Real-time Panoramic Mapping and Tracking on Mobile Phones

Daniel Wagner, Alessandro Mulloni, Tobias Langlotz, Dieter Schmalstieg

Graz University of Technology

Figure 1. Outdoor and indoor panoramas created on the fly using real-time mapping and tracking.

ABSTRACT

We present a novel method for the real-time creation and tracking of panoramic maps on mobile phones. The maps generated with this technique are visually appealing, very accurate and allow drift-free rotation tracking. This method runs on mobile phones at 30Hz and has applications in the creation of panoramic images for offline browsing, for visual enhancements through environment mapping and for outdoor Augmented Reality on mobile phones.

KEYWORDS: Panorama creation, Tracking, Mobile phone

INDEX TERMS: H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems-Artificial, augmented, and virtual realities; I.4.1 [Image Processing and Computer Vision]: Scene Analysis-Tracking

1 INTRODUCTION

Tracking for outdoor Augmented Reality (AR) applications has very demanding requirements: It must deliver an accurate registration with respect to a given coordinate system, be robust and run in real time. Despite recent improvements, outdoor tracking still remains a difficult problem. Recently, mobile phones have become increasingly attractive for AR. With the built-in camera as the primary sensor, phones facilitate intuitive point-and-shoot interaction with the environment.

Most outdoor tracking systems rely on inertial sensors to improve robustness. Even though some modern smart phones integrate a linear accelerometer, it is of little help in typical AR scenarios since it only delivers translational motion. Instead, most successful approaches rely on gyroscope sensors that measure rotations, which are primary sources for tracking instabilities. However, no mobile phone today possesses such sensors.

It is likely that mobile phones will soon be equipped with gyroscopic sensors too. Schall et. al have shown [11] that carefully integrating a panoramic tracker into a system with GPS, compass, linear accelerometer and gyro can further improve the system's robustness.

In this paper we describe a natural-feature mapping and tracking method that is efficient, robust and allows for 3-degree-of-freedom tracking in outdoor scenarios on mobile phones. Assuming pure rotational movements, the method creates a panoramic map from the live camera stream (see Figure 1). The conceptual approach is similar to simultaneous localization and mapping (SLAM) [4][6]: For each video frame, the camera is first registered based on features in the map; In a second step, the map is then extended with new features from viewing directions that have not been observed before. Yet, while traditional SLAM systems create a sparse map of the environment and refine features over multiple observations (typically using triangulation), our approach creates a dense map of features, which are mapped during their first observation and not refined again.

The first camera image is completely projected onto the environment map. When possible, the orientation and position of the first frame in the map can be derived from the phone's accelerometer and compass. For all successive frames, the camera pose is updated – based on the existing data in the map – and the map is extended by only projecting areas that have not yet been

{wagner|mulloni|langlotz|schmalstieg} @ icg.tugraz.at
Institute for Computer Graphics and Vision; Inffeldg. 16, 8010 Graz, Austria

stored. Since a large number of features and a sub-pixel-accurate projection model are used for camera tracking, we show that this is a valid approach for accurate, robust and drift-free tracking.

Figure 2 shows the mapping and tracking pipeline. Tracking requires a map for estimating the orientation, whereas mapping requires an orientation for updating the map. A known starting orientation with a sufficient number of natural features in view must be used to initialize the map.

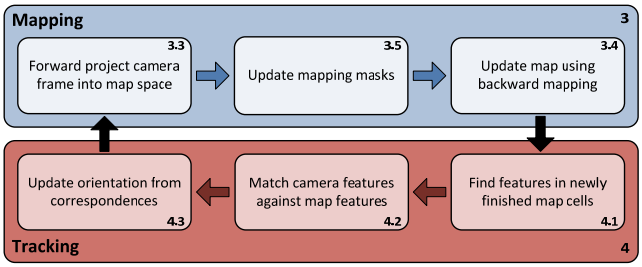

Figure 2. High-level overview of the mapping and tracking pipeline. For each block, we embedded the corresponding section numbers.

The method assumes a pure rotational motion. This assumption is not always viable for a mobile phone. However, in many outdoor scenarios the distance between the camera and the objects in the environment is large compared to the involuntary translational motion that occurs when rotating a handheld device. As shown by DiVerdi et. al [5], errors are therefore negligible.

The contribution of this paper is a new method that creates and tracks panoramic maps in real time (30Hz) on a mobile phone. Similar results have previously only been available on hardware at least one order of magnitude more powerful. We present a careful analysis of the individual steps of panorama creation and tracking, and how higher efficiency can be obtained through various algorithmic means and trade-offs.

We describe two proof-of-concept applications that show the practicability of our approach: A tool that guides a user in creating gapless panoramic pictures and an application to create and browse annotated panoramic images.

2 Related work

Panorama creation is a widely discussed topic in computer vision. Most of the existing approaches create panoramic images in an offline process [3][12][13]. These methods typically use SIFT [8] or similar descriptors to match image features. Szeliski [14] gives a good overview of the many existing techniques for image alignment and stitching.

In contrast to these offline approaches, Adams et al. [1] align camera images in real time on mobile phones in their *View Finder Alignment* work. Consecutive camera images are roughly aligned by calculating a histogram of gradients for four 2D directions. The alignment is then refined using feature points. Successively, tracking of the optical flow is used to create a panoramic image. While this approach works in real time on current mobile phones, it neither permits the creation of a closed 360° panorama, nor does it track the 3D motion of the phone. In [16], the View Finder Alignment technology is used to trigger the automatic capturing of high-resolution images that can be used to generate a high-quality panoramic image. The creation of this panoramic image does not run in real-time and requires offline processing.

