Global Localization from Monocular SLAM on a Mobile Phone

Jonathan Ventura, *Member, IEEE*, Clemens Arth, *Member, IEEE*,
Gerhard Reitmayr, *Member, IEEE*, and Dieter Schmalstieg, *Member, IEEE*

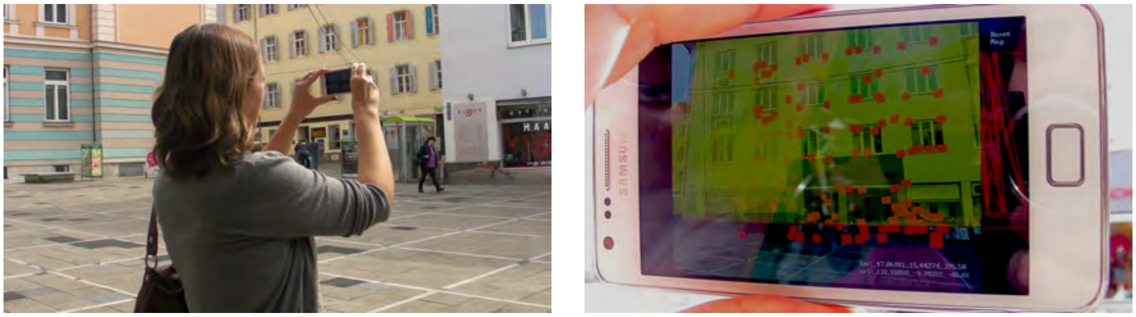


Fig. 1: From a SLAM map created on a smartphone (*left*), the system determines an accurate global device localization which can be used for augmented reality overlays (*right*). Here, a geo-registered city model is rendered in yellow on the building facade, with SLAM map points in red. The edges of the model are visibly aligned with the building extents.

Abstract—We propose the combination of a keyframe-based monocular SLAM system and a global localization method. The SLAM system runs locally on a camera-equipped mobile client and provides continuous, relative 6DoF pose estimation as well as keyframe images with computed camera locations. As the local map expands, a server process localizes the keyframes with a pre-made, globally-registered map and returns the global registration correction to the mobile client. The localization result is updated each time a keyframe is added, and observations of global anchor points are added to the client-side bundle adjustment process to further refine the SLAM map registration and limit drift. The end result is a 6DoF tracking and mapping system which provides globally registered tracking in real-time on a mobile device, overcomes the difficulties of localization with a narrow field-of-view mobile phone camera, and is not limited to tracking only in areas covered by the offline reconstruction.

Index Terms—Image-based localization, monocular SLAM, real-time tracking, global positioning, mobile augmented reality

1 Introduction

Augmented reality interfaces rely on pixel-accurate and ubiquitously available pose estimation. To achieve this, vision-based methods for global localization are the most promising, because of their high precision in estimating the camera pose, and their ability to run in real-time on mobile phone hardware. "Global localization," while sounding like an oxymoron, means localization, or determining the position and orientation of a device, in a global reference frame. State-of-the-art approaches to real-time global localization track a pre-made point cloud [38] or edge [31] model of the environment, which could be registered to a useful global reference frame, such as the floorplan of a building or a geographic coordinate system such as a UTM zone.

There are several problems with directly tracking the globally-aligned point cloud. Typically, the point cloud needs to be stored on the device to achieve real-time performance, which introduces problems in data transfer, storage, and maintenance. In terms of the interface experience, the user may have to physically search with the camera for some time, until a proper viewpoint is found, from which the system can localize. The localization estimate itself may then take several seconds to compute, further adding to the system startup cost. Another issue is that the global point cloud is made in a offline process, and thus becomes outdated when the environment geometry, appearance or illumination changes.

In this work, we solve these problems with a novel system concept, which combines a client-side monocular simultaneous localization and mapping (SLAM) system with a server-side map registration process to achieve wide-area localization on a mobile device. We see our approach as the logical next step after the previous work of Arth *et al*. [1], which used rotation-only tracking and mapping on a mobile client to create a panorama, while iteratively computing the globally-referenced device location in a background process. In our work, a SLAM system provides rotational and translational pose tracking and mapping on the mobile client, while a networked server receives keyframes from the client and sends back an estimated global registration. As keyframes are added, the registration is recomputed and potentially improves given the added information. Anchor points and their observations are integrated into client-side bundle adjustment to further refine the alignment and control drift.

Because the SLAM system maps and tracks the environment without any prior environment knowledge, the system can begin operating immediately in a local reference frame, without waiting for a successful global localization. Furthermore, the system performs camera tracking based on a current visual map of the scene, which allows for globally-referenced tracking even when parts of the scene have changed since the offline model was captured.

Our contributions are as follows: 1) A novel system concept for global localization on smartphones, which combines local SLAM and global map registration; 2) A system design which allows for parallel operation of the localization process on the server, while the client-side system is free to continue expanding and refining the map lo-

- *Jonathan Ventura is with Graz University of Technology. E-mail: ventura@icg.tugraz.at.*
- *Clemens Arth is with Graz University of Technology. E-mail: arth@icg.tugraz.at.*
- *Gerhard Reitmayr is with Graz University of Technology. E-mail: reitmayr@icg.tugraz.at.*
- *Dieter Schmalstieg is with Graz University of Technology. E-mail: schmalstieg@tugraz.at.*

Manuscript received 12 September 2013; accepted 10 January 2014; posted online 29 March 2014; mailed on 1 May 2014.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

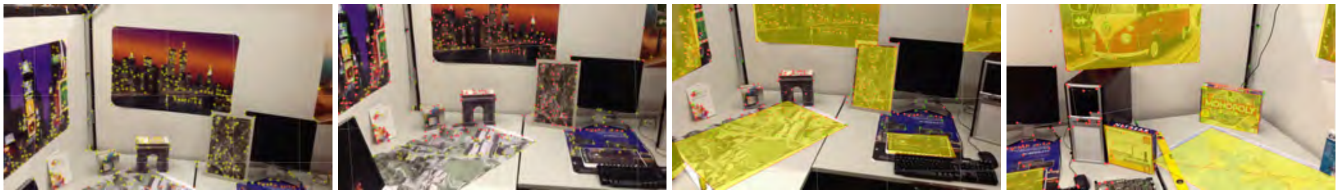

(a) 0.2 seconds (b) 3.6 seconds (c) 6.6 seconds (d) 22.3 seconds

Fig. 2: Illustration of system stages using an indoor video processed with our system. (a) Tracking starts immediately in the initialization phase. (b) When the camera moves far enough, the map (in pink) is created and the first two keyframes are sent to the server. (c) When the localization result is received, global annotations are shown. (d) Additional keyframes expand the map and refine the localization.

cally; 3) An iterative matching and registration scheme which allows for much of the information needed for localization to be cached on the server; 4) Evaluations of our system, in both indoor and outdoor environments, including timing and positional accuracy measurements. Our evaluations highlight the advantages of SLAM localization over a model-based approach which uses single-image localization on each frame.

