Towards Real-Time Markerless Tracking of Magic Lenses on Paper Maps

Michael Rohs[*], Johannes Schöning[†], Antonio Krüger[‡], and Brent Hecht[§]

Abstract. *Recently, many applications have been proposed that focus on the combination of paper maps and mobile devices. Paper maps provide high-resolution, large-scale information with zero power consumption. Mobile devices offer dynamic, personalized, up-to-date content. The common feature of all these applications is that they use handheld devices as map-referenced mobile lenses that overlay geo-referenced information on top of the physical map. Despite their differences these applications all require additional infrastructure on the map, for example RFID tags or visual markers. In this paper we present a first approach of continuous real-time tracking of a camera-equipped mobile device relative to a map using visual structures already present in many maps.*

1. Introduction

The combination of paper maps and mobile devices is an excellent use case for proof-of-concept implementations linking physical and virtual worlds. Paper maps are still superior to their digital counterparts on mobile displays in terms of size, resolution, and visual presentation quality. They do not consume any power and provide a static frame of reference. But in contrast to mobile devices their content is static, not adaptable or selectable by the user, and not always up-to-date. Applications like [2, 3] try to combine the advantages of both worlds by realizing a digital link between a high resolution paper map and a mobile device. Reilly et al. [2] use maps fitted with an array of RFID tags that have the disadvantage of low spatial resolution and high production costs. The marker-based approach of Schöning et al. [3] has the disadvantage that the markers obscure valuable map space. This problem has been addressed in several ways: semi-transparent markers, multiple but smaller markers, and covering the marker by an appropriate digital patch of the map on the mobile device display. *Map Snapper*[1] is a non-realtime application that takes a picture of a paper map, sends it off to a central server for analysis, and responds with details on nearby points of interest. In our approach we provide video see-through augmentation of the paper map with a camera-equipped handheld device. The user's view is mediated by the device and combined with overlay graphics on the display. The user acts on two layers of information – the "transparent" device screen in the focus and the map surface in the visual context. The camera display unit acts as a movable window into a computer-augmented view of the physical map. This is known as a *see-through interface* or a *magic lens* [1].

[*]Deutsche Telekom Laboratories, TU Berlin, Germany, michael.rohs@telekom.de
[†]Institute for Geoinformatics, University of Münster, Germany, j.schoening@uni-muenster.de
[‡]Institute for Geoinformatics, University of Münster, Germany, antonio.krueger@uni-muenster.de
[§]Department of Geography, University of California Santa Barbara, USA, bhecht@geog.ucsb.edu
[1]`www.ecs.soton.ac.uk/about/news/1038`

We are not primarily focusing on the computer vision problem here, but on realizing a way to use camera phone based real-time tracking for exploring novel interactions with paper maps.

2. Tracking of the Magic Lens

In the presented approach the video stream of the camera is continuously analyzed in order to accurately align the overlay graphics with the camera view. To simplify the task of real-time tracking on a mobile phone with limited computing resources the map is modified in our prototype setup: it contains black dots arranged in a regular grid. In a future version, we aim to replace the dots by horizontal and vertical lines (a *map grid*), which are less obtrusive and commonly found on maps, for example to simplify the lookup of street names. The map grid subdivides the map into squared patches that are used to correlate the camera view with the map.

The goal of the real-time tracking algorithm is to compute the position of the camera view, such that appropriate graphical overlays can be generated with pixel-level accuracy for the spatial extent visible on the mobile display. Since objects on the paper map typically appear perspectively distorted in the camera view, this requires the computation of a projective mapping from the map coordinate system to the camera image coordinate system for each camera frame. The algorithm first computes an undirected graph with potential grid dots as vertices and extracts all correlation patches. A correlation patch is defined as a loop of four vertices containing no other vertices, with successive edges close to orthogonal and having almost the same length. Once patches are identified, they are sampled and the sample points are correlated with precomputed sample points of the map. The highest correlation wins and predicts the position on the map. In more detail, the algorithm performs the following steps:

Find grid dot candidates: First, an adaptive thresholding method is performed to produce a black-and-white version of the camera image. Next, connected regions are determined and the axis ratios of their minor and major axes are computed. If size and axis ratio of a connected region are within certain limits, the region is classified as a potential grid dot on the map.

