense of visual clutter. In this paper, we propose several new marker techniques, which are less obtrusive than the usual black and white squares. Furthermore, we report methods that allow tracking beyond the visibility of these markers further improving robustness. All presented techniques are implemented in a single tracking library, are highly efficient in their memory and CPU usage and run at interactive frame rates on mobile phones.

KEYWORDS: marker tracking, pixel flow, mobile phones

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

1 INTRODUCTION

Although there has been much work on Augmented Reality (AR) tracking from natural features, these techniques are commonly less robust and require much more processing resources than tracking from markers. In particular, when using mobile phones as a platform for AR, computing power is an order of magnitude smaller than on desktop computers making marker-based tracking solutions the best trade-off between computational feasibility and robustness. Moreover, markers containing digital barcode patterns can not only be used for pose tracking, but also to uniquely distinguish thousands of objects or even provide unique pointers to online resources such as web pages or 3D content to be displayed on the phone. Providing the equivalent capabilities from purely natural features would require not only implementing a pose tracking system, but also a reliable object detection system, all under stringent real-time constraints.

We were therefore motivated to extend our previous work on marker-based tracking for mobile phones [12] with new features that are designed to overcome the most severe limitations of previous approaches, without sacrificing the robustness and overall low computational complexity. Specifically, we describe three new marker designs that occupy significantly less space and therefore reduce the amount of visual pollution in the augmented area.

We also describe two computationally inexpensive techniques based on feature following and pixel flow, which can be used for incremental tracking in cases where the marker is partially occluded or out of sight. Together, space-economic marker designs and incremental tracking allow placing markers in situations that were previously not really feasible, or at least very cumbersome to instrument. All techniques have been implemented to run in real time on current mobile phones and can be combined to make the use of markers significantly more flexible and less painful.

[1]e-mail: wagner@icg.tugraz.at
[2]e-mail: langlotz@icg.tugraz.at
[3]e-mail: schmalstieg@icg.tugraz.at

2 RELATED WORK

Probably the first marker tracker developed for AR was Rekimoto's Matrix Code [4]. It pioneered the use of square planar shapes for pose estimation and embedded 2D barcode patterns for distinguishing markers. Later Kato used a similar approach in ARToolKit [2], which was released as open source and consequently became enormously popular among AR researchers and enthusiasts alike. Since then, many similar systems emerged, of which Fiala's ARTag [1] is most well known.

Compared to the vast number of marker tracking systems available on desktop computers, only few solutions for mobile phones have been reported in literature. In 2003 our group ported ARToolKit to Windows CE and thus created the first self-contained AR application [10] on an off-the-shelf embedded device. This port later evolved into the ARToolKitPlus tracking library [9]. In 2004, Möhring [3] created a tracking solution for mobile phones that tracks color-coded 3D marker shapes. Around the same time, Rohs created the VisualCodes system for smartphones [5]. Both Möhring's as well as Rohs' techniques provide only simple tracking of 2D position on the screen, 1D rotation and a very coarse distance measure. In 2007, Rohs created a software for Symbian phones that tracks maps, which are outfitted with regular grids of dots, again tracked with 2.5 DOF [6]. The dot markers, presented in section 3.3 are similar, but provide full 6DOF tracking.

3 UNOBTRUSIVE MARKER TRACKING

Albeit still popular, the techniques used in the original ARToolKit [2] have become dated, as new, more efficient techniques are being developed. We therefore stopped the work on ARToolKitPlus [9] and started developing *Studierstube Tracker*, a new marker tracking library developed from scratch to optimally support mobile phones [12].

Studierstube Tracker currently supports 6 different marker types (including those 3 described in this paper), 2 different pose estimators and 3 different thresholding algorithms, that all have their specific strengths and weaknesses. Memory requirements are one order of magnitude lower than with ARToolKitPlus and are typically in the range of 150Kbyte.

Studierstube Tracker supports digitally encoded ids with forward error correction (Bose/Chaudhuri/Hocquenghem) in the style of ARTag, but has more flexibility in the structure and layout of the digital code. This allows to encode a large amount of information – for this purpose, Studierstube Tracker supports the DataMatrix barcode standard (ISO/IEC16022), which can store up to 2KB of data. If the marker must encode only a few bits, it is sensible to reduce the area covered by the marker, leaving a larger portion of the interaction space untouched.