2 Related Work

The most common approach for global registration of a mobile camera is to rely on other sensors such as GPS and compass to determine the device position and orientation. For example, the MARS backpack-mounted system used a worn differential GPS unit for positioning [13]. Augmented reality apps today often use the built-in GPS and compass, but this only provides coarse positioning (within tens of meters), which is not suitable for pixel-accurate graphical overlays, and not available indoors.

Many approaches exist for localization from a single image, in some cases only with accuracy comparable to consumer GPS [42, 32, 34, 40, 36, 37, 5] and often without regard to the subsequent tracking phase [17, 22, 20, 7, 21]. For our system, we chose a relatively simple method of direct matching from image features to 3D points, and our work is focused on the integration of image-based localization into a flexible, real-time tracking and mapping system which is suitable for implementation and use on a handheld device.

Visual SLAM systems use the camera itself to determine device position, by tracking and mapping detectable features in the surrounding environment. Davison *et al.* [10, 11] were the first to propose monocular SLAM using a filtering approach. Klein and Murray proposed keyframe-based SLAM [18]. In keyframe-based SLAM, keyframes are sampled from the camera and processed in a background thread to produce a point cloud reconstruction (the map). In parallel, the current camera image is tracked using the map. By triangulating and tracking thousands of points, the system provides accurate pose estimation across a wide range of viewpoints in the scene. In general, monocular SLAM provides high accuracy camera tracking in real-time, and is even capable of running on mobile phone platforms [19]. However, the camera pose is only given in a local reference system, defined with respect to the first camera frame or an initialization target. [9]

In the context of SLAM systems, the process of detecting an overlap between the current map and a pre-existing map, and then estimating the registration between the two maps, is called “loop closure.” Typically, loop closure is used to detect overlaps within a single SLAM map, for example when the path of the camera crosses over itself. For example, Cummins and Newman describe appearance-only SLAM, which builds a topograhic map based on visual loop closures [8, 9]. In our case, we are interested in detecting the overlap of the entire local map with some part of a larger, pre-made global map. Some previous SLAM systems with loop closure (*c.f.* [12, 39]) have a similar concept to our system, namely fast initialization in a local map followed by loop closure with an existing map. However, we are the first to apply this system concept for wide-area localization on mobile phones, and our system design addresses the issues which arise from having limited computational resources on the client device, an asynchronous server-client implementation, and a very large feature database (millions of features).

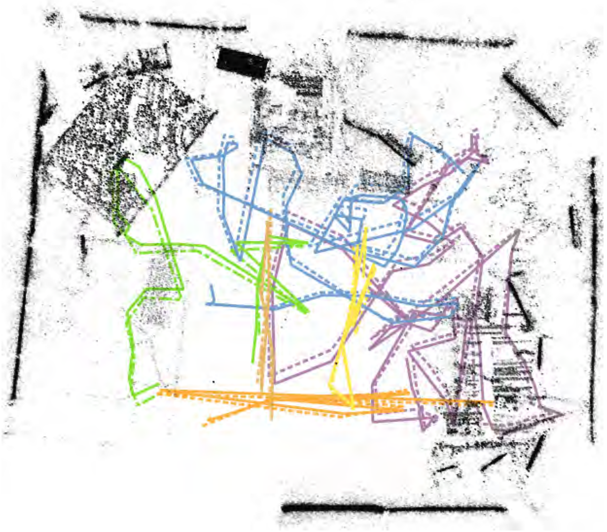


Fig. 3: Camera keyframe paths estimated by SLAM localization (solid lines) and ground truth from the ART-2 optical tracker (dashed lines) for six indoor image sequences.

An alternative to online mapping with SLAM is model-based tracking, *i.e.*, tracking directly from a pre-made environment model. Reitmayr and Drummond [31] present a system for edge-based tracking from textured polygonal building models. Arth *et al.* achieved real-time rates on a mobile phone while localizing each camera frame against a point cloud scene model [4]. Takacs *et al.* describe a system for real-time localization with respect to geo-registered panoramas, but are restricted to continuous tracking from the visible dominant plane [35]. Ventura and Höllerer separated localization and tracking in a client/server system for real-time, general motion tracking in outdoor environments using a pre-made point cloud model [38]. Lim *et al.* propose amortizing the cost of feature recognition across several frames, aided by feature tracking, so that localization and tracking can both take place on the client [23]. The limitation of these model-based tracking approaches is that the online tracking cannot extend beyond what was captured in the offline model.

A few previous works also use some combination of mapping and tracking with global localization. Arth *et al.* use visual orientation tracking on a client and perform 6DoF localization from the resulting panorama as a background task [1, 2]. While the system tracks pure rotational movement of the camera, translational movement is not supported. Lothe *et al.* register a SLAM map captured from a moving vehicle to a polygonal 3D city model, but require an initialization provided by GPS or manual input [24]. Oskiper *et al.* use a stereo visual odometry system for relative motion estimation, and compute global registration corrections from matches to a global feature database [28, 29]. This approach is conceptually similar to our system, in that a client-side tracking result is corrected by matching to a global reconstruction. However, the advantage of our system is that, once localized, the client can continually track from the local SLAM map without any overhead of global feature matching on the server. The visual odometry approach of Oskiper *et al*. does not maintain a persistent local map, and thus requires constant global feature matching to avoid drift.

Castle and Murray describe an approach for detecting and localizing multiple small objects while using a keyframe-based SLAM system [6]. However, their system would not be suitable for global localization, because each localization based on a small detected object would give a noisy global localization estimate, and some method of reconciling the conflicting estimates would be needed. In our work, we adopt a similar method for multi-view object detection and localization, which uses feature matching to the global database, point triangulation and registration. To estimate an accurate and stable global localization, we treat the entire surrounding environment as the single object to be detected and localized, and update and refine the localization as keyframes are added.

3 System Overview

The client side of our system consists of a keyframe-based SLAM implementation which is capable of running at real-time rates on a smartphone. As in the work of Klein and Murray [18], the SLAM system samples keyframes from the video stream and performs bundle adjustment in a background thread to optimize the keyframe camera poses and the 3D locations of triangulated points. At the same time, the system tracks the estimated point cloud (the map) in each camera frame in real-time, providing live pose estimation, which can be used for augmented reality rendering. To initialize the map, we use the method of Mulloni *et al*. [27], which refines an initial planar model of the scene, and thus can begin tracking immediately from the first frame.

