Computer Vision Winter Workshop 2010, Libor Špaček, Vojtěch Franc (eds.)
Nové Hrady, Czech Republic, February 3–5
Czech Pattern Recognition Society

3D Camera Tracking in Unknown Environments by On-line Keypoint Learning

Paul Wohlhart, Peter M. Roth, and Horst Bischof

Institute for Computer Graphics and Vision, Graz University of Technology
{wohlhart,pmroth,bischof}@icg.tugraz.at

Abstract

In this paper we present an efficient algorithm for camera tracking applicable for mobile devices. In particular, the work is motivated by the limited computational power and memory, precluding the use of existing methods for estimation of the 6-DoF pose of a mobile device (camera) relative to a previously unknown planar object. Similar to existing methods, we introduce a keypoint based approach. We establish a relationship between the object and its image by selecting keypoints on the object, preferably such with a distinctive appearance, and identifying their location within subsequent images. In contrast to existing works, we solve the problem of re-identifying such feature points by robustly learning their appearance with an on-line learning algorithm. We demonstrate the proposed algorithm, hence not limited to this application, in the context of AR. In particular, we give several qualitative and quantitative evaluations showing the benefits of the proposed approach.

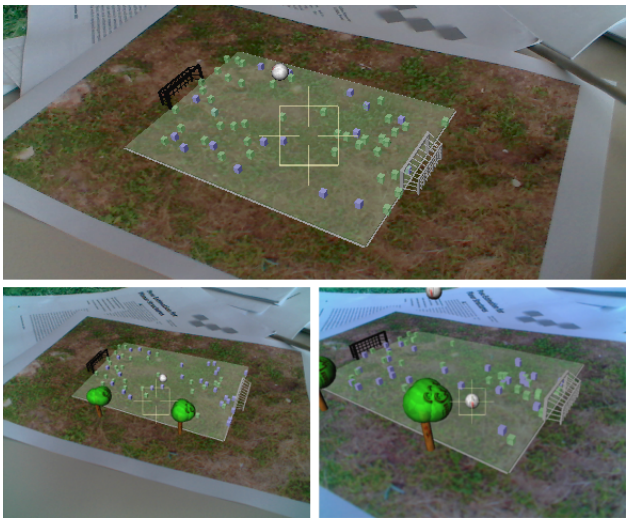


Figure 1: A planar textured target augmented with a virtual soccer field. The user can explore the virtual world by viewing the target from different angles and the tracker keeps the virtual image well aligned to the ground plane.

1 Introduction

For many applications in computer vision and graphics the exact position and orientation (together called the *pose*) of a moving camera, relative to the depicted objects, has to be estimated. This process of continuously recovering the camera's pose is known as visual camera tracking. The majority of successful methods are based on local feature points, *i.e.*, they try to identify points that can be robustly re-identified in different views. In this way, typical problems such as partial occlusions or specular reflections, that make detection of parts of the object impossible, can be handled by redundancy.

The first steps are identifying interesting feature points and describing them such that a re-identification under slightly different conditions (viewing angle, lighting, etc.) is possible. Next, when a set of correlations of 2D points in the current image and 3D points in the world is established and the geometrical constellation of the points (*i.e.*, their coordinates) on the object is known, the correct pose (matching those two sets of points) can be estimated. This is called the *Perspective-n-Point* problem.

Essentially, there are two backgrounds from which most of the camera tracking approaches emerged, namely Structure from Motion (SfM) and Simultaneous Localization and Mapping (SLAM). SfM is mainly targeted at reconstructing a perfect model in an off-line manner, where all images are available and processing time is only of limited importance. Thus, bundle adjustment can be used to achieve very accurate results [18, 2, 13, 4, 24]. Such systems are used, for instance, in the post production of movies. This line of work is clearly not suited for applications in real-time, low resources domains. SLAM, on the other hand, was originally developed for robot self localization. Thus the goal is to build and constantly refine a rough model of the environment, just detailed enough to fulfill tasks such as navigation planning with obstacle avoidance, and at the same time always have a good estimate of the current pose. As pointed out in [1], there are still differences in the prerequisites for traditional SLAM algorithms used in robot navigation and those of camera tracking for hand-held mobile devices. The main difference is that the movement of a hand-held camera can be very fast and unpredictable and thus no reliable motion models can be constructed. Additionally, motion blur and occlusions might degrade the image, thus, one has to deal with situations where tracking fails and has to be recovered without any external support.

This problem gets more difficult, if we want to track a camera in a previously unknown environment. Hence, the system has to identify interesting local feature points and to learn their appearance for robust re-identification. Additionally, the initially unknown virtual coordinate system used to render virtual objects, should be oriented and scaled relative to the target. Thus, we have to specify the initial virtual coordinate system at startup. In our case this is realized by user interaction. Furthermore, since no predefined geometrical model of the target object is available, the use of model based approaches (*e.g.*, [7, 11, 20, 17]) is prohibited. Instead the location of chosen keypoints on the object with respect to the virtual coordinate system (*i.e.*, 3D world coordinates) must be calculated. For the first frame, with the initial pose given by the user, this is straight forward. But during operation, when certain points fail to be tracked or become invisible, new keypoints have to be initialized for tracking and their 3D coordinates have to be calculated from estimates of the current pose (subjected to noise, in a robust manner).