Three designs for such less obtrusive markers, frame markers, split markers and dot markers are presented in this section, while

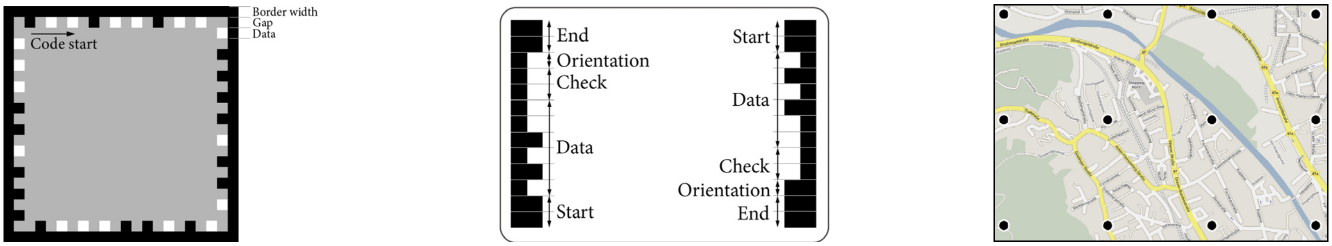

Figure 1. frame markers (left), split markers (middle), and map with dot markers (right)

the next sections explain how tracking can be continued incrementally if the marker is lost.

3.1 Frame markers

Robustness of marker tracking is largely owed to the high contrast afforded by the black frame in a thresholded image. The frame itself is not disturbing in many situations, if the interior can be filled with application specific artwork, like a framed painting. With *frame markers* we therefore take the approach of encoding a digital id with error correction at the interior side of the frame, making it appear like a frame decoration (see left image in Figure 1 and 1st picture in Figure 2). Frame markers do not require any interior at all and can therefore be put around existing flat objects such as pictures on a wall.

A 36 bit code including 27 bit redundancy is encoded alongside each of the 4 sides of the frame, in the form of 9 individual black or white squares encoding 1 bit each (9bit = 512 different markers). The code is arranged in clockwise order, and is chosen in a way so that only one of the 4 possible corners yields a valid code without errors. This allows determining the code as well as the correct orientation of the marker. For both the frame maker and the split marker described later, the squares describing a bit should have a size of at least 2 pixels to be clearly identified. This is equivalent to existing marker based tracking techniques.

3.2 Split markers

While frame markers still contain a closed border that defines the marker area, *split markers* are composed of two separate barcodes, which further reduces the occupied area. We directly track the two barcodes, which define the marker geometry as well as its id. Compared to typical rectangular markers (such as frame framers), split markers use a different rectangle fitting algorithm and sample the marker area differently.

After employing the standard contour finder also used for square markers, the dominant direction of the elongated candidate contours is determined using perpendicular line regression. The intersection of the contours with the line through the contour's centroid M in the direction of the perpendicular line regression yield the left and right border points B_1 and B_2 of the barcode. The 4 outer corners are constructed out of this border points.

Once the corners are known, the barcode is sampled by setting up a 2x13 regular sampling grid: The outer row of samples must be all black. 2 samples each to the left and right hand side of the inner row must be verified to be black. This design is necessary to reliably detect the corner points in the previous step. The inner 9 samples (see middle picture in Figure 1) encode a 6 bit id plus a 2 bit checksum to improve robustness. The remaining bit out of the inner 9 bits is used to store the orientation.

The algorithm searches for pairs of matching barcodes with opposite orientation bits. If such a pair is found, the outer corner points from both barcodes are used to construct a rectangle. The camera pose relative to the marker is computed from the homography of this rectangle. Since only two of the four sides of the marker contain features required for tracking, one can conveniently hold a marker in the hand with the thumb covering part of the marker (see 2nd picture in Figure 2).

3.3 Dot markers

The previous two marker types are well suited for tracking of small objects such as cards with minimum obtrusion, but are less suitable for covering larger areas, due to the increased visual clutter resulting from placing multiple markers in it.

Most often markers are deemed undesirable since they cover underlying objects or images. It is therefore preferable to reduce the area covered by artificial markings as much as possible, and instead make use of the already existing natural features.

This hybrid approach of minimal markings, which make the analysis of natural texture fast and robust, was taken with the *dot markers* (see right picture in Figure 1 and 3th picture in Figure 2). A dot marker consists of a two-dimensional grid of black circular dots with white surrounding rings, superimposed on a textured flat surface, similar to the design described in [6]. The original texture enclosed by four dots is interpreted as a grid cell, and the appearance of all these grid cells is precomputed out of the dotted map to rapidly and reliably identify a particular grid cell. Thereby the marker covers just 1% of the whole area.