The server process has access to a point cloud reconstruction of the target environment, which has been created beforehand, offline. Many previous works consider how to create such point cloud reconstructions from single-camera videos [33] or panoramic videos [38]. For our outdoor tests, the point cloud is registered to a global coordinate system by incorporating GPS measurements [3] and directly aligning to a geometric model of the environment [24] provided from public sources.

During and after tracker initialization, but before any localization result has been computed, the mobile client operates in a local reference system, defined with respect to the first camera frame. In this state, global augmentations can be displayed with low accuracy, if GPS, inertial sensors and compass are available. It is also possible in this state to create and display annotations which are stored in the local coordinate system. Once at least two keyframes are available, the system attempts to compute a global localization of the keyframes using their estimated relative poses and matches with the server-side point cloud (see Section 4). If successful, the mobile client begins operating in the global coordinate system, so that augmentations are displayed more accurately, and the locations of annotations added by the user can be transferred to the global coordinate system. Sample system output images and localized camera paths from our indoor experiment (Section 6) are shown in Figures 2 and Figure 3, respectively.

4 Localization Process

The localization process is illustrated in Figure 4. 1) After initialization, the first two keyframes are matched to the global point cloud on the server. 2) These matches and the keyframe poses are used to triangulate corresponding local points, from which the system attempts to determine a 7-DoF transformation (translation, rotation and scale) which determines the global poses of all the keyframes. 3) Matches from subsequent keyframes are used to triangulate more points and improve the absolute orientation estimate. 4) Client-side bundle adjustment optimizes the localization solution as more keyframes and points are added. These steps are discussed further in the following subsections.

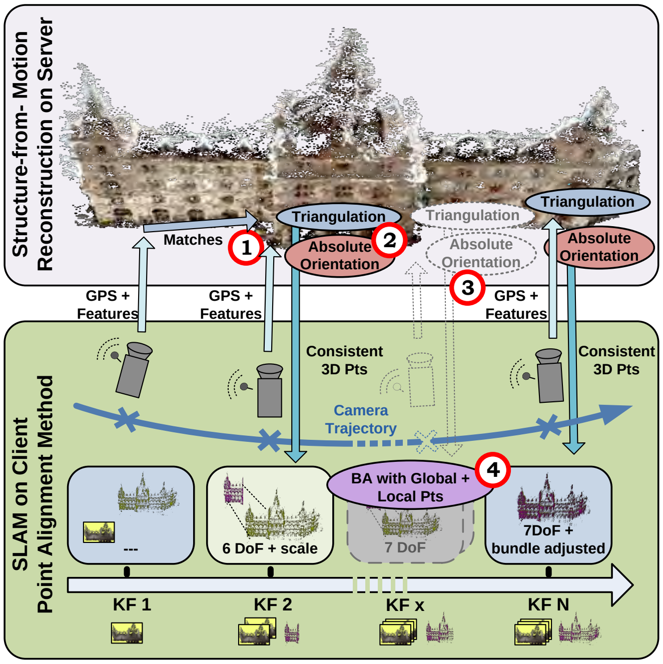


Fig. 4: Iterative localization process: 1) Feature matching; 2) Point triangulation and registration; 3) Integration of additional keyframes; 4) Client-side bundle adjustment with anchor points.

4.1 Feature Matching

From each keyframe, SIFT features [25] are extracted and matched to the global points. Each global point is represented by the mean descriptor of all corresponding features in the offline reconstruction. A randomized kd-forest [26] is used for fast, approximate nearest neighbor search. We apply the second neighbor ratio test [25] to remove ambiguous matches and ensure that each point matches to at most one feature in each keyframe.

We denote here the N global points as X_i, where $i \in 1 \ldots N$, the M keyframes as K_j, where $j \in 1 \ldots M$, and the F_j features in keyframe K_j as $f_{j,k}$, where $k \in 1 \ldots F_j$. If feature $f_{j,k}$ is matched to X_i, then $f_{j,k} \rightarrow X_i$. As depicted in Figure 5, each global point maintains a list of matched features from the keyframes, $L_i = \{f_{j,k} | X_i \rightarrow f_{j,k}\}$. When a new keyframe is added, its feature matches are added to the end of the appropriate lists.

4.2 Point Triangulation

For each global point X_i, we take its aggregated set of matched features L_i and attempt to find a corresponding local point $X_i^\prime$ which fits the observations. This triangulation is computed in the local SLAM reference frame by using the keyframe camera poses as estimated by the SLAM system. Because the number of matches per point is expected to be low, we use an exhaustive pairwise search over all features in L_i to find a consistent triangulation. For each pair of features matched to a global point, we compute a linear triangulation [15]. Any observation with a re-projection error of less than five pixels is considered an inlier, if the point lies in front of the camera. The point triangulation with the highest number of inliers is chosen as the best triangulation, which is refined by non-linear minimization of the re-projection error over all inlier measurements [15]. Any triangulation with at least two inliers is accepted into the next stage of processing. This allows for some inaccurate points to be accepted, but these outliers are handled by robust estimation in the next stage.

We fully re-compute all triangulations after each keyframe addition, since additional matches might lead to new point triangulations, and the keyframe poses may have been updated by the client-side bundle adjustment between keyframes. Since the point triangulations are independent calculations, they can easily be computed in parallel.

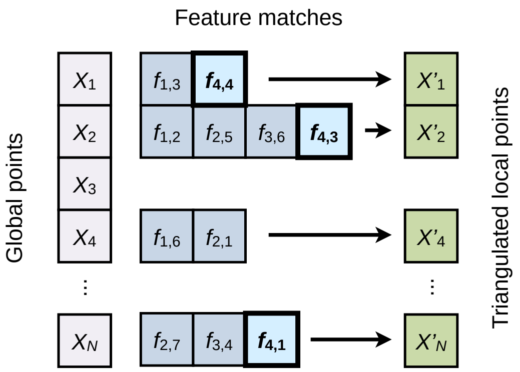


Fig. 5: Illustration of feature matching and triangulation. Each global point, X_i (left) maintains a list of matched keyframe features $f_{j,k}$ (middle) and a triangulated local point X_i' (right) which fits the feature observations. When a new keyframe is added, matched features are added to the lists (in bold) and the local point triangulations are updated.

4.3 Point Registration

Given all corresponding global and local points X_i and X_i', we now compute the similarity transform T which aligns the two point sets, using the absolute orientation method of Horn [16]. Because some points may have a poor triangulation due to noisy measurements or mismatches, we use a RANSAC loop [14] to robustly estimate the alignment transform. The RANSAC loop iteratively samples a minimal set S of point indices and computes the similarity transform which minimizes the point-to-point distance error metric:

$$e = \sum_{i \in S} ||X_i - TX_i'||_2. \tag{1}$$

In a typical RANSAC scheme, the residual error for each correspondence is thresholded to label inliers and outliers, and the sample with the highest number of inliers is selected as the best solution. However, in the case of point alignment, it is unclear how to correctly select an error threshold which separates inliers from outliers, because the threshold depends on the scale of the point cloud and the noise in the data. To address this problem, Raguram and Frahm proposed the Residual Consensus (RECON) algorithm, which adaptively selects an inlier threshold by looking for a consensus set of inliers [30].