This process of continuously estimating the pose, and at the same time constructing a model of the environment, is known as simultaneous localization and mapping (SLAM) or as parallel tracking and mapping (PTAM) [3, 9, 27, 1]. The method combines the advantages of accurate model building with a probabilistic framework, constructing a full covariance matrix of all model points and optimizing a robust cost function with bundle adjustment, while still tracking in real-time. This is achieved by splitting the two processes apart. Tracking is performed for every frame on a small subset of the 3D map constructed up to this point. Map building is deferred to a background thread, only triggered for keyframes, which are selected with a certain distance in time and only if the current pose is robust and shows a significantly different view of the scene.

One application that requires highly accurate camera pose estimation is mobile augmented reality (AR). In general, augmented reality systems allow the user to experience the real environment augmented by artificially computer generated objects. The user can see the virtual objects as if they were present in the real world. To give the user the consistent impression of a virtual object really living in the natural space, it is absolutely crucial that the images of the virtual objects are well aligned with the environment (*i.e.*, well registered), depending on the user's current point of view. Thus, a strong requirement on the algorithms in this field is real-time capability under limited computational resources. Although mobile hand-held devices evolve rapidly and components for fast floating-point operations and even dedicated graphics hardware with good performance are integrated into these systems, the power of a fully equipped desktop computer is not at hand.

Hence, the goal of this work was to develop an efficient camera tracking method that can be applied within the context of mobile AR. Figure 1 illustrates such a system in action. An arbitrary planar target defines the ground plane and is augmented by a virtual interactive soccer field. The small cubes show the locations of tracked object features. The color indicates if the feature was found in the current image and its position was geometrically consistent with the others (green/light) or not (blue/dark); only the green cubes are used for pose estimation.

Even though recent advances in mobile device hardware make it possible to use systems such as PTAM for mobile AR, we want our system to run on platforms lacking the power to run background jobs and to perform affine warps of keypoints for template matching. To overcome these limitations, more efficient trackers can be applied. For instance, in [25] the authors show that by using a combination of a modified version of SIFT and FERNS [16] very fast and stable tracking results can be achieved on standard low cost hand-held devices like mobile phones. However, the target objects have to be known in advance for off-line keypoint classifier learning. Similarly, recently an algorithm for tracking planar textured objects by its natural features has been proposed [6]. It defines tracking as a classification problem, learning classifiers for each point to track on-line. Compared to methods utilizing image patches or other keypoint descriptors such as SIFT [12, 10] this method is especially interesting because of its low performance and memory requirements.

Since the tracker presented in [6] is computationally very efficient even on low performance hand-held devices, it is highly interesting for our target application, *i.e.*, mobile AR. However, to meet the requirements of our application, several extensions are required. First, a mapping of tracked keypoints to a virtual coordinate system to deliver a global camera pose estimate relative to the target object has to be established. Additionally, amendments were made to the process of searching for object features and the policy of exchange of features. Finally, the whole system was integrated into the OpenTracker framework, to serve as pose estimation module in the Studierstube Augmented Reality framework[1].

The rest of the paper is organized as follows. Section 2 gives an overview on the algorithms building the basis for our approach. Section 3 gives a short overview of the complete system's design and a detailed explanation of the tracking process. In Section 4 qualitative and quantitative experiments analyze the system's capabilities, and finally, Section 5 gives a short discussion and conclusion.

2 Camera Tracking and Pose Estimation

In the following, we give an overview on the algorithms building the basis for our work, *i.e.*, on-line learning for feature tracking and stable and efficient pose estimation.

2.1 Tracking

The work at hand is built on the tracker presented in [6], which is able to robustly track a large variety of textured planar objects. In our case, the re-identification of feature points can be formulated as a classification problem between successive frames. As a first step, interest points are extracted from the current frame (*e.g.*, Harris corners [8]) and a subset of fixed size is selected (preferably uniformly distributed over the target object). For each of them a classifier is trained to distinguish a small patch around the keypoint from the rest in the image (*i.e.*, other keypoints on the object,

[1] `http://www.studierstube.org/`

as well as in the background). For training the classifiers Online Boosting for Feature Selection [5] is used. Tracking is then performed by classifying all interest points in the new frame and selecting the one with highest confidence as a candidate match for the associated keypoint in the last frame. These candidates are verified for geometric consistency by robustly estimating a homography between the constellations in the two successive frames (using RANSAC). This allows for robust re-detection of points with slightly changed appearance in the next frame and subsequently incorporation of the new appearance by on-line learning, thus adapting to context specific transformations of the patch. In contrast to [14] the matching of keypoints is more robust and faster and can therefore be applied on the whole image instead of just a small surrounding. Thus, allowing for very fast camera movement and fast recovery from tracking failure due to strong motion blur or full occlusion.