Baudisch et al. [2] created a real-time preview for panoramic imaging based on the work presented in [12]. Their application stitches low quality panoramas on the fly and thereby provides an estimate of the maximum rectangular cropping area that can be created from the set of images taken so far. The visualization of the already captured panorama is similar to the visualization used by our system.

Envisor [5] tracks the orientation of a camera in real time and an environment map is created on the fly. This is achieved by calculating the optical flow between successive frames. Similar to [1], the optical flow measurements are refined with computationally expensive landmark tracking to avoid the drift introduced by frame-to-frame feature matching. While the results of this approach are similar to ours, the system cannot run on phones due to the high computational cost of the method, which requires extensive GPU processing to run in real time.

Montiel and Davison created a visual compass [9] based on single-camera SLAM [4]. They used an extended-Kalman-filter formulation of the tracking problem to compute orientation from dynamically acquired landmark features. Since their approach creates a sparse 3D reconstruction of the environment, the system is not restricted to rotations only. Klein and Murray also introduced another successful approach of SLAM-based tracking for augmented reality [6]. Recently Klein and Murray showed a SLAM system running on a mobile phone [7]. However, due to low processing power of mobile phones Klein's SLAM system is limited to a few hundred keypoints whereas our method can handle 1000s of keypoints and is several times faster on a similar device.

The related work discussed above either does not run in real time on current phones due to high computational costs, or it solves only one task between panorama creation and panorama tracking. The approach described in this paper combines panoramic mapping and orientation tracking, both working on the same data set. It can therefore be used for creating panoramic content as well as for browsing and augmenting previously created panoramic images.

3 Panoramic mapping

Several types of maps can be used in order to create a map of the environment. Cube maps are common in computer graphics, but they present discontinuities at the cube's edges. Spherical maps solve the problem of discontinuities at the price of strong nonlinearity: Up and down directions are mapped to all the top and bottom pixel rows of the map. We chose a cylindrical map (see SEQ) because it can be trivially unwrapped to a single texture with a single discontinuity on the left and right borders. While the horizontal axis does not suffer from nonlinearities, the map becomes more compressed at the top and the bottom. Since the cylinder is not closed vertically, there is a limit to the pitch angles that can be mapped. This limit is acceptable for practical use, since a map of the sky and ground is usually not required.

We fix the cylinder's radius to 1 and its height to $\pi/2$. Since the circumference of the cylinder is 2π, the map that is created by unwrapping the cylinder is exactly 4 times as wide as high ($\pi/2$ high and 2π wide). A power of two for the aspect ratio simplifies using the map for texturing. The map covers 360° horizontally while the range covered vertically is given by the arctangent of the cylinder's half-height ($\pi/4$), therefore [-38.15°, 38.15°].

3.1 Organization of the map

Most mobile phones today can take multi-megapixel photos but the live video feed is usually restricted to 320x240 pixels. A typical mobile phone camera has roughly a 60° horizontal field of view. A complete 360° horizontal panorama would be about 320 pixels / 60° · 360° = 1920 pixels wide. We chose a map resolution of 2048x512 pixels, which is the smallest power of two bigger than the camera's resolution and therefore transfers image data from the camera into map space without any loss in image quality.

To increase tracking robustness lower-resolution maps (1024x256 and 512x128) are also created (see sections 4.2 and 4.3).

The map is split into a regular grid of 32x8 cells that simplify working with the unfinished map (see Figure 3). Every cell can have two states: either unfinished (empty or partially filled with mapped pixels) or finished (completely filled). When a cell is finished, it is down-sampled from the full resolution to the lower levels and keypoints are extracted for tracking purposes.

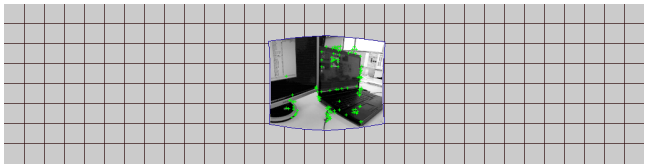

Figure 3. Grid of cells composing the map, after the first frame has been projected. The green dots mark keypoints used for tracking.

3.2 Calibrating the camera

Since the map is filled by projecting pixel data from the camera image onto the map, full knowledge on the intrinsic and extrinsic camera parameters is required for an accurate mapping process.

Assuming that the mobile camera does not change zoom or focus, the intrinsic parameters can be estimated once in an off-line step and stored for later use. The principle point and the focal lengths in the x and y directions are estimated. Modern mobile phones internally correct most of the radial distortion introduced by the camera's lens. However, there is still distortion left, so additional correction is required. To measure these parameters, pictures of a calibration pattern are taken, which are then evaluated with the Caltech camera calibration toolbox[1].

We additionally correct for artifacts due to vignetting, which consists of a reduction in pixel intensities at the image periphery. While there are several causes for vignetting, digital cameras mostly suffer from "pixel vignetting" which is caused by the sensors depending on the angle of the incoming light: Sensor elements farther from the image centre receive light at a steeper angle and therefore sense darker pixel intensities. This effect can be modeled with a non-linear radial falloff. The vignette strength is estimated by taking a picture of a diffusely-lit white board. The average intensities close to all the four corners are measured and the difference from the image centre is noted.