At runtime, a low threshold value (typically at a 3% level in the intensity histogram) is selected, and closed black contours with a diameter of 3 pixels are extracted. The next step aims to identify groups of 4 dots which form a grid cell. Since there is a large number of possible combinations, including contours falsely identified as dots, it is important to quickly reject incorrect combinations using a variety of heuritics.

Grid cells that pass all conditions by identifying 4 dots forming a grid cell are then tested using template matching. The matching starts with the cell with the largest area, which is assumed to provide the best matching result. The unwarped image patch is then checked in all four 90° rotations against all patterns in the precomputed database using normalized cross correlation. The comparison starts with the 8x8 resolution and proceeds to the more expensive test at 32x32 only if the matching at 8x8 exceeds a threshold.

If a grid cell was successfully detected, its offset in the grid is determined and used for estimating the 6DOF of the camera pose relative to the grid. Nonlinear optimization of the camera pose is performed using standard Gauss-Newton iteration.

4 INCREMENTAL TRACKING

Each of the marker technologies presented in the previous chapter tries to minimize the visual clutter by reducing the size of artificial features.

Yet, none of these approaches is able to track completely from natural features without previous training. However, in practice tracking at least temporarily from unknown environments is very desirable since users can usually not be constrained to always

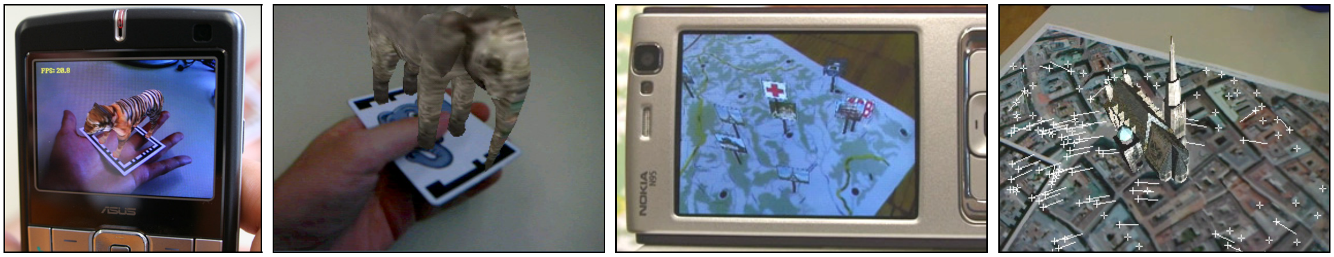

Figure 2. 1st image: frame marker, 2nd image: split marker, 3rd image: dot markers, 4th image: flow vectors of features matched from the previous frame (added for illustration only). Corners without a line could not be matched. The embedded marker (left bottom) is not sufficiently visible for tracking anymore.

point the camera straight to the marker or refrain from occluding dots and other marker features.

In the following, we present two computationally inexpensive approaches to support marker based global localization with incremental tracking from untrained natural features. Both techniques have been successfully implemented on cell phones at interactive frame rates, and can extend the usability of markers well beyond their original purpose: If markers are temporarily lost or occluded, the incremental tracking fills in the gap until a marker is reacquired.

4.1 Incremental tracking using feature following

In many applications markers are placed on a planar surface of interest that shall be augmented by the AR application. Hence there is usually texture around the marker that can be used for natural feature tracking. We exploit this fact by combining marker tracking with a feature following approach operating in a plane. As long as the marker is visible, it is treated as ground truth, and features around the marker are extracted, but not used. Since we assume that features lie in the same plane as the marker, their 3D location can directly be computed from the marker tracking. As soon as the marker tracking fails, the tracker matches the features of the current frame against those of the previous frame via template matching, and begins tracking incrementally.

Candidate feature detection is performed using the FAST corner detector [7], which turned out to deliver high performance rates on phones (~8ms at 320x240 on a 400Mhz ARM CPU). For each candidate, an 8x8 patch is extracted and blurred using a 3x3 Gauss kernel. The blurring increases robustness against pixel offsets introduced by inaccurate corner detection and small affine transformations. To quickly match a candidate against a previous frame, an active search in a 25x25 pixel search neighborhood is used. All features from the previous frame are inserted into a 4x4 search grid of the 320x240 image that provides an almost linear search time. Candidates for matching are tested using sum of absolute differences (SAD) and ranked. If the highest ranked match for a candidate exceeds a certain threshold, it is treated as positive match (see 4th image in Figure 2).

From the matched n point pairs, 4 features are chosen to combine an initial estimate of the homography from the last frame to the current frame.

The selected homography is then refined by minimizing the projection error of all inliers using a Gauss-Newton least-squares fitting process. Finally, the camera pose estimate is updated via homography chaining: The homography of the frame-to-frame correspondence is applied on top of the homography from the previous frame. The pose is then calculated from the updated homography. A similar approach has been described by Simon et al. [8].