Additionally, a feature exchange mechanism increases the adaptivity. If one of the initially chosen features cannot be re-detected reliably over time, the system can discard such a feature and initialize a new classifier for a different interest point. For that purpose, a simple confidence measure is calculated, estimating the probability to re-detect each feature in the next frame from its detection history:

$$P_{i,t+1} = \beta \cdot P_{i,t} + (1 - \beta) \cdot \delta_i, \tag{1}$$

where $P_{i,t+1}$ is the probability to re-detect feature point i in the next frame; δ_i is 1 if it was detected in the current frame; 0 otherwise; $\beta \in \mathbb{R}, 0 \leq \beta \leq 1$, determines the influence of the feature point's history.

2.2 Pose Estimation

The natural feature tracker described above successfully tracks the location and orientation of planar objects throughout a video, but it does not establish a virtual coordinate system and thus also does not calculate a 6-DoF camera pose relative to it. The frame to frame homographies implicitly encode the camera motion between frames and concatenating the homographies would thus give a pose relative to the first frame. But since every homography is estimated, such a scheme would sum up the individual errors and make it susceptible to drift. Furthermore simple plane to plane homography calculation does not take into account internal camera parameters available from calibration, but includes them in the estimation procedure. Altogether recovering the pose directly from a summed up homography would not deliver the accuracy and stability of registration needed in AR.

When internal camera parameters are available, image points can be transformed into the coordinate system of an ideal pinhole camera, and finding the pose that matches the correspondences of 3D points in the world and their 2D projections on the current camera's image plane is known as the *Perspective-n-point* (PnP) problem. Formally, the problem is stated as follows: Points on the object with the three dimensional coordinates $M_i = (X_i, Y_i, Z_i)$ are being projected perspectively to the points $m_i = (u_i, v_i)$ on the image plane ($i = 1, \ldots, n$) by the 3×4 projection matrix $\boldsymbol{P}$ defined up to scale s:

$$s\tilde{\boldsymbol{m}}_i = \boldsymbol{P}\tilde{\boldsymbol{M}}_i. \tag{2}$$

In the case of a perspective camera $\boldsymbol{P}$ can be further decomposed into

$$\boldsymbol{P} = \boldsymbol{K}\,[\boldsymbol{R}|\boldsymbol{t}]\,, \tag{3}$$

where $[\boldsymbol{R}|\boldsymbol{t}]$ is the composition of a 3×3 rotation matrix $\boldsymbol{R}$ and a 3×1 translation vector $\boldsymbol{t}$, describing the rotation and translation of the world coordinate system to the camera coordinate system, *i.e.*, the camera's *pose*. They are also called the camera's *external* parameters.

$\boldsymbol{K}$ is the camera calibration matrix, holding the camera's *internal* parameters, describing how points expressed in the camera coordinate system are projected onto the image plane:

$$\boldsymbol{K} = \begin{pmatrix} f_u & s & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{4}$$

where f_u and f_v are the camera's focal length multiplied by the pixel resolution along the u- and v-axis of the image; u_0 and v_0 define the image origin's offset to the camera's principal point (the point where the optical axis intersects the image plane); s is the skew factor usually being 0 unless the camera's u- and v-axis are not perpendicular. In our case, these parameters were estimated using publicly available software[2].

Consequently the task is to find $\boldsymbol{P}$ (*i.e.*, given $\boldsymbol{K}$, find $[\boldsymbol{R}|\boldsymbol{t}]$) that satisfies Equation (2) for a given set of point correspondences $\boldsymbol{m}_i \leftrightarrow \boldsymbol{M}_i$. Since in real world measurements we always have to deal with noise, in general, there will be no solution fitting all points perfectly, and one has to search for a $\boldsymbol{P}$ that fits best, *i.e.*, minimizes an error function. Essentially, either the *image space error* E_{is} or the *object space error* E_{os} can be taken. E_{is} describes the total residue of projections of the $\boldsymbol{M}_i$ onto the image plane by $\boldsymbol{P}$, to the actual measured $\boldsymbol{m}_i$. E_{os} sums up the perpendicular distance of $\boldsymbol{M}_i$ to rays coming from the camera defined by $\boldsymbol{P}$ and going through $\boldsymbol{m}_i$. Minimizing E_{is} would be preferable for AR because it directly defines how well the final rendering is aligned to the image, unfortunately, it is very hard to efficiently formulate the PnP problem in terms of E_{is}, resulting in that most methods optimize E_{os}.