3.3 Projecting from camera into map space

Our method assumes pure rotational motion. Although this is unlikely for a handheld camera, a trained user can effectively minimize parallax errors. DiVerdi [5] provides a detailed analysis on the effect of violating the pure-rotation requirement with respect to the distance of the objects in the mapped environment. We conveniently set the camera position to the origin *(0,0,0)* at the centre of our mapping cylinder (see Figure 4).

A fixed camera position leaves 3 rotational degrees of freedom to estimate for correctly projecting camera images onto the mapping cylinder. Depending on the availability of an accelerometer, the system is either initialized from the measured roll and pitch angles, or we assume a roughly horizontal orientation. In section 5 we show the effects of starting with a wrong orientation and how to fix them during the mapping process by warping the map.

$$R = \pi'(\delta'(K^{-1} \cdot P)) \quad (1)$$

$$M = \mu (\iota (O^{-1} \cdot R, C)) \quad (2)$$

[1] http://www.vision.caltech.edu/bouguetj/calib_doc/

Given a known (or assumed) camera orientation O, we use *forward mapping* to estimate the area of the cylinder's surface that is covered by the current camera image. Given a pixel's device coordinate P, we first transform it into an ideal coordinate by multiplying it with the inverse of the camera matrix K and removing radial distortion using a function δ' in (1). This 2D coordinate is then unprojected into a 3D ray R using a function π' by adding a z-coordinate of 1. The ray is then rotated from map space into object space using the inverse of the camera rotation matrix O^{-1}. Next, the ray is intersected with the cylinder using a function ι to get the pixel's 3D position on the mapping cylinder. Finally, the 3D position is converted into a 2D map position M using a function μ in (2).

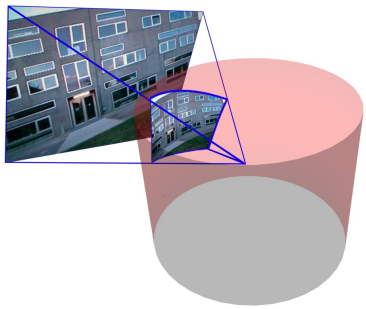

Figure 4. Projection of the camera image onto the cylindrical map.

We define a rectangle for the camera frame in camera space, setting its corners in pixel coordinates of the camera image. The rectangle is then forward-mapped onto the map and defines the mask for those pixels that are covered by the current video frame. Due to radial distortion and the nonlinearity of the mapping, each rectangle side is sub-divided three times to get a smooth curve in the target space (see blue image frame in Figure 3).

3.4 Filling the map with pixels

The forward-mapped camera frame gives an almost pixel-accurate mask for those pixels that the current video image can contribute. However, using forward mapping to fill the map with pixels can cause holes or overdrawing of pixels.

To fill the map with pixels we therefore use *backward mapping*. Starting with a 3D map position M' on the cylinder, we calculate a ray from M' to the camera centre using function μ', and then rotate the ray using the orientation O. This results in R' in (3), which is then projected onto the camera plane using function π. Radial distortion is applied using function δ and the resulting ideal coordinate is converted into a device coordinate P' via the camera matrix K in (4). The resulting coordinate generally lies somewhere between pixels, so we interpolate linearly to achieve a sub-pixel-accurate color. Finally, we compensate for vignetting and store the pixel color in the map.

$$R' = O * \mu'(M') \quad (3)$$

$$P' = K * \delta (\pi (R')) \quad (4)$$

3.5 Speeding up the mapping process

One 320x240 camera image will always require back projecting roughly 75,000 pixels. Such a workload is too high to run in real time (at least 15 frames per second) on current mobile phones. To speed the process up, each map pixel is set only once as soon as it can be back projected for the first time. During the first camera frame a large number of pixels have to be transferred (as in Figure 3). For all subsequent frames, only a few pixels are mapped: with slow camera movements only a few rows or columns of new pixels become visible per frame. This drastically reduces the

required computational power for keeping the map up to date. For instance, horizontally rotating a camera (with a resolution of 320x240 pixels and a field of view of 60°) by 90° in 2 seconds results in only about 16 pixel columns – or 3840 pixels – to be mapped per frame. This is only 5% of a whole camera image.

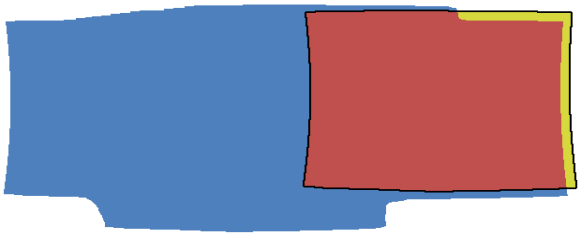

Figure 5. Masks created during a rotation of the camera to the right. Blue: Mask *M* of the map so far; Black border: Mask *T(P)* for the current camera pose; Red: Intersection of *M* and *T*(P); Yellow: Mask *N* of operation (5) representing the pixels that still need to be mapped. *N* is exaggerated here for better visibility.

This approach requires quickly filtering out those pixels that fall inside the projected camera frame and that have already been mapped. While a simple mask with one entry per pixel would be sufficient, the process would be too slow and too memory intensive. We use a run-length encoded (RLE) mask to store zero or more spans per row that define which pixels of the row are mapped and which are not. A span is a compact representation that only stores its left and right coordinates. Spans are highly efficient for Boolean operations, which can be quickly executed by simply comparing the left and right coordinates of two spans.