Find edges: A fully connected undirected graph is computed with the grid dot candidates as vertices (Figure 1a). Edges are classified as "horizontal" ($\Delta x > \Delta y$) or "vertical" ($\Delta y \geq \Delta x$) and the role of each vertex in an edge (0 = left, 1 = right, 2 = top, 3 = bottom) is stored for efficient lookup. Each edge is stored twice in a hash table: once for its left/top vertex v_{lt} and once for its right/bottom vertex v_{rb}. Their hash keys are $k_{lt} = 4v_{lt}^{index} + v_{lt}^{role}$ and $k_{rb} = 4v_{rb}^{index} + v_{rb}^{role}$, respectively. The edge classification implies that the camera cannot be rotated by more than 45° clockwise or counterclockwise from the upright direction. If the map correlation patches are stored $\pm 90°$ rotated in addition to upright, this limitation can be dropped.

Find patches: Each area between four grid dots that does not contain another grid dot is called a correlation patch. For each correlation patch that is present in the image, this step identifies its four corners (Figure 1b). The algorithm iterates through the hash table looking for loops of length 4 of alternating horizontal and vertical edges, starting with the top edge of a patch and then going clockwise. Successive edges in the loop have to be close to orthogonal $(\pm 5\%)$ and of nearly the same length $(\pm 10\%)$. Given a vertex index i, $i \in \{0, \ldots, v-1\}$, and its vertex role r, $r \in \{0, \ldots, 3\}$, the key of the corresponding edges in the hash table is $k = 4i + r$. The search for correlation patch corners starts at the left vertices of horizontal edges, which have key values $k = 4i + 0$ in hash table h. This yields the first set of edge candidates $e_{top} = h(k)$. The next edge candidates in the loop are $e_{right} = h(4\, e_{top}.v_{rb}^{index} + 2)$, $e_{bottom} = h(4\, e_{right}.v_{rb}^{index} + 1)$, and $e_{left} = h(4\, e_{bottom}.v_{lt}^{index} + 3)$.

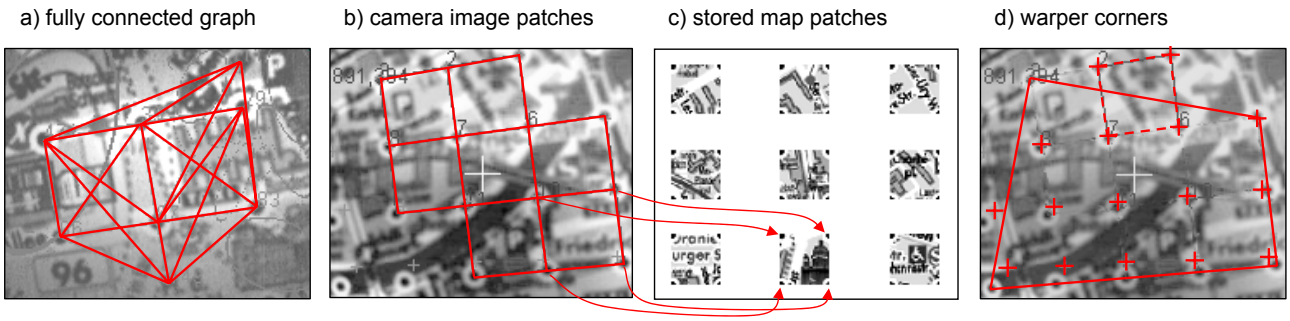

Figure 1. (a) Fully connected graph of grid dot candidates. (b) Correlation patches identified in a camera image. (c) Precomputed stored patches of the map. (d) Final perspective warper (large frame) and seed warper (small dashed frame) with extrapolated grid points (small crosses).

Sample each patch: For each patch that is found, a projective mapping of the patch to a $s = 32 \times 32$ pixel area is computed and equally-spaced gray-value pixel samples are taken.

Compute correlations: In this step the correlation coefficients between each patch found in the image and each stored map patch are computed. The correlation coefficient between image patch I and map patch M with sample points $i_j, m_j, j \in \{1, \ldots, s\}$, respectively, is defined as:

$$r(I, M) = \frac{\sum_{j=1}^{s}(i_j - \bar{i})(m_j - \overline{m})}{\sqrt{(\sum_{j=1}^{s}(i_j - \bar{i})^2)(\sum_{j=1}^{s}(m_j - \overline{m})^2)}}, \quad \bar{i} = \frac{1}{s}\sum_{j=1}^{s} i_j, \quad \overline{m} = \frac{1}{s}\sum_{j=1}^{s} m_j$$

This step was found to be the most computationally intensive and was thus optimized as much as possible: The deviations and sum of squares for the map are precomputed; since most camera phones do not contain a floating point unit shifted integer operations are used instead of floating point operations; and squared correlations are computed to avoid taking the square root for each value.