Solving the PnP problem has been investigated thoroughly and various methods exist with different computational complexity, speed and stability of convergence, robustness, and accuracy. For our purpose, we are especially interested in the case where the world coordinates of the object are coplanar. Since this is a special case, leading to singularities in most mathematical formulations. For many algorithms there exist special extensions for this case (*e.g.*, [15]). Particularly challenging effects in the planar case with noisy measurements are pose ambiguities arising in certain configurations (distant camera or steep viewing angle). In [22], an algorithm is presented that pays special attention to those cases and basically iteratively refines both possible solutions until one is more probable and thus avoids

[2]Camera Calibration Toolbox for MATLAB: `http://www.vision.caltech.edu/bouguetj/calib_doc/`

randomly choosing one solution for each frame, making the final outcome jump around. Since this method delivers very accurate results and an efficient implementation is available, we use it to solve the pose estimation task in this work.

3 NaturalFeatureTracker for AR

In the following, we give a detailed overview of our natural feature tracker built on the algorithms discussed in Sections 2.1 and 2.2 and show how it can be included into an AR system.

3.1 NaturalFeatureTracker

The following is an outline of the process of tracking and pose estimation within the NaturalFeatureTracker (NFT). In the beginning, as mentioned above, the system needs to be initialized with a definition of the target object to track and the ground plane of the virtual coordinate system relative to it in the first frame. When the user decides to start, the current frame is frozen, and the user clicks the four corner points of the rectangular planar target region. The rest of the system does not rely on a rectangular target area, but it gives a convenient way to enter the initial pose and scale. The aspect ratio of the target area is not predefined, but can be calculated along with the initial pose [23]. Every new frame arriving is first preprocessed by a conversion to gray-scale and smoothing with a Gauss kernel for noise reduction. Subsequently Harris-corners [8] are extracted as interest points (however, any other interest point detector for which fast implementations exist could be used).

Feature Point Matching On the first frame, classifiers are trained to re-detect feature points in follow-up frames. Using the target region the keypoints found in the image are split into object and background keypoints. For a fixed number of keypoints on the object, chosen such as to cover most of the area, classifiers are initialized by learning the patch around the respective location as positive; patches around all other object keypoints or those in the background are learned as negative. Starting from the second frame, classifiers for a set of feature points have already been initialized and can then be used to re-identify them in the current image. This is done by evaluating every classifier on all candidate keypoints in the new frame and choosing the one with the highest confidence.

Since this mapping might include false matches, a (RANSAC style) verification step is introduced. A series of random subsets of 4 point correspondences is created and for each of them a homography is calculated, mapping the locations of the points in the previous frame to the current ones. By applying each homography to all points in the last image and comparing the distance of the mapped points to the locations of the re-identified matches, inliers supporting the homography can be found. The homography with most inliers and smallest total error (distances) is chosen as the correct one. Only inliers of the best homography are considered as true matches and used further on.

Feature Classifier Update The classifiers of all positively re-identified feature points are updated with a positive patch around the new location and a random negative one. Thus, scene specific appearance changes of feature points are incrementally learned on-line. The others are not updated, since they might not have been found, because they currently are occluded.

Pose Estimation Taking all features for which the world coordinates and the position on the current frame are known, a pose that projects the keypoints from world coordinate system to the respective current keypoint coordinates on the image can be found by solving the PnP problem with the method described in Section 2.2. For the first frame world and image coordinates are only known for the corners of the initial object region defined by the user. For all consecutive frames matched and verified feature points for which world coordinates have been calculated before are included.

Map Building Using the current pose, world coordinates for new object features are calculated by projecting the image coordinates back onto the object plane. This calculation is repeated on a series of frames for which poses have been recovered to have a set of world coordinate estimates over time. These can be averaged (again with outlier removal) to reduce the influence of noisy pose estimates. Only if a new object feature is considered stable enough, it is included in the pose estimation.

Feature Exchange As a final step the quality of the tracked object features is evaluated. Features that have not been found in a certain number of frames are dropped and replaced by new ones initialized on a new interest point on the object. Since for accurate pose calculation it is beneficial to have matched feature points not clustered on one spot in the image but spread over the entire object, the target object region is split into a 3x3 grid and new keypoints are selected preferably in cells with few tracked feature points.

One improvement to speed up the process of feature point recognition is to reduce the search area. In this way, not every classifier has to be matched to every interest point. A common way is to use an EKF (Extended-Kalman-Filter) framework with a motion model to predict the locations of the object features in the next image. This works especially well for robot navigation tasks, where accurate motion models can be constructed and often some kind of control and odometry data is available. Since camera motion of hand-held cameras cannot be estimated very well, it is a safe bet to take the simplest model, which is to just assume small motions with equal probability for all directions. Thus, we define a small radius around the locations of feature points in the previous frame as the search area. Only if no match can be established or too few matches remain as inliers of the best homography, the search radius is increased until, if necessary, the whole image is searched again (tracking loss recovery).

3.2 AR System

The overall AR system including the previously introduced tracker, based on the Studierstube framework, is shown in Figure 2. The video capturing module OpenVideo acquires the camera's current picture, which is sent to the Studierstube core and forwarded into its event system, which is responsible for all interactions, and holds an instance of OpenTracker for tracking all moving artefacts relevant to the AR application. On the arrival of the image, OpenTracker notifies all

subcomponents that registered for video events. The natural feature tracker developed for this work was implemented as a plug-in module for OpenTracker. It receives the image, calculates the pose and returns it to OpenTracker, from where it is propagated to the Studierstube core, which makes it possible to use it in the scene graph as the pose of the virtual camera, used to render the augmented scene on top of the camera's image.