In our case, we note that we can avoid the difficulties in choosing a 3D distance threshold by using a different criterion for choosing the best RANSAC sample. Instead of counting the number of inliers based on the point-to-point error metric, we project the global points into the keyframes and compute the re-projection error for all putative observations:

$$resid_{i,j,k} = ||\text{proj}(P_j T^{-1} X_i') - f_{j,k}||_2 \tag{2}$$

where P_j is the camera matrix for keyframe K_j. The re-projection error is invariant to the scale of the reconstruction, and allows us to choose an inlier threshold in pixel units. In our experiments, we used a threshold of five pixels. All feature matches are tested, not just those which were successfully triangulated. In our experiments, we found that this method of sample selection gave a higher number of inliers in successful cases, which made it easier to determine when the localization result should be accepted – we used a threshold of twenty inliers. Furthermore, this method produced more stable results across different datasets, in comparison to manually tuning the point-to-point error threshold for each dataset.

5 Asynchronous Implementation

In general, localization is not expected to be computed within the time of one frame capture (33 milliseconds), and the computation is actually on the order of seconds (see Section 6). For this reason, we must compute the localization asynchronously to allow for continuous tracking. This means that the localization will work with map information which might be outdated by the time the result is ready.

5.1 Caching

The server keeps a cache of received and computed information, to avoid unecessary re-computation. The server stores extracted features and descriptors for each keyframe and matches between the features and global points. As described in Section 4.1 the matches are stored in an associative list structure, indexed by global point. These lists are cached on the server between localization requests, since the feature matching is independent of any further localization result and needs to be computed only once per keyframe.

When a new keyframe is added by the SLAM system, the client only needs to send the new keyframe's image data (compressed using JPEG) and the current estimated poses for all keyframes. It is necessary to inform the server of all current pose estimates, because these might have been updated by the client-side bundle adjustment since the previous localization request.

5.2 Iterative Estimation

Once the estimated localizing transformation is received by the client, the question arises of how to use this transformation in the SLAM system. One approach is to directly transform the map points and keyframes. However, the asynchronous nature of the system makes this problematic, because of the need to keep all map copies consistent, including those which are used by queued and in-process localization requests.

We chose an alternative approach: instead of directly transforming the map, we store the localization transform T separately and keep the SLAM map in the local reference frame. When a new localization transform received, the system simply replaces the old transform with the new estimate. The inverse transform T^{-1} is used to bring globally-referenced content into the local reference frame for rendering.

One unavoidable consequence of our asynchronous design is that the localization server works with potentially outdated keyframe pose estimates. During the localization computation time, more keyframes may be added and local or global bundle adjustment may have changed the keyframe pose estimates. This means that the received localization transform may not be the optimal alignment for the current map. Our assumption is that the map will not change so substantially that this would cause an issue, since the system keeps the first keyframe pose and the map scale constant.

5.3 Bundle Adjustment

After computing the optimal similarity transformation, inlier 2D-3D observations of global points are used for further non-linear refinement of the localization. This adjustment could be performed on the server, but, while the server is processing, the client may be adding keyframes and also adjusting the map in parallel. In order to avoid conflicts and merges between the server-side and client-side maps, we chose to instead integrate the global point observations into the client-side bundle adjustment process.

In addition to the localization transform T, the server also sends the set O of all point-feature pairs which were counted as inliers. Because the SLAM system always operates in the local reference frame, the bundle adjuster uses the current inverse transform T^{-1} to transform the global points into the local reference frame. The global points are kept constant in the bundle adjustment and act as "anchor" points which help to eliminate drift in the SLAM map. The combined error term minimized in bundle adjustment is:

$$\begin{aligned} &\sum_{(i,j,k) \in O^L} ||\text{proj}(P_j X_i^L) - f_{j,k}^L||_2^2 \\ + &\sum_{(i,j,k) \in O^G} ||\text{proj}(P_j(T^{-1} X_i^G)) - f_{j,k}^G||_2^2 \end{aligned} \tag{3}$$

where the labels L and G are used to identify local information from the SLAM map and global information from the localization process,

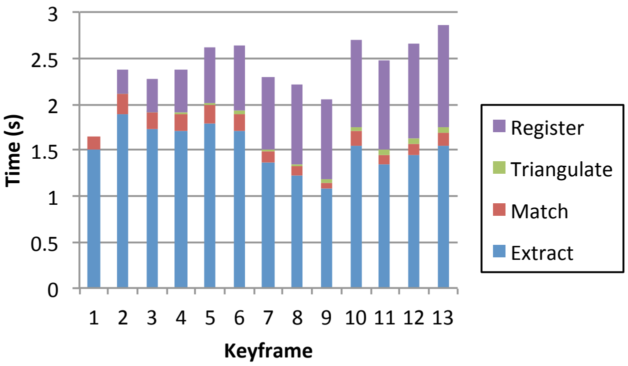


Fig. 6: Localization timing for indoor point cloud (132,994 points). For each keyframe, the system *extracts* SIFT features, *matches* them to the point cloud, *triangulates* the matches into local points, and *registers* the local points to the global point cloud in a RANSAC loop.

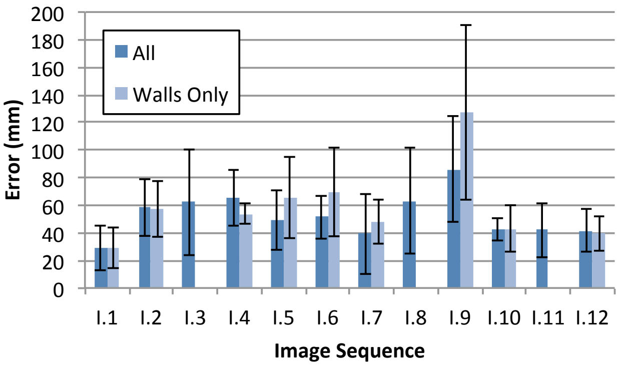


Fig. 7: Average keyframe localization accuracy for all indoor sequences, using the ART-2 optical tracker for ground truth measurements.

respectively.