A mask *M* (see Figure 5) is defined for the map at its highest resolution. Initially this mask is empty. For every frame, the projected camera frame is rasterized into spans creating a temporary mask *T(P)* that describes which pixels can be mapped under the current pose *P*. The camera mask *T(P)* and the map mask *M* are then combined using a row-wise Boolean operation. The resulting mask *N* in (5) contains locations for only those pixel that are set in the camera mask *T(P)* but not in the map mask *M*. Hence, *N* describes those pixels in the map that will be filled in the current frame. The map mask *M* is updated to include new pixels using the operation (6).

$$N = T(P)\ AND\ NOT\ M \quad (5)$$

$$M = T(P)\ OR\ M \quad (6)$$

$$F_i = U(C_i)\ AND\ M \quad (7)$$

The pixels covered by the mask *N* are back projected and the resulting color values are written into the map. As introduced in section 3.1, the map is subdivided into cells. While filling the map, a book of the cells that are updated during the current frame is kept. Once the mapping process for a frame is finished, we check whether each updated cell has been completely filled: For each updated cell C_i a mask $U(C_i)$ is defined containing only the area of such a cell. This mask is then intersected with *M* using the operation (7). If the combined mask F_i in (7) covers the complete area of C_i, then this cell has been completely filled and can be marked as finished. A finished cell is downsampled to the smaller map levels and keypoints are extracted for tracking, as described in the next section.

4 PANORAMIC TRACKING

The mapping process assumes an accurate estimate of the camera orientation. We now present an efficient method for tracking the camera orientation from the map that is being built.

4.1 Keypoint extraction

The tracker applies the FAST corner detector [10] on finished cells to extract keypoints. For every keypoint, FAST gives a score of how strong the corner appears. The thresholds are adjusted for this score according to the resolution of the map the cell belongs to (as detailed in Table 1). E.g., for the cells of the highest-resolution map (64x64 pixels in size) we use a FAST threshold of 12. For the cells of the smaller levels a lower threshold is used, to consider the smoothing due to downsampling. These threshold values are chosen deliberately low to ensure that always more than enough keypoints are extracted. The keypoints are then sorted by corner strength and only the strongest keypoints are kept. E.g., for 64x64-pixel cells 40 keypoints are kept.

cell size in pixels	*FAST threshold*	*max keypoints per cell*
64x64	12	40
32x32	9	20
16x16	9	15

Table 1. Cell configurations

We organize keypoints on a cell-level because it is more efficient to extract keypoints in a single run once an area of a certain size is finished. It also avoids problems with looking for keypoints close to areas that have not yet been finished: Since each cell is considered as a separate image, the corner detector itself takes care to respect the cell's border. Finally, organizing keypoints by cells provides an efficient method to determine which keypoints to match during tracking.

4.2 Keypoint tracking

We apply an active-search procedure based on a motion model in order to track keypoints from one frame to the following one. Keypoints in the camera image are always compared against their counterpart in the map. Hence, unlike other trackers, this tracking approach is generally drift-free. Still, errors in the mapping process accumulate so that the map is not 100% accurate: a rotation around a certain angle is not mapped exactly with the angle in the database (see section 8.1). However, once the map is built, tracking is as accurate as the map that has been created.

To estimate the current camera orientation the tracker requires a rough guess. In the first frame this guess corresponds to the orientation used for initializing the system. For all successive frames, we use a motion model with constant velocity to guess an orientation. We calculate velocity as the difference in orientation between the current and the previous frame.

We then refine the initial guess: Based on the guessed orientation, the camera frame is forward projected onto the map to find those cells that overlap with the visible part of the map. The keypoints of these cells are then back projected onto the camera image. All keypoints that are back projected outside the camera image are filtered out. We create 8x8-pixel patches for each keypoint by affinely warping the map area around the keypoint using the current orientation matrix. The warped patches represent the support areas of the corresponding keypoints, as they should appear in the current camera image. The tracker uses normalized cross correlation (over a search area) at the expected keypoint locations in the camera image. Since template matching is slow, it is important to limit the size of the search area. A multi-scale approach is applied to track keypoints over long distances while keeping the search area small: The first search is at the lowest resolution of the map (512x128 pixels) against a camera image that has been down-sampled to quarter size (80x60 pixels) using a search radius of 5 pixels. The coordinate with the best matching score is then refined to sub-pixel accuracy by fitting a 2D quadratic term to the matching scores of the 3x3 neighborhood.

Since all three degrees of freedom of the camera are respected while warping the patches, the template matching works for arbitrary camera orientations.

4.3 Orientation update

The correspondences between 3D cylinder coordinates and 2D camera coordinates are used in a non-linear refinement process with the initial orientation guess as a starting point. The refinement uses a Gauss-Newton iteration: The same optimization takes place as by a 6-degree-of-freedom camera pose, but position terms are ignored and the Jacobians are only calculated for the three rotation parameters. Reprojection errors and inaccuracies are dealt with effectively using an M-estimator. The final 3x3 system is then solved using Cholesky decomposition.

Starting at a low resolution with only a few keypoints and a search radius of 5 pixels allows correcting gross orientation errors efficiently but does not deliver a very accurate orientation. The orientation is therefore refined again by matching the keypoints from the medium-resolution map (1024x512 pixels) against a half-resolution camera image (160x120 pixels). Since the orientation is now much more accurate than the original guess, the search area is restricted to a radius of 2 pixels only. Finally, another refinement step is executed at the full resolution map against the full-resolution camera image.