Compute maximum correlation indices: For each patch in the camera image $I_j, j \in \{1, \ldots, n\}$ the best correlating map patch $M_k, k \in \{1, \ldots, m\}$ is determined (Figure 1c). The result is a list of pairs (I_j, M_k) of image patches and map patches.

Find reliable patch: This step tries to identify an image patch whose map position has been correctly recognized. For each maximum correlation pair (I, M) the corners of I and M are treated as correspondences that define a projective mapping. For each mapping we project the cursor point at the image center to the corresponding map point. If two different mappings (I_j, M_k) and $(I_{j'}, M_{k'})$ vote for the same point, then patches I_j and $I_{j'}$ are classified as reliable. Either mapping can now be used to generate graphical overlays. However, if the patch appears very small in the frame, the mapping is unstable. Hence, two further steps are added to base the projective mapping on a larger area.

Set grid dot coordinates: A reliable patch found in the previous step is used as a seed to assign map coordinate values to each grid dot in the image. This projective mapping will generally be more inaccurate the further the distance of the grid point to the seed patch (small dashed frame in the upper part of Figure 1d). Using the seed's mapping the image coordinates of each grid dot candidate that was found in the first step are warped to the map coordinate system (small crosses in Figure 1d). If the distance of the resulting point to the nearest regular grid point is below a threshold, then the region is classified as an actual grid dot and gets assigned the map coordinates of the nearest grid point.

Compute maximum warper: Finally, the perspective mapping for generating the graphical overlays

is computed by looking for grid dots that are closest to the image corners (large frame in Figure 1d) and thus covers the maximum area.

3. Current State of Implementation and First Test

Our test map, a city map of Berlin, has 16×12 = 192 correlation patches. If printed on a DIN A3 sheet (print area 36×27 cm), the size of each patch is 21×21 mm and the diameter of each black grid dot is 3 mm (surrounded by 1 mm whitespace). From each patch 32×32 gray-value samples are taken. With a frame size of 176×144 pixels the algorithm presented above reliably identifies the correct map patch. We successfully tested the algorithm with 3 other city maps, with a maximum of 386 patches. The method depends on the visual appearance of the map and in particular on the distinguishability of patches from one another. If multiple patches are visually identical the algorithm cannot determine the location of the patch. However, in practice city maps have such a rich and varied visual content that this does not seem to be a problem. If a patch cannot be uniquely identified by its sample points then the movement of the grid dots (or grid lines) can still be tracked. In addition, the history of movement coordinates can be used in order to decide which estimate to pick from a number of alternatives with similar correlation values. Moreover, the search space can be limited to the set of most recently identified patches and their neighbors. This should massively speed up the correlation computation, but has not been implemented yet. Given the test map with 192 patches, the current implementation (without these optimizations and with 32×32 samples per patch) processes 1 to 5 frames per second on a Nokia N80, depending on the number of patches found in the frame.

4. Discussion and Conclusions

In this paper we have presented an algorithm that allows for continuous real-time tracking of mobile embedded cameras over physical maps, based on structural information partially derived from the map itself. Instead of using (several) visually obtrusive codes, the approach is based on a grid of points uniformly distributed over the map. Although these points are still consuming map space, they cover considerably less space than an equivalent approach relying on visual codes. Of course, further steps are necessary to achieve true markerless tracking. As a first approximation, we will try to use the map grid often found in reference maps to detect the patches. To further integrate the markers into the visuals of the map we also plan to experiment with irregular point distributions. This would allow the placement of dots at less prominent locations on the map. Even better results could be achieved by using signatures already present (for example symbols for parking lots [2]) depending on their distribution and frequency.

References

[1] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose. Toolglass and magic lenses: the see-through interface. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 73–80. ACM Press, 1993.

[2] D. F. Reilly, M. E. Rodgers, R. Argue, M. Nunes, and K. Inkpen. Marked-up maps: combining paper maps and electronic information resources. *Personal and Ubiquitous Computing*, pages 215–226, 2006.

[3] J. Schöning, A. Krüger, and H. J. Müller. Interaction of mobile camera devices with physical maps. *Adjunct Proceeding of the Fourth International Conference on Pervasive Computing*, 2006.