5.4 Server Design

For small datasets (such as our indoor dataset, described in Section 6), it is feasible to maintain one kd-tree index for all descriptors, which is loaded when the server starts. In the outdoor case (Section 7), we have reconstructed a total area of about 10,000 m^2 which resulted in a descriptor database of 3.9 GB. At this size it is still possible to load and store all descriptors in memory; however, as we increase the size of the database, nearest neighbor performance decreases and query time increases. Given that our reconstruction produces about 900 descriptors or 450 KB per meter squared, we can extrapolate that for the entire city center area of 763,730 m^2 we would produce 327.3 GB of descriptors, and for the entire city of 64.19 km^2, we would produce 26.44 TB. To allow the system to scale to such large datasets, we apply two partitioning techniques to reduce the set of points under consideration for one client-server localization session.

We apply cell partitioning to the point cloud. In our case, we manually separated out for three courtyard areas in the city. Visibility partitioning [4] or regular or hexagonal grids [1, 41] could alternatively be used for automatic spatial partitioning. When a client starts a localization session, it sends its estimated latitude and longitude (measured by GPS), which is used to select the appropriate cell.

Additionally, a single cell partition is divided into eight orientation slices [2]. One orientation slice contains all features whose viewing direction is within the slice. We used eight slices spaced 45 degrees apart, with an overlap of 30 degrees on either side, so that each slice covers an angular area of 105 degrees total. In each slice, all descriptors for the same point are averaged together, so that each point has one average descriptor. The client sends a compass reading which is used to select the appropriate slice of points from the partition.

When the first keyframe is received from the client, the appropriate slice from a single partition is loaded into memory along with its kd-tree nearest neighbor index. Because we keep the point cloud slices and kd-tree indexes as serialized files, startup time is negligible.

6 Indoor Evaluation

To test our system in an indoor setting, we built an office cubicle scene in a 3×4 meter space with walls on all four sides and desks on three sides. Visual texture was added by placing posters on the walls and desks as well as other textured 3D objects (see Figure 2). An ART-2 infrared optical tracking system was mounted at the corners of the room. The optical tracking target was attached to the back of an Apple iPad 2 tablet.

To create a global reconstruction of the scene, we captured a long video sequence with the tablet, observing the scene from many viewpoints and distances. This video was subsampled to about 400 images which were then processed in a large-scale structure-from-motion pipeline. The external position measurements were integrated into the reconstruction pipeline, to create an aligned point cloud in meter units. The indoor reconstruction contains 132,994 points and 1,261,918 descriptors (616 MB). Figure 3 shows a visualization of the point cloud. We manually created 3D annotations, consisting of quads around various posters and boxes in the scene, in the global reference system of the aligned point cloud.

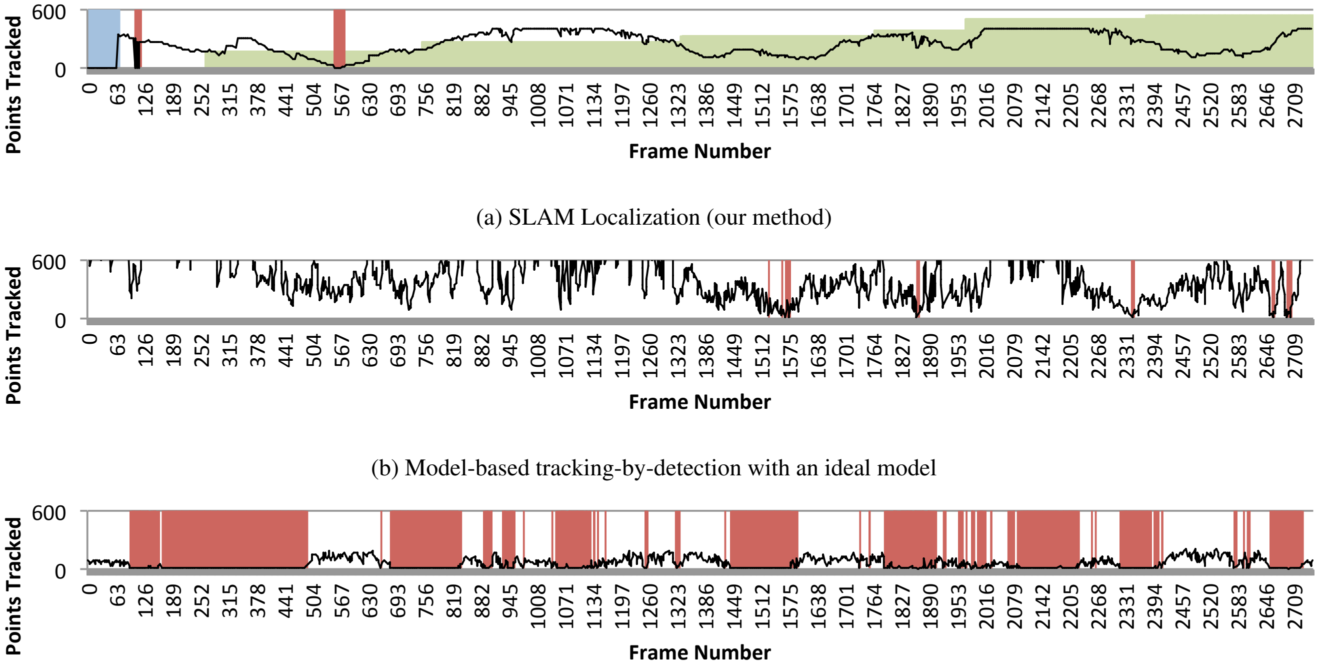
We also captured twelve test video sequences, named I.1–I.12, with the tablet, covering many different viewpoints in the scene. These sequences were then processed in our system, with the client and server separated on different computers. Six of the resulting localized camera paths are visualized in Figure 3 with a top-down view of the room. Frames from sequence I.2 are shown in Figure 2.

A Macbook Pro with a 2.8 GHz Intel Core i7 dual-core processor and 16 GB RAM was used for the localization server. A breakdown of timing for different localization tasks is shown in Figure 6. Feature extraction and matching times are roughly constant per keyframe, taking, on average, 1.53 and 0.15 seconds, respectively. We used the single-core SIFT extraction implementation of OpenCV, and so we expect that this step could be made significantly faster with a multi-core or GPU implementation. Point triangulation time increases as more matches are added, but the computation time is negligible compared to other tasks (0.03 seconds on average). Point registration time also increases as more matches are added, from 0.26 seconds with two keyframes, up to 1.10 seconds with all thirteen keyframes. The point triangulation and registration steps were parallelized, but faster performance could be achieved by adding more cores or using a GPU.

Arguably, the most important timing measure is the “time-to-first-localization” – the aggregate amount of time taken from system startup to the first usable localization result produced by the system (when the success threshold on number of inliers is exceeded). This time consists of both the SLAM initialization period and the time taken to match and localize at least the first two keyframes. We recorded an average initialization time of 4.6 seconds, and an average localization time of 4.8 seconds, for a total average time-to-first-localization of 9.4 seconds.