Since each successive refinement is based on larger cells it also uses more keypoints than the previous refinement. In the last step several hundred keypoints are typically available for estimating a highly accurate orientation.

4.4 Relocalization

The tracker can only follow the keypoints from one frame to the next. As any pure tracker it is therefore not able to reinitialize itself from an arbitrary orientation. However, at some point every tracker can fail and relocalization is fundamental for any practical system. A relocalization mechanism is therefore added in case the tracker does not find enough keypoints, or when the reprojection error after refinement is too large to trust the orientation.

The relocalizer works by storing low-resolution keyframes with their respective camera orientation in the background, as the map is being created. In case the tracking is lost, the current camera image is compared to those keyframes using normalized cross correlation. To make the matching more robust both the keyframes (once, when we store them) and the camera image are blurred. If a matching keyframe is found, an orientation refinement is started using the keyframe's orientation as a starting point.

In order to limit the memory overhead of storing keyframes, the camera image is downsampled to quarter resolution (80x60 pixels). Additionally, the relocalizer keeps track of orientations already covered by a keyframe: The orientation is converted into a yaw/pitch/roll representation and the three components are quantized into 12 bins for yaw (±180°), 4 bins for pitch (±30°) and 6 bins for roll (±90°). Storing only ±90° for roll is a contribution to the limited memory usage but results in not being able to recover an upside-down orientation. For each bin a unique keyframe is stored, which is only overwritten if the stored keyframe is older than 20 seconds. In the described configuration, the relocalizer requires less than 1.5MByte of memory for a full set of keyframes.

5 Fixing Incorrect Initial Orientations

The mapping process relies on an initial rough guess of the camera orientation. Starting with a wrong initial guess for pitch or roll violates the limits of the cylindrical environment model. Figure 6 (right) shows the effect of a pure rotation around the yaw axis after a wrong initial guess for the pitch angle: The mapping module assumed that the initial pitch was zero, but in fact the camera was looking about 20° upwards. Rotating the camera around the vertical axis does not result in a horizontal movement in map space as expected, because the vertical axis is not where the mapping module believes it to be. The effect is that the camera quickly moves out of the map, which strongly limits the horizontal angle that can be mapped. Similar effects occur when the roll angle is wrongly initialized.

Fortunately, a map that was built from an incorrect starting orientation can be fixed later on, by reprojecting it onto another cylinder's map. Carrying out this reprojection on a low-resolution map of 256x64 pixels allows interactive frame rates. A user can therefore warp the preview map, e.g. by dragging the map's center on the screen into its correct position.

Internally, the remapper rotates a second cylinder around the x- or y-axis (see Figure 6, left), accordingly to the user's input, and reprojects all pixels of the low-resolution map from the reference cylinder onto the rotated cylinder. The result can be displayed on the screen in real time. When the user is satisfied with the correction the remapper reprojects the map at its full resolution. It creates new down-sampled levels and extracts new keypoints for all levels, so that the tracker can work with the warped map. For quality reasons a Lanczos filter is used for resampling the map.

Figure 6. Left: Mapping from the reference cylinder (red) onto another cylinder (gray), tilted 20°. Top-right: Effect of a pure horizontal camera movement after a wrongly initialized pitch angle. Bottom-right: The same map fixed using reprojection.

Reprojection will always create holes in the map in those parts that were previously outside the map and have then been moved in. The user can continue to map the environment to fill these holes.

6 Initialization from an Existing Map

The relocalization method described in section 4.4 is fast, but it only works for orientations where camera image samples exist. Hence, this method is suitable for relocalization, but not for initializing from a previously existing map, because storing all camera image samples would require too much memory. In this section, the method for initializing the camera's orientation is outlined, which relies only on a (partially) finished map that was previously created. This method is suitable for initializing the tracker after loading a map from the device's storage or from the Internet.

Starting with a map loaded from a file, keypoints are extracted from the map and PhonySIFT descriptors [15] are created, which allow robust, rotation invariant matching (see Figure 7). Keypoints are also extracted and descriptors are created for the live camera image. Online creation of an efficient search structure such as the spill forest suggested in [15] is too slow to be executed on a mobile phone. Hence, brute force matching is relied upon.

Given the descriptor sets from the map and the current camera image, a RANSAC-like approach is applied to find the orientation. To begin with, all camera features are matched

against all map features, obtaining a set of correspondences. Next, a histogram of correspondence occurrences is created in the horizontal direction, and the direction with the largest number of correspondences in a window of 78,75° (7 cells) is selected. Following this, only correspondences that fall into this window are considered.

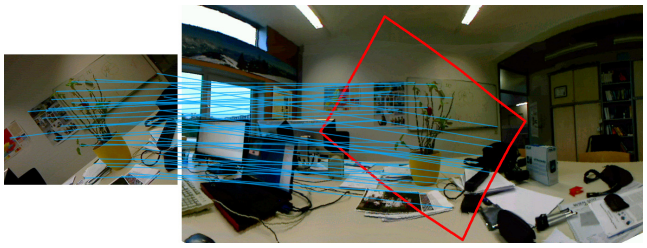

Figure 7. Initialization from an existing map; Left: Camera image to localize. Right: 3DOF localization of the camera image.

Since localization has three degrees of freedom, two matching features are required as a hypothesis. Pairs of two correspondences are built and an orientation is calculated, which is then validated against the other correspondences. If a large number of correspondences support this hypothesis it is checked again and the hypothesis is refined using the method described in section 4.3. If no orientation can be found in any possible pairs of correspondences, the whole process is repeated with the next live camera image.