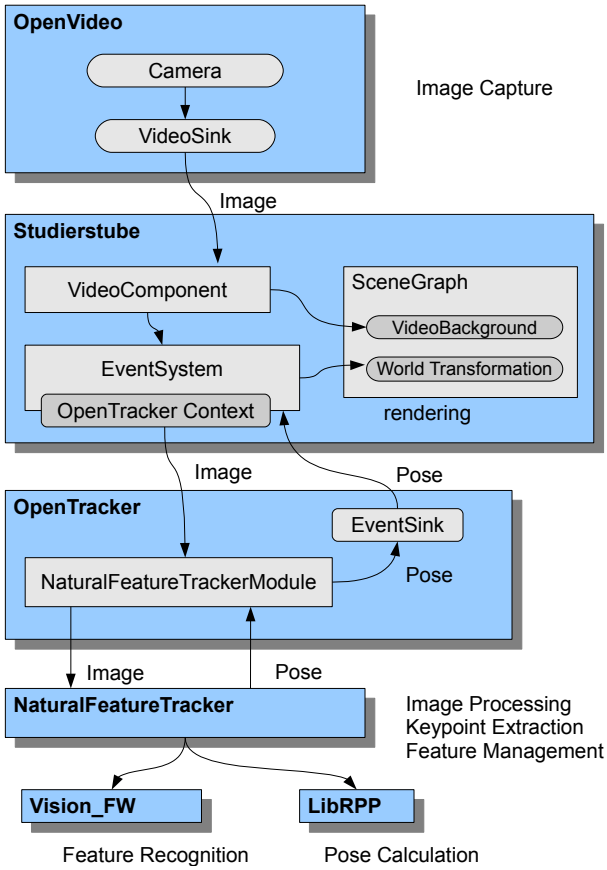


Figure 2: Dataflow within the Studierstube system with NaturalFeatureTracker for camera pose estimation: from image acquisition to pose calculation and final output of the augmented image

4 Experimental Results

Since the target application of this work is mobile augmented reality, we first present some qualitative results, showing the natural feature tracker working in two AR scenarios. Figure 3 shows the system, augmenting an ortho-photo of the Jakominiplatz at the center of Graz with a 3D model of the place's subsurface infrastructure (gas and water pipelines). The data was taken from the Vidente[3] project, targeted at helping construction workers in planning projects [21]. Figure 4 shows an interactive soccer field, rendered on top of any kind of planar surface. Users can investigate the scene, plant trees and make balls jump by using the tracked camera as an input device.

The first quantitative evaluation investigates the accuracy of the resulting pose. We compare the pose coming from

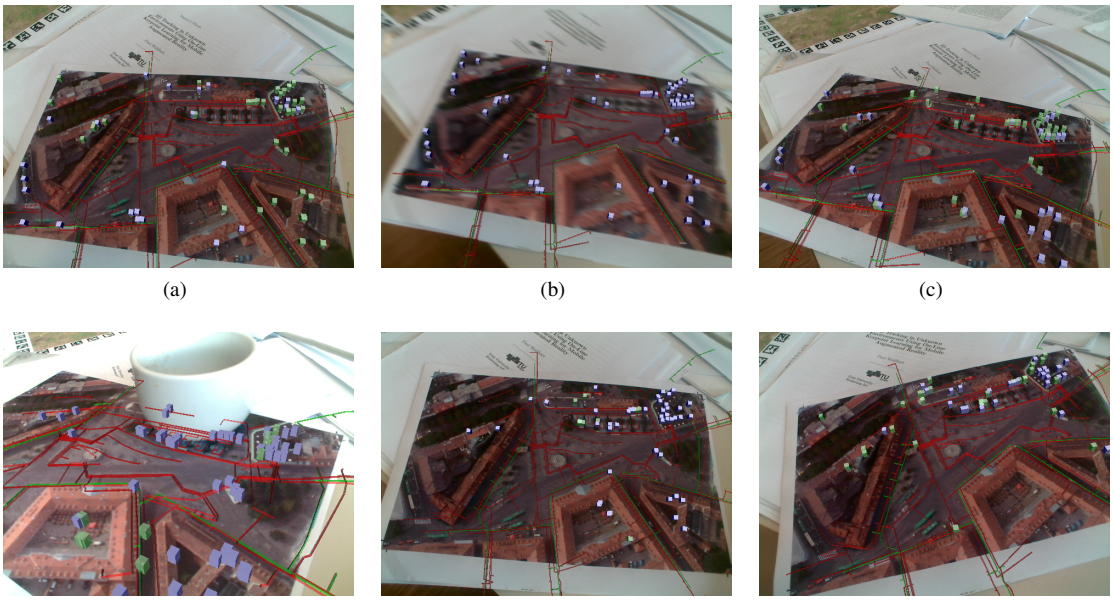
[3] `http://www.studierstube.org/vidente`