We also investigated localization performance when parts of the scene are missing from the reconstruction. We tried to cover every possible viewpoint with our reconstruction video. However, in a practical setting, some parts of the scene may be missing from the reconstruction; for example, we might have a reconstruction of a well-photographed landmark, but not the areas around it. Also, the modeled scene may have changed since the time of capture, or there may be occluding objects in the current environment which were not covered in the offline reconstruction.

To test SLAM localization in these scenarios, we made a partial



(a) SLAM Localization (our method)

(b) Model-based tracking-by-detection with an ideal model

(c) Model-based tracking-by-detection with a non-ideal model (walls, but not tabletops)

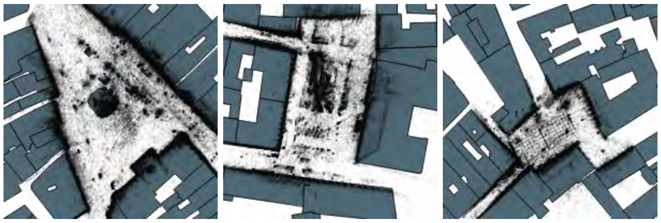
Fig. 8: Number of points tracked (in black) in image sequence I.2 using different tracking methods. The red areas indicate tracking failures (less than twenty points tracked), and the blue area indicates SLAM initialization. The green bars indicate SLAM localization inliers using the walls-only model.

reconstruction by removing a part of the point cloud. We removed everything on the tables from the point cloud and left only the posters on the walls. This simulated the scenario when the objects on the walls have been reconstructed, but the objects on the desks have been moved or are occluded. In an office setting, this scenario has practical value since typically the objects on the desks will be moved, but the decoration on the walls will stay fixed for long periods of time, and thus could be reconstructed once.

Figure 7 plots the average keyframe localization accuracy for each image sequence. Position measurements from the optical tracking system were used as ground truth. The average localization accuracy using the full point cloud ranged from 29 to 86 mm. The accuracy using the partial point cloud was similar, ranging from 29 to 127 mm. In three out the twelve sequences, not enough inliers were found to provide a localization result with the partial model, because these sequences did not provide enough views of the walls to find sufficient matches.

As a point of comparison, we computed a *P*3*P* [14] localization result for each frame in sequence I.2, using either the full point cloud or the partial point cloud. This corresponds to a version of model-based tracking, where each image is directly localized using the point cloud. The per-frame localization time is not considered for our purposes, and these tracking results were computed offline. Thus this test does not evaluate a practical real-time tracking system, but instead is a representation of ideal model-based tracking performance.

The number of points tracked using the SLAM system and model-based tracking is shown in Figure 8. The red bars indicate when the system fails to provide a camera pose, because too few points are tracked, using a threshold of twenty points. The SLAM system creates and tracks its own map, and so its behavior is independent of the global point cloud. However, the model-based tracking fails for several long periods of time when using the partial point cloud. This is because model-based tracking can only work when the area being viewed is in the global reconstruction. The SLAM system, in contrast, can bridge tracking across parts of the scene which are missing from the global reconstruction. The green bars in Figure 8(a) indicate the number of localization inliers using the partial model. The number of inliers starts at about 100 inliers with the first two keyframes and increases as more keyframes are added.



(a) Area *A* (b) Area *B* (c) Area *C*

Fig. 9: Reconstructed areas in the city with building outlines and point cloud (black dots).

7 Outdoor Evaluation

To evaluate our system in an urban environment under realistic capture conditions, we prepared reconstructions of three large courtyards in the city center of Graz, Austria. We denote these areas as *A*, *B* and *C*. The areas were visually mapped with a backpack-mounted Point Grey Ladybug spherical panorama camera. A differential GPS unit was attached to the top of the camera. Panoramic images from the capture session were processed in a structure-from-motion pipeline. After reconstruction, we robustly aligned the camera centers to the GPS measurements, and then further aligned the reconstruction to a polygonal city model extracted from survey data provided by the regional government. This model was created by combining 2D building outlines with aerial laser scans of the city to estimate ground and roof geometry. The alignment process resulted in highly accurately geo-registered 3D point clouds, specified in metric coordinates using

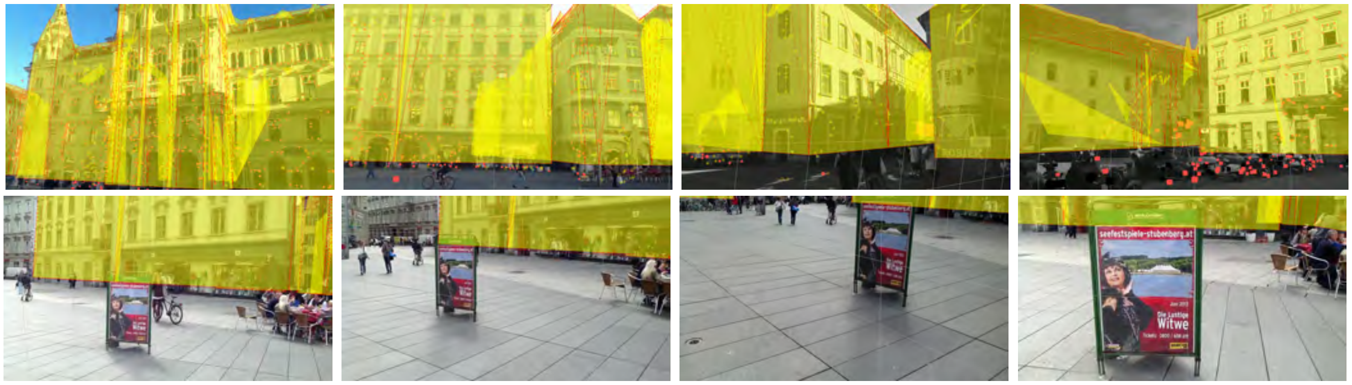


Fig. 10: *Top:* Localized images from different areas in the city center. *Bottom:* Localized sequence showing how the SLAM system can track through partial or full occlusion of the global model. Here, the poster is missing from the global reconstruction, but it is mapped and tracked locally by the SLAM system.

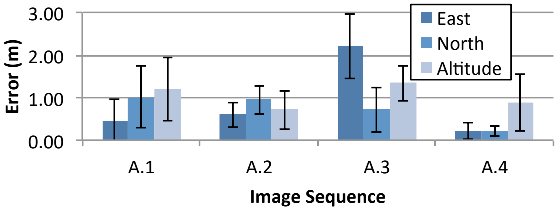


Fig. 11: Average keyframe localization error for four outdoor sequences, using a differential GPS receiver for ground truth measurements.