The rotation invariance of PhonySIFT enables this method to estimate an orientation under arbitrary device rotation as long as the currently visible environment is available in the map. Usually an orientation is found within a few frames.

7 Loop Closing

Due to precision errors that accumulate as the map is extended away from its starting orientation, a full 360° sweep will not be mapped exactly at the map's edges. There will be a noticeable discontinuity at the location in the map where the left-most and right-most mapped pixels touch. Loop closing is the process of accurately estimating the error in the map and transforming the map to adjust for such error.

7.1 Estimating the loop error

In order to estimate the loop error, the tracker must be able to recognize that is has returned to a previously visited direction. There are two possible approaches to this task: The features in the current camera image can be compared to features in the map, or an overlapping map can be created and features can be matched within the overlapping areas.

The first approach has the advantage that the current mapping method is sufficient, but creates the problem that the tracking could directly jump to the previously visited direction without closing the loop. More importantly, this method can only use the current camera features for loop closing.

The second approach was therefore decided upon. Since the map can only store one data item per pixel, an extended map that can deal with overlapping areas is required. The map is therefore enlarged to cover a horizontal angle larger than 360°. Our accuracy measurements suggest that an additional angle of 45° (4 columns of cells) is sufficient for robust loop detection. Hence, if loop closing is enabled the map covers a range of 405° (2304 pixels wide) horizontally.

Again our cell structure is applied for jump-starting the loop closing procedure: The loop closer starts making hypotheses as soon as only one column of cells is unfinished in the map, meaning that 393.75° have been mapped horizontally. Since the completion of cells is already monitored during the regular mapping process, this task does not create any overhead.

After the need for loop closing has been detected, keypoints are extracted from both overlapping regions (see Figure 8) and a RANSAC-like method for matching is used: For each keypoint on the left region, the best match is found in the right region using normalized cross correlation (NCC). Since loop closing is treated as a two-dimensional problem, a single match is sufficient for making a hypothesis. When a good match is found, it is checked to see how many other keypoint pairs supported this hypothesis. After all hypotheses have been checked, the one with the largest inlier set is used for calculating the mapping error: The offsets between all inliers are weighted by their NCC confidence to estimate a single, highly accurate mapping error.

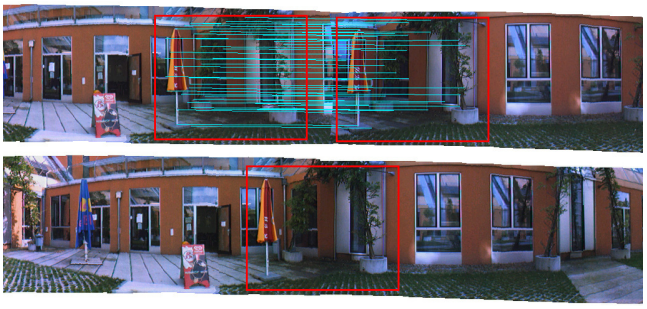

Figure 8. Loop closing; Top: Part of a 405° loop with overlapping areas. Bottom: Part of a same panorama closed to 360°.

7.2 Fixing the loop error

The accumulative tracking and mapping error, which finally results in the loop gap, arises from many sources, including imperfect orientation updates, camera calibration errors, violating the rotation-only restriction, etc. This error cannot be modeled and hence an optimal transformation for removing this error cannot be implemented using our system.

A simple transformation is therefore chosen that aligns the matched keypoints in the overlapping areas in such a way that the offset between keypoint pairs becomes minimal: In order to move the keypoints horizontally to their ideal position, the map is scaled in a horizontal direction. For vertical alignment a shear transformation is applied using as a pivot the cell column farthest away from the gap. Both operations use Lanczos filtered sampling to minimize resampling artifacts. In the tests this method accurately closed the loops without noticeable artifacts (Figure 8).

After the scale-shear operation, the map is cropped to 2048 pixel width (360°) and shifted horizontally, such that the original map starting position is at the map center again. Finally, new keypoints are extracted and tracking is initialized as described in section 6.

8 Results

We have created 30 panoramas at different indoor and outdoor locations. We specifically included difficult scenarios resulting in that 5 of the 30 panoramas could not be finished to a full 360° panorama since tracking broke in the process. Figure 9 shows examples of typical problems encountered in practice. The 25 complete panoramas were used for accuracy measurements.

8.1 Mapping accuracy

The visual tracker works at a camera resolution of 320x240 pixels. The map is created at a resolution of 2048x512 pixels. For

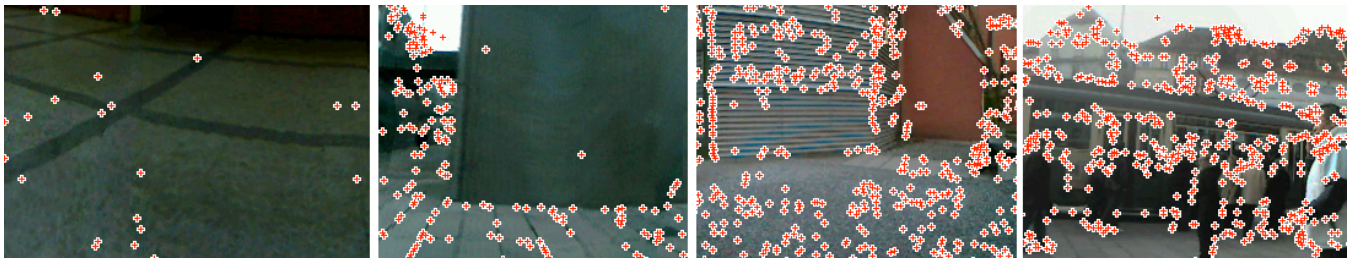

Figure 9. Problematic scenes: a) Floor that is poor on texture. b) Wall that is poor on texture, only line details on the floor. c) Line details on the wall (keypoints are due to sampling artefacts), pebbles on the floor. d) Moving objects (tram, people) covering most of the image.