Figure 5: Target with fiducial markers to be tracked by ARToolkitPlus and content tracked by NaturalFeatureTracker.

our tracker with the one returned by the marker-based tracker ARToolkitPlus [26] on a specifically designed target shown in Figure 5. The ARToolkitPlus module calculates the pose from the markers, whereas the NFT is initialized to track the image. Figure 6(a) shows the path of the camera as recorded by our tracker, while Figures 6(b) and 6(c) show the differences between the positions and orientations for each frame. The results are very similar, demonstrating the capability of the classifiers to locate the feature points and the robustness and accuracy of the pose estimation. The differences in translation remain below 0.5%. The angle between the cameras' orientations has a mean value of $0.45°$ with a standard deviation of $0.27°$.

In addition, to have a steerable environment for the pose and to have ground-truth, synthetic scenes were constructed. One particular challenge for pose estimation from natural features on planar targets are situations where the viewing angle gets steep. The features are perspectively maximally distorted and in the case of a $90°$ angle reduced to a line. We want to investigate how our on-line learners can adapt to continuous perspective distortion of the features. Figure 7(a) shows the results of the first synthetic scene, in which the target object initially faces the camera and is then rotated to the side. Multiple clips were recorded with different rotation speeds, defining the x-axis of the figures. The blue line (dots) shows the mean value and the standard deviation of the last angle at which the target was successfully tracked in a series of trial runs. However, the tracker might have failed before and recovered, indicated by the red line (crosses) showing the range of angles at which the tracker failed the first time. Figures 7(b) and 7(c) show the according errors in position and orientation for successfully tracked frames.

With the second experiment, we tested the system's capability to track objects continuously rotating in front of the camera, around its viewing axis. If there is too much rotation, feature points get lost and feature exchange will have to kick in to continue operation. Results are shown in Figure 8. As the results show, tracking works fine up to an angle of about $60°$ for both kinds of rotations and rotation speeds up to $2°$ per frame, which makes $30°$ per second at a frame rate of 15 fps. One conclusion that can be drawn from both



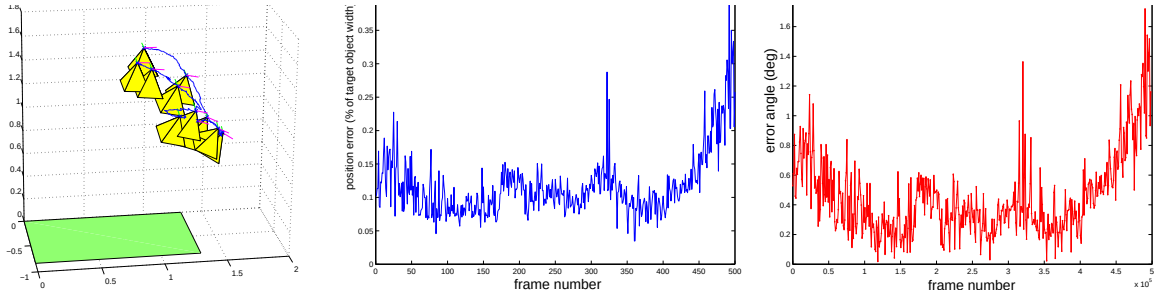
(a) (b) (c)

(d) (e) (f)

Figure 3: Tracking sequence of an ortho-photo of Jakominiplatz augmented with sub-surface infrastructure (gas and electricity pipelines): (a) initial frame showing the aligned sub-surface infrastructure models and the feature points as cubes, (b) fast camera movement causes the tracker to fail on one frame, (c) tracking is recovered in one of the subsequent frames, (d) continuous tracking even when several feature points cannot be detected in the current frame (zoomed in and occluded), (e) when zooming out again after a longer period of close-up frames, the feature exchange mechanism placed all the features in the formerly visible area (feature points clustered in top right), (f) some frames later the feature points spread out again to cover the entire object.

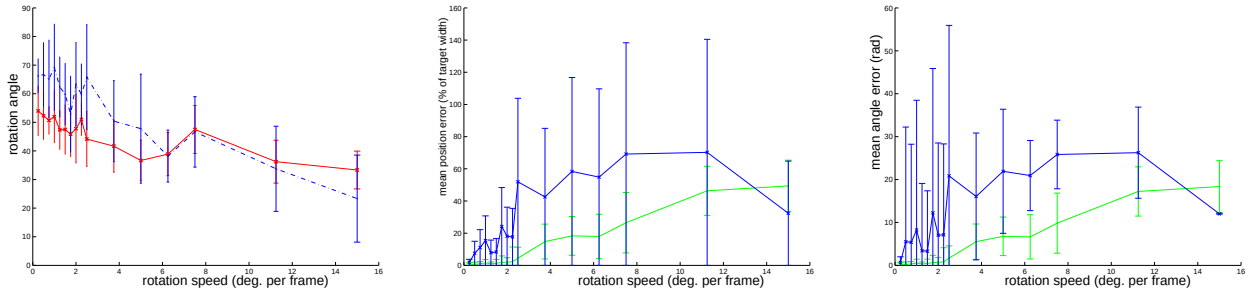
(a) virtual soccer field (b) a different target object (c) zooming in and placing a new ball