Naturally, the approach only works as long as at least 4 suitable points can be matched from one frame to the next. In practice, many more points are required for accurate results. Measurement errors inevitably accumulate, so the estimated pose drifts. In practice acceptable tracking can be provided for about 3-10 seconds, depending on the amount of camera movement and error to be tolerated. This is sufficient in many situations to continue tracking when the marker is lost by an unintended movement of the user. Obviously, the homography-based approach works only for planar or nearly planar environments; in practice this covers most table-top and wall-mounted environments.

4.2 Incremental tracking using pixel flow

Incremental tracking of orientation with inertial sensors has been shown to be highly useful for AR applications to either improve tracking robustness or as a fallback when no other tracking approach is available. While most of today's mobile phones do not have inertial sensors, their built-in camera can be used in a similar way using pixel flow detection [11].

Our pixel flow detector is intended for augmenting a panoramic view of the environment. A marker is used for initially determining the current global location and viewing direction. Then the user is free to turn around observing the augmentations, while remaining in the same location. We apply two different approaches to pixel flow – a more accurate method is tried initially for slow and medium camera movements. If it fails because of fast camera movement, a second, more robust method is used.

The accurate pixel flow tracker uses the same feature following approach as described in section 4.1. All feature flow vectors are inserted into a 2D histogram that encodes the image's movement in X- and Y-direction. The histogram has a size of 32x32 bins and can therefore detect movements of up to ±15 pixels. To detect the dominant pixel flow, the histogram is searched for local maxima. The pixel flow in the overall image is finally estimated as a weighted sum of the maximum and its neighboring values in the histogram. If a second local maximum with a value of more than 60% of the absolute maximum is found, the algorithm assumes a failure and repeats the step with a version of the image scaled down by 50%. Downscaling suppresses noise and hence increases robustness, but also doubles the effective range of the flow detection.

If no pixel flow can be successfully determined within 3 levels of the image pyramid, a second approach for estimating the pixel flow is tried, which is yet more robust, but less accurate. This approach is based on template matching over an image pyramid, which is more commonly used for estimating the optical flow of images [11]. Since a pure rotation model is assumed, a single motion vector valid for the whole image is expected. Hence, the estimated motion vectors from one image pyramid level are averaged and forwarded to the next lower level of the image pyramid as a starting point to limit the search area. In practice, the corner tracking method turns out to be much more accurate than the template matching. However, template matching is more robust under fast camera movement, which often results in images that are too blurred for corner detection. While in most cases, the first method works fine, the second method provides a robust fall back for extreme conditions.

The 2D pixel flow is finally interpreted as rotational motion based on the intrinsic camera parameters. Hence, this simple approach cannot cope with rotations around the viewing direction ("roll"), but "yaw" and "pitch" only. The accumulated rotational offset is then applied on top of the last known absolute pose.

5 EVALUATION

We tested the described methods on an Asus M530W Windows Mobile smartphone. This phone was selected for its CPU, which is clocked at 400MHz (phones currently use CPUs clocked from 200-600MHz) and for its high quality camera, which delivers 25 frames per second.

Table 1 shows the timings of the three proposed marker tracking methods. Naturally, thresholding is independent of the applied marker mode. Shape detection of split markers is slightly slower than for frame markers due to their more complex shape. Split markers take more time for marker detection since each marker consists of two parts and hence requires calculating 2 homographies for unprojection (plus another one for pose estimation).

Dot markers require to spend much time in filtering out non-circular structures at the shape detection stage. The marker detection stage includes the detection of the dot-grid as well as unprojecting candidates and matching them against the database of templates. Altogether this makes dot markers about 2 times slower than rectangular markers.

	Split marker	Frame marker	Dot marker
Thresholding	0.9ms	0.9ms (0.9ms)	0.9ms (0.9ms)
Fiducial Detection	1.6ms	1.4ms (1.4ms)	3.9ms (2.8ms)
Marker Detection	3.1ms	1.8ms (0.0ms)	3.6ms (0.0ms)
Pose Estimation	0.9ms	0.7ms (0.7ms)	0.6ms (0.4ms)
Overall	**6.5ms**	**4.8ms (3.0ms)**	**9.0ms (4.1ms)**

Table 1. Benchmarks of the proposed marker tracking methods. Values in parentheses are for slow camera movement.

Table 1 presents an overview of average timings obtained by tracking the target for about 15 seconds (~400 frames). The values in parentheses present timings for a slow moving camera (or marker). In this case the tracker is able to redetect the marker without executing the full pipeline.