Seq.	Inliers	Average Error (m)			Standard Deviation		
		East	North	Alt.	East	North	Alt.
A.1	254	0.45	1.01	1.19	0.53	0.73	0.73
A.2	309	0.60	0.95	0.71	0.29	0.33	0.45
A.3	37	2.21	0.72	1.34	0.76	0.52	0.42
A.4	468	0.21	0.22	0.89	0.19	0.13	0.67

Table 1: Keyframe localization accuracy using DGPS ground truth.

the Universal Trans-Mercator (UTM) coordinate system. Figure 9 illustrates the point clouds (in black) overlaid on a 2D city map with polygonal building outlines.

The polygonal city model is displayed as a virtual annotation in our localized sample images (Figures 1 and 10). This provides a qualitative means of evaluating registration accuracy and the potential of our system for use in augmented reality applications.

The SLAM system and localization client were implemented using the Android mobile operating system. We used a Samsung Galaxy S II for live tests of the system. In each mapped area, we performed three live tests by initializing and localizing a SLAM map of the surrounding buildings. We used a 3G cellular wireless connection to send keyframe images to the localization server. The GPS and compass reading sampled at initialization time was transmitted to the server to select the appropriate slice from the feature database. Figure 1 shows sample images of the smartphone localization system in use. These tests were performed under weather conditions similar to the model capture.

For each live smartphone test, we recorded the time-to-first-localization. The average initialization time, over nine samples, was 12.6 seconds and the average localization time was 5.3 seconds. In total, the average time-to-first-localization outdoors is 17.9 seconds – about twice that of our indoor test. However, this time is clearly dominated by the SLAM initialization procedure, while the actual time spent computing a localization is roughly the same as the indoor case. The initialization procedure takes more time outdoors because the distance to the scene is greater, and thus a longer distance needs to be traversed before the system can confidently triangulate points and initialize the map. Generally, we needed to walk about two meters to initialize the system, with a distance to the buildings of about twenty meters. During initialization, we walked parallel to the dominant building facade in all tests, with the camera directly facing the building facade.

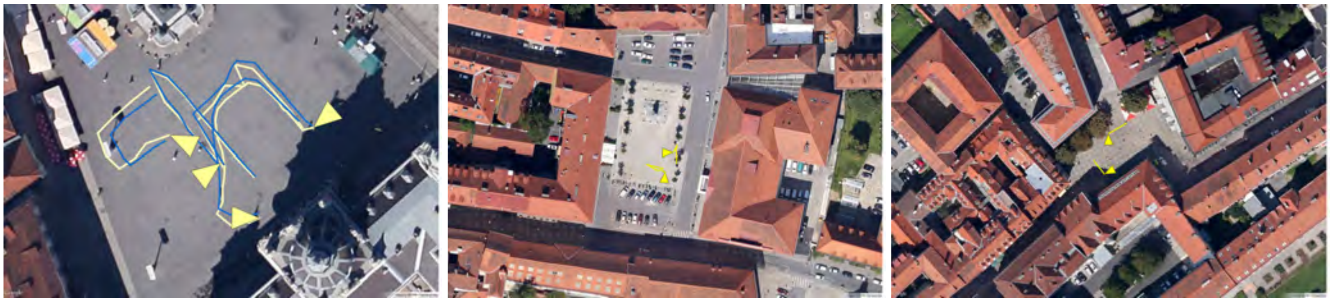
To determine the extent that using a 3G cellular connection affects the localization time, we measured the time it takes to prepare and send the localization request from the smartphone using either 3G or WiFi. This includes the time taken to compress the 640 × 480 keyframe image using JPEG compression and to send the image data along with all keyframe poses. Using 3G, we recorded an average request time of 802 milliseconds over nine trials, with a standard deviation of 90 milliseconds. Using WiFi, the average request time was 226 milliseconds with a standard deviation of 28 milliseconds.

We also recorded four videos in area *A* using the iPad tablet. A differential GPS receiver was attached to the top of the device, which provided a ground truth measure of position for testing the accuracy of the localization system. The results for the test videos are graphed in Figure 11 and numbers are given in Table 1. Sequence A.4 had the most inliers and the lowest error: less than 25 cm error in the east and north directions, and less than one meter error in altitude. Sequence A.6 had the fewest inliers and also the highest positional error of 2.21 meters average error in the east direction. This positional error appears to be due to incorrect rotational estimation about the horizontal axis of the dominant building facade. We note here that the GPS receiver only provided a 1 Hz signal, and was not synchronized with the camera, so that there was an unknown temporal offset between the clocks of the camera and GPS unit. The best temporal offset for each sequence was chosen by searching within a ten-second window and choosing the offset with the best median error, using linear interpolation of the GPS signal. Also, the accuracy of the localization is limited by the positional accuracy of the global point cloud and the city model to which it was aligned.

More test videos were captured in all areas using the tablet. Some sample localized images are shown in Figure 10, and estimated paths in all areas are plotted in Figure 12. The bottom row of Figure 10 shows a sequence which, for some periods, viewed an object which was missing from the global reconstruction; namely, a seasonal poster which was not there at the time of panorama capture. This gives a practical example of how the system can bridge tracking across areas missing from the global model by tracking from a local map.

8 Conclusions and Future Work

Our system provides real-time, accurately-registered camera pose tracking in indoor and outdoor environments. In comparison to previous model-based approaches, the use of a SLAM system allows for



(a) Area *A* (*Hauptplatz*)

(b) Area *B* (*Freiheitsplatz*)

(c) Area *C* (*Tummelplatz*)

Fig. 12: Examples of estimated camera paths outdoors. The yellow triangle indicates the viewing direction of the last keyframe in each sequence. For area *A*, the ground truth paths as measured by differential GPS are shown in blue.

continuous 6DoF tracking during the localization latency period and robustness to occlusion of the global model. Because the SLAM system is entirely self-contained on the client device, the cost of global localization is only incurred when the local map is expanded, and tracking within the local map does not require global feature lookup.

One improvement to our localization system would be to further divide the tasks on the server into separate requests, so that feature extraction and matching from multiple keyframe images could be processed in parallel. This would also allow the client more control in choosing when to request localization updates.

Outdoors, we rely on GPS and compass to choose the relevant database partition. Because these sensors can be unreliable or unavailable in some urban areas, the system could also be improved by introducing some form of image-based retrieval to help choose the correct database partition in these cases.

One interesting modification of our system would be to insert map points from the server-side global reconstruction into the client-side map. This would help during initialization and expansion phases, possibly easing the requirement for the user to translate to triangulate new points in the map. We might also consider updating and extending the server-side map with keyframes from the SLAM system.

Finally, we note that there are many yet-unexplored issues in interface design for global-registered augmented reality. With fully georegistered tracking and rich geometric information available, such as the city model used here, many new and interesting augmented reality interfaces are possible. Furthermore, the ideal interface would adapt its display to varying levels of localization accuracy and augmentation accuracy, with seamless transition between states of initialization, locally-referenced tracking, and globally-referenced tracking.