Figure 4: Rendering an interactive animated virtual soccer field on top of different grass-textured objects: (a) and (b) different grass textures, robust tracking of the whole region or a sub-area; (c) additional virtual objects (bouncing balls and trees) can be added interactively by pointing the cross-hairs at the desired location on the ground and pressing a key.



(a) Camera poses from NFT (b) Distance between the calculated positions (c) Angle between the cameras' orientations

Figure 6: Comparison of the camera poses calculated by NaturalFeatureTracker and ARToolkitPlus on a short video sequence.



(a) Average rotation angle for which tracking failed the first time (red/crosses) and up to which tracking worked (blue/dots).

(b) Mean value and standard deviation of camera position error for continuously tracked frames (green/dots) and for frames where tracking was recovered after failure (blue/crosses).

(c) Mean value and standard deviation of camera orientations error for continuously tracked frames (green/dots) and for frames where tracking was recovered after failure (blue/crosses).

Figure 7: Results of tracking an ortho-photo of Jakominiplatz rotating about the world coordinate frame's y-axis for different rotation velocities

experiments is that the Haar-features, forming the base of the classifiers' decisions, perform badly on increasingly rotated patches. When pure rotation exceeds about 30° lots of feature points are lost. For slow rotations, adaptation to the rotated appearance of the patches is possible, but the discriminative power is lost and hence, the effect of learning has a smaller impact on the angle up to which the target can be tracked, than the initialization of new features gradually replacing old ones that cannot be found anymore.

The second observation is that the accuracy of the resulting pose drops as soon as the tracker lost the object in one frame, even if the tracker finds enough correct matches again. This is illustrated in Figures 7(b), 7(c), 8(b) and 8(c). The green lines (dots) show the mean and standard deviation of errors in position and orientation for all frames, from the beginning to the point when tracking starts to fail. For small rotation speeds the translation error is usually around 1% of the target object's width and the error of the rotation angle is about 1°. The blue lines (crosses) however show the errors in position and orientation when tracking was recovered after failing on one frame. The poses returned in those cases are not really suitable for AR purposes any more. The same applies for fast camera rotation (about 10° per frame) already on consistently tracked frames.

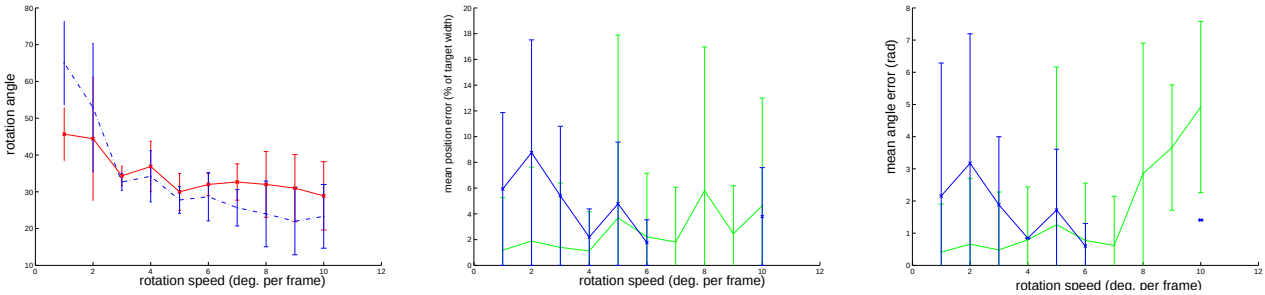
Finally, we evaluate the performance in terms of computation time. As the primary target platform an ultra-mobile PC (UMPC): Sony Vaio VGN-UX50 (1.06 GHz) was chosen. The built-in rear camera's resolution was set to 320x240. Additional experiments were conducted on the development platform: Sony Vaio VGN-FE11M, Intel Core2 Duo (1.66 GHz). Apart from the slightly faster CPU clock speed, the laptop has the advantage of the dual core architecture, allowing the multi-threaded Studierstube framework to basically dedicate one core to the tracker only. Here the camera is a standard WebCam and with a resolution of 640x480. Table 1 shows the profiling data of two experiments on both platforms. The most essential observation is, that without restriction of the search area, the matching of all classifiers to all interest points is the dominating performance bottleneck. With search area restriction in place, it becomes a minor factor and the remaining costly tasks are the extraction of interest points and the update of the classifiers. For the former, faster methods could be considered (*e.g.* [19]), the latter is constitutive to this approach, but could be sped up a little, by working on some implementation issues. Summing it up, one can see that interactive frame rates can be achieved even on the UMPC, if some form of search area restriction is applied.