a typical field of view of 60°, the camera resolution is therefore close to the map's resolution: 320 pixels / 60° · 360° = 1920 pixels. The theoretical angular resolution of the map is therefore 360° / 2048 pixels = 0.176 degrees per pixel. In practice, mapping errors accumulate, especially along the rotation around the vertical axis (yaw).

We evaluated the accumulated mapping error on the 25 full 360° panoramas and measured the offset in the overlapping regions using the method of section 7.1. We noticed that most panoramas had a horizontal error of +4°, which is obviously related to a systematic imprecision of our approach. However, by adapting the focal length in our calibration data we could shift the average error close to zero as can be seen in Figure 10. With the updated calibration 16 out of 25 panoramas have an error of ~1°.

We also analyzed pose jitter from live camera feed under zero motion, resulting in a measured jitter of ~0.05° for head, pitch and roll respectively.

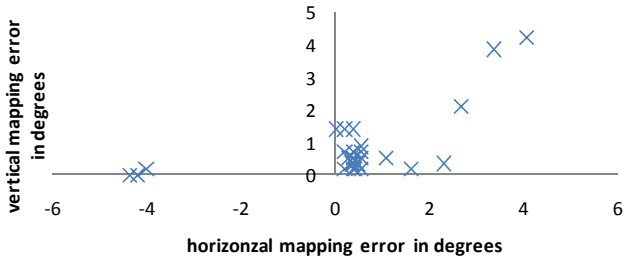

Figure 10. Mapping errors of 25 panoramas.

8.2 Speed on mobile phone and PC

The panoramic tracker was benchmarked on a 2.5GHz quad-core notebook (only a single core was used) and an Asus P565 smartphone with an XScale ARM CPU running at 800MHz. Since the mobile phone's CPU does not have hardware floating point support, a version with fixed-point math was created whereas the PC version runs using floating point.

	PC	*Phone*
Down sample	<0.1ms	0.5ms
Track low-res	0.5ms	1.9ms
Track mid-res	0.2ms	1.8ms
Track high-res	1.0ms	6.4ms
Map color (b/w)	0.4ms (0.4ms)	4.5ms (4.2ms)
Overall (b/w)	**2.2ms (2.2ms)**	**15.2ms (14.9ms)**

Table 2. Average tracking and mapping times per frame.

Table 2 shows the timings for PC and mobile phone for tracking the camera and updating the map for a new camera image. The values in Table 2 represent average durations per task while creating a complete map. Exact values vary depending on the environment (number of keypoints possible to track) and user behavior. E.g. if the camera only points to areas that have already been mapped then the mapping time goes close to zero.

Loop closing is an expensive operation taking 10 seconds (see Table 3) for a full color map on the mobile phone when using the Lanczos filter for sampling. Only about 50-100ms (PC) and 200-500ms (phone) are used for finding the loop gap; the rest of the time is spent on updating the map. If we use nearest neighbor filtering instead (creating noticeable artifacts, but still good for tracking), loop closing takes less than 2 seconds on the phone. The duration can be reduced further by only using grayscale maps. However, in practice loop closing is a rare situation since the creation of 360° maps takes ~1-2 minutes and is required only once for a map.

A considerable amount of the localization time goes into extracting features from a loaded map (usually several thousands). Fortunately, this is only required once, whereas all successive frames only need to extract features from the camera image, match then against the map features and find a pose, which takes ~40ms on the PC and ~120ms on the phone.

	PC	*Phone*
Loop-close Lanzos (b/w)	~290 (170) ms	~9800 (3900) ms
Loop-close NN (b/w)	~150 (130) ms	~1700 (1300) ms
Localize - first frame	~60ms	~190ms
Localize - further frames	~40ms	~120ms

Table 3. Average timings for loop closing: Lanzos and Nearest Neighbor (NN), color and grayscale; localizing from a loaded map.

8.3 Guidance for taking high-resolution panoramas

The proposed method for panoramic mapping and tracking internally generates color and grayscale panoramas of the environment. Such panoramas can be stored on disk and later viewed by users like normal photographs. Yet, the panoramas cannot compete in quality and resolution with a panorama that is generated offline on a desktop computer from several multi-megapixel photographs.

Since modern mobile phones integrate good-quality digital cameras, our tracking method is exploited for guiding users in capturing high-resolution photos of the environment. As a user taps on the screen of her mobile phone, the camera of the phone is triggered to capture a photograph and to later store it on disk. The tracking system, and the internally generated panorama, can be used for providing a live preview of the orientation of all photos that have already been taken, thus empowering users with a tool to verify optimal photo coverage of a panoramic scene (see Figure 11 left).