Acknowledgments

Thanks to Christian Pirchheim for helping to set up the optical tracker. This work was funded by Christian Doppler Gesellschaft (CD Labor Handheld Augmented Reality).

References

[1] C. Arth, M. Klopschitz, G. Reitmayr, and D. Schmalstieg. Real-Time Self-Localization from Panoramic Images on Mobile Devices. In *ISMAR*, 2011.

[2] C. Arth, A. Mulloni, and D. Schmalstieg. Exploiting Sensors on Mobile Phones to Improve Wide-Area Localization. In *ICPR*, 2012.

[3] C. Arth, J. Ventura, and D. Schmalstieg. Geospatial management and utilization of large-scale urban visual reconstructions. In *COM.Geo*, 2013.

[4] C. Arth, D. Wagner, M. Klopschitz, A. Irschara, and D. Schmalstieg. Wide Area Localization on Mobile Phones. In *ISMAR*, 2009.

[5] S. Cao and N. Snavely. Graph-Based Discriminative Learning for Location Recognition. In *CVPR*, 2013.

[6] R. O. Castle and D. W. Murray. Keyframe-based Recognition and Localization during Video-rate Parallel Tracking and Mapping. *IVC*, 2011.

[7] D. Chen, G. Baatz, Köser, S. Tsai, R. Vedantham, T. Pylvanainen, K. Roimela, X. Chen, J. Bach, M. Pollefeys, B. Girod, and R. Grzeszczuk. City-scale Landmark Identification on Mobile Devices. In *CVPR*, 2011.

[8] M. Cummins and P. Newman. FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance. *IJRR*, 27(6):647–665, June 2008.

[9] M. Cummins and P. Newman. Appearance-only slam at large scale with fab-map 2.0. *IJRR*, 30(9):1100–1123, Aug. 2011.

[10] A. J. Davison. Real-time Simultaneous Localisation and Mapping with a Single Camera. In *ICCV*, 2003.

[11] A. J. Davison, W. W. Mayol, and D. W. Murray. Real-Time Localisation and Mapping with Wearable Active Vision. In *ISMAR*, 2003.

[12] E. Eade and T. Drummond. Unified loop closing and recovery for real time monocular SLAM. *BMVC*, 2008.

[13] S. Feiner, B. MacIntyre, T. Höllerer, and A. Webster. A touring machine: Prototyping 3D mobile augmented reality systems for exploring the urban environment. *Personal and Ubiquitous Computing*, 1997.

[14] M. A. Fischler and R. C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Commun. ACM*, 1981.

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

[16] B. K. P. Horn. Closed-form Solution of Absolute Orientation using Unit Quaternions. *J. Opt. Soc. Am. A*, 1987.

[17] A. Irschara, C. Zach, J. Frahm, and H. Bischof. From Structure-from-Motion Point Clouds to Fast Location Recognition. In *CVPR*, 2009.

[18] G. Klein and D. Murray. Parallel Tracking and Mapping for Small AR Workspaces. In *ISMAR*, 2007.

[19] G. Klein and D. Murray. Parallel Tracking and Mapping on a Camera Phone. In *ISMAR*, 2009.

[20] A. Le Bris and N. Paparoditis. Matching terrestiral images captured by a nomad system to images of a reference database for pose estimation purpose. *IAPRS*, XXXVIII(Part 3A), 2010.

[21] Y. Li, N. Snavely, D. Huttenlocher, and P. Fua. Worldwide pose estimation using 3d point clouds. In *ECCV*, 2012.

[22] Y. Li, N. Snavely, and D. P. Huttenlocher. Location Recognition using Prioritized Feature Matching. In *ECCV*, 2010.

[23] H. Lim, S. Sinha, M. Cohen, and M. Uyttendaele. Real-Time Image-Based 6-DOF Localization in Large-scale Environments. In *CVPR*, 2012.

[24] P. Lothe, S. Bourgeois, F. Dekeyser, E. Royer, and M. Dhome. Towards geographical referencing of monocular slam reconstruction using 3d city models: Application to real-time accurate vision-based localization. In *CVPR*, 2009.

[25] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *IJCV*, 2004.

[26] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP*, 2009.

[27] A. Mulloni, M. Ramachandran, G. Reitmayr, D. Wagner, R. Grasset, and S. Diaz. User-friendly SLAM initialization. In *ISMAR*, 2013.

[28] T. Oskiper, H.-P. Chiu, Z. Zhu, S. Samaresekera, and R. Kumar. Stable vision-aided navigation for large-area augmented reality. In *VR*, 2011.

[29] T. Oskiper, S. Samarasekera, and R. Kumar. Multi-sensor navigation algorithm using monocular camera, IMU and GPS for large scale augmented reality. In *ISMAR*, 2012.

[30] R. Raguram and J.-M. Frahm. RECON: Scale-adaptive Robust Estimation via Residual Consensus. In *ICCV*, 2011.

[31] G. Reitmayr and T. W. Drummond. Going Out: Robust Model-Based Tracking for Outdoor AR. In *ISMAR*, 2006.

[32] G. Schindler, M. Brown, and R. Szeliski. City-Scale Location Recognition. In *CVPR*, 2007.

[33] N. Snavely, S. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3D. *ACM Transactions on Graphics (TOG)*, 2006.

[34] G. Takacs, V. Chandrasekhar, N. Gelfand, Y. Xiong, W. C. Chen, T. Bismpigiannis, R. Grzeszczuk, K. Pulli, and B. Girod. Outdoors augmented reality on mobile phone using loxel-based visual feature organization. *MIR*, 2008.

[35] G. Takacs, M. El Choubassi, Y. Wu, and I. Kozintsev. 3D mobile augmented reality in urban scenes. In *ICME*, 2011.

[36] A. Torii, J. Sivic, and T. Pajdla. Visual localization by linear combination of image descriptors. In *ICCV Workshops*, 2011.

[37] G. Vaca-Castano, A. R. Zamir, and M. Shah. City scale geo-spatial trajectory estimation of a moving camera. In *CVPR*, 2012.

[38] J. Ventura and T. Höllerer. Wide-area Scene Mapping for Mobile Visual Tracking. In *ISMAR*, 2012.

[39] B. Williams, G. Klein, and I. Reid. Automatic Relocalization and Loop Closing for Real-Time Monocular SLAM. *PAMI*, 2011.

[40] A. R. Zamir and M. Shah. Accurate Image Localization Based on Google Maps Street View. In *ECCV*, 2010.

[41] J. Zhang, A. Hallquist, and A. Zakhor. Location-Based Image Retrieval for Urban Environments. In *ICIP 2011*, 2011.

[42] W. Zhang and J. Kosecka. Image based localization in urban environments. *3DPVT*, 2006.