The incremental tracker as described in section 4.1 runs in two modes: When marker tracking succeeds, it only detects corners and harvests patterns. As Table 2 shows, in this mode most of the time is spent in the corner detector. As soon as the marker is lost, the incremental tracker additionally matches the new patches against those of the previous frame and estimates the homography for chaining.

	Corner Detection	Corner Tracking	Homography + Pose	Overall
Detected	8.1ms	1.6ms	0.0ms	**9.7ms**
Undetected	8.1ms	2.3ms	2.7ms	**13.1ms**

Table 2. Benchmarks for Incremental tracking using feature following (as described in section 4.1)

While the speed of the methods mentioned above is mostly independent of image properties, the speed of the incremental tracker using pixel flow depends on many factors, including the number of corners detected, the repeatability of the extracted features, and especially on the speed of the camera movement, which determines how many levels of the image pyramid have to be created and checked. Our measurements show that the pixel flow timings vary between 8 and 14 milliseconds.

6 DISCUSSION AND FUTURE WORK

We have presented a toolkit of new marker tracking techniques running in real-time on off-the-shelf mobile phones. The marker designs produce less image clutter than previous designs, and more easily blend into typical AR environments. By using incremental tracking based on planar feature following or hierarchical pixel flow, situations with occlusions or rapid movements that were difficult to accommodate with previous marker tracking can now be handled with ease. The primary advantages of marker based tracking, in particular its reliability and the built-in object detection capability remain unchanged.

In future we plan to extend the incremental tracker based on planar feature following with true localization and mapping, creating a kind of a "Poor man's SLAM". Such an approach will map the marker's environment and therefore not suffer from drift. However, this approach requires improved feature matching method that can tolerate larger affine changes.

ACKNOWLEDGEMENTS

This project was funded in part by Austrian Science Fund FWF under contracts Y193 and W1209-N15, as well as the European project FP6-2004-IST-4-27571. We would like to thank Gerhard Reitmayr for his valuable comments.

REFERENCES

[1] Fiala, M., ARTag, An Improved Marker System Based on ARToolkit. NRC Canada, Publication Number: NRC: 47419, 2004
[2] Kato, H., Billinghurst, M., Marker Tracking and HMD Calibration for a video-based Augmented Reality Conferencing System, Proceedings of the 2nd International Workshop on Augmented Reality (IWAR 99). pp. 85-94, 1999
[3] Möhring, M., Lessig, C., Bimber, C.. Video See-Through AR on Consumer Cell Phones. Proceedings of International Symposium on Augmented and Mixed Reality (ISMAR'04), pp. 252-253, 2004
[4] Rekimoto, J., Matrix: A Realtime Object Identification and Registration Method for Augmented Reality. Proceedings of Asia Pacific Computer-Human Interaction (APCHI), pp. 63-68, 1998
[5] Rohs, M., Gfeller, B., Using Camera-Equipped Mobile Phones for Interacting with Real-World Objects. Advances in Pervasive Computing, Austrian Computer Society (OCG), pp. 265-271, 2004
[6] Rohs, M, Schöning, J., Krüger, A., Hecht, B., Towards Real-Time Markerless Tracking of Magic Lenses on Paper Maps, Adjunct Proceedings of the 5th International Conference on Pervasive Computing (Pervasive), Late Breaking Results, pp. 69-72, 2007
[7] Rosten, E., Drummond, T., Machine learning for high-speed corner detection, Proceedings of 9th European Conference on Computer Vision (ECCV 2006), pp. 430-443, 2006
[8] Simon, G., Fitzgibbon, A.W., Zisserman, A., Markerless Tracking Using Planar Structures in the Scene, Proceedings of International Symposium on Augmented Reality (ISAR 2000), pp. 120-128, 2000
[9] Wagner, D., Schmalstieg, D., ARToolKitPlus for Pose Tracking on Mobile Devices, Proceedings of 12th Computer Vision Winter Workshop (CVWW'07), pp. 139-146, 2007
[10] Wagner, D., Schmalstieg, D., First Steps Towards Handheld Augmented Reality. Proceedings of the 7th International Conference on Wearable Computers (ISWC 2003), pp. 127-135, 2003
[11] Wang, J., Zhai, S., Canny, J., Camera Phone Based Motion Sensing: Interaction Techniques, Applications and Performance Study, In Proceedings of ACM UIST 2006, pp. 101-110, 2006
[12] Schmalstieg, D., Wagner, D., Experiences with Handheld Augmented Reality, The Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07), 2007