8.4 Augmenting annotations

Since the proposed method can create and track high-res panoramas on the fly, users are enabled to put annotations on such

Figure 11. Running at 30Hz on mobile phones, our method gives room for end-user applications. Left: Our method can be used to guide users in taking hi-res photos of the environment to be later stitched with a desktop application. The system can show a live preview of where photos have been taken, to ensure optimal coverage of the whole panorama. Right: Our method can be used for annotating panoramas and sharing them with other mobile users coming later in-situ, or with desktop users.

panoramas directly on a mobile phone. This can be useful in several application domains, such as posting geo-referenced messages or supporting navigation tasks, as well as exploring the panoramic image from desktop applications like Google Earth.

Users can identify the position of the annotation by clicking on the corresponding location of their device's screen. After the user has typed in the annotation's text, it is automatically referenced to the chosen position in the panoramic image. The application can then store all annotations and their map coordinates in a XML file, and save this file together with the panoramic image into a zip container. Since many of the current mobile phones ship with a location system (GPS or WiFi triangulation) a GPS tag is added to the created data and the zip file is sent to a server hosting GPS-tagged content.

If another user is exploring the same location at a later stage, the user's phone will send its current GPS position to the server and retrieve the zip file again. Our system is able to initialize the tracking using the panoramic image contained in the zip file, as described in section 6, and can therefore overlay the view with the annotations created by the previous user (see Figure 11 right).

Desktop applications can also access this geo-referenced content and embed the annotated panoramas on a virtual globe, similarly to the Street View images presented in Google Earth.

9 Conclusions

The paper presented an approach for accurate, robust and drift-free rotation tracking in outdoor scenarios. The method is efficient enough to run at high frame rates on mobile phones. As a side effect, the method creates colored, cylindrical panoramic maps with little user input. Creating GPS tagged panoramas with this method is a first step to allow user participation in creating tracking data for outdoor Augmented Reality. Furthermore it opens the way for other applications like user-annotated street-side views, which can be explored in-situ or on a desktop computer.

The presented method does not target the problem of auto-exposure yet. A work around exists by disabling auto-exposure; however, this is not possible on all mobile phones. Actively addressing this issue is therefore left as future work.

10 Acknowledgements

This work was sponsored in part by the Christian Doppler Lab for Handheld Augmented Reality and the European Union under contracts FP6-IST-27571 and FP6-IST-27731.

References

[1] A. Adams, N. Gelfand, and K. Pulli, "View Finder Alignment," *Computer Graphics Forum (Proceedings of Eurographics) 27*, 2008, pp. 597-606.

[2] P. Baudisch, D. Tan, D. Steedly, E. Rudolph, M. Uyttendaele, C. Pal, and R. Szeliski, "Panoramic viewfinder: providing a real-time preview to help users avoid flaws in panoramic pictures," *In Proceedings of the 17th Australia conference on Computer-Human Interaction: Citizens Online: Considerations for Today and the Future*, 2005, pp. 1-10.

[3] M. Brown and D.G. Lowe, "Recognising Panoramas," *Proc. of the Ninth IEEE International Conference on Computer Vision*, 2003, pp. 1218-1225.

[4] A. Davison, "Real-time simultaneous localisation and mapping with a single camera," *Proceedings of the Ninth IEEE International Conference on Computer Vision*, IEEE, 2003, pp. 1403-1410.

[5] S. DiVerdi, J. Wither, T. Höllerer, Envisor: Online Environment Map Construction for Mixed Reality. *In Proc. of IEEE VR 2008*, pp. 19-26.

[6] G. Klein and D. Murray, Parallel Tracking and Mapping for Small AR Workspaces, *In Proceedings of ISMAR'07*, 2007, pp. 225-234.

[7] Klein, G., Murray D., Parallel Tracking and Mapping on a Camera Phone, *In Proceedings of ISMAR'09*, 2009, pp. 83-86.

[8] D.G. Lowe, Distinctive image features from scale-invariant keypoints, *In International Journal of Computer Vision*, Vol.60, Nr. 2, pp. 91-110, 2004.

[9] J. Montiel and A. Davison, "A visual compass based on SLAM," *Proceedings of the 2006 IEEE International Conference on Robotics and Automation, ICRA 2006.*, IEEE, 2006, pp. 1917-1922.

[10] E. Rosten and T. Drummond, "Machine Learning for High-Speed Corner Detection," *In Proceedings of ECCV 2006*, Berlin/Heidelberg: Springer-Verlag, 2006, pp. 430-443.

[11] Schall, G., Wagner, D., Reitmayr, G., Wieser, M., Teichmann, E., Schmalstieg, D., Hofmann-Wellenhof, B., Global Pose Estimation using Multi-Sensor Fusion for Outdoor Augmented Reality, *In Proc. ISMAR'09*,, 2009, pp. 153-162.

[12] D. Steedly, C. Pal, and R. Szeliski, "Efficiently Registering Video into Panoramic Mosaics," *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, IEEE, 2005, pp. 1300-1307.

[13] R. Szeliski and H. Shum, "Creating full view panoramic image mosaics and environment maps," *Proc. of the 24th annual conference on Computer graphics and interactive techniques (Siggraph 1997)*, 1997, pp. 251 - 258.

[14] R. Szeliski, "Image alignment and stitching: a tutorial," *Foundations and Trends in Computer Graphics and Vision*, vol. 2, 2006, pp. 1 - 104.

[15] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg, "Pose Tracking from Natural Features on Mobile Phones," *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, 2008, pp. 125-134.

[16] X. Wang, M. Tico, and K. Pulli, "Panoramic Imaging System for Mobile Devices," *Poster at the 36th international conference and exhibition on Computer graphics and interactive techniques (SIGGRAPH 2009)*, 2008.