Platform	UMPC		Laptop	
Image res.	**320x240**		**640x480**	
# Interest Points	200		300	
# Tracked Features	25		30	
Search area restr.	**no**	**yes**	**no**	**yes**
Task	t(ms)	t(ms)	t(ms)	t(ms)
Img. cap. & prep.	7	6	15	15
Interest Points	15	13	31	30
Feat. Matching	108	8	46	23
Homography	5	4	2	2
Feat. Update	22	20	10	11
Feat. Exchange	0.5	0.5	0.5	0.1
Pose est.	3	3	1.2	1.1
Total	166	58	111	65

Table 1: Timing of the algorithm's steps per frame.

5 Discussion and Conclusion

For many applications in computer vision, the exact estimation of the 6-DoF pose of the camera, has to be estimated, which is referred to as camera tracking. One prominent application is mobile AR (in unknown environments). However, due to limited computational power and memory on mobile devices state-of-the-art approaches such as SLAM or SfM can not be applied. Hence, in this paper we proposed an efficient camera tracking approach that is highly applicable for mobile devices. The key idea is to apply an on-line learning method to robustly identify keypoints by adaptively learning scene specific representations. Hence, a real-time capable natural feature tracker can be derived, which together with a robust pose estimation builds the basis for the camera tracker. Since our target application was



(a) Average rotation angle for which tracking failed the first time (red/crosses) and up to which tracking worked (blue/dots).

(b) Mean value and std. dev. of camera position error for (green/dots) continuously tracked frames and (blue/crosses) after recovering from failure.

(c) Mean value and std. dev. of camera orientation error for (green/dots) continuously tracked frames and (blue/crosses) after recovering from failure.

Figure 8: Results of tracking an ortho-photo of Jakominiplatz rotating in front of the camera (*i.e.*around the z-axis) for different rotation velocities (synthetic scene created with Blender).

mobile AR, the tracker was integrated into the Studierstube AR framework, where OpenTracker and our NaturalFeature-Tracker submodule are used to determine the position and orientation of the tracked camera. The experimental results show promising results of the working system under real-live conditions. Interactive frame rates can be reached and the registration of virtual content to the scene shown in the camera's image show good visual results. Future work will include finding better mechanisms for feature exchange and including new features which can handle rotations considerably better.

Acknowledgement

The work was supported by the FFG project MDL (818800) under the Austrian Security Research Programme KIRAS. In addition, the authors want to thank Helmut Grabner and Gerhard Schall for their support.

References

[1] R. O. Castle, D. J. Gawley, G. Klein, and D. W. Murray. Video-rate recognition and localization for wearable cameras. In *Proc. British Machine Vision Conf.*, pages 1100–1109, 2007.

[2] K. Cornelis, M. Pollefeys, and L. v. Gool. Tracking based structure and motion recovery for augmented video productions. In *Proc. Symp. on Virtual Reality Software and Technology*, pages 17–24, 2001.

[3] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 26(6):1052–1067, 2007.

[4] A. W. Fitzgibbon and A. Zisserman. Automatic camera recovery for closed or open image sequences. In *Proc. European Conf. on Comp. Vision*, pages 311–326. Springer-Verlag, 1998.

[5] H. Grabner and H. Bischof. On-line boosting and vision. In *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*, pages 260–267, 2006.

[6] M. Grabner, H. Grabner, and H. Bischof. Real-time tracking with on-line feature selection. In *Video Proc. in conjunction with IEEE Conf. on Comp. Vision and Pattern Recognition*, 2006.

[7] C. Harris. Tracking with rigid models. In A. Blake and A. Yuille, editors, *Active vision*, pages 59–73. MIT Press, 1992.

[8] C. Harris and M. Stephens. A combined corner and edge detector. In *Proc. Alvey Vision Conf.*, pages 147–151, 1988.

[9] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. Int. Symp. on Mixed and Augmented Reality*, 2007.

[10] T. Lee and T. Höllerer. Multithreaded hybrid feature tracking for markerless augmented reality. *IEEE Trans. on Visualization and Comp. Graphics*, 15(3):355–368, 2009.

[11] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 28(9):1465–1479, 2006.

[12] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. Journal of Computer Vision*, 60(2):91–110, 2004.

[13] D. Nistér. *Automatic dense reconstruction from uncalibrated video sequences*. PhD thesis, Royal Institute of Technology KTH, Stockholm, Sweden, 2001.

[14] D. Nistér, O. Naroditsky, and J. Bergen. Visual odometry. In *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*, volume 1, pages 652–659, 2004.

[15] D. Oberkampf, D. F. DeMenthon, and L. S. Davis. Iterative pose estimation using coplanar feature points. *Computer Vision and Image Understanding*, 63(3):495–511, 1996.

[16] M. Özuysal, P. Fua, and V. Lepetit. Fast keypoint recognition in ten lines of code. In *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*, 2007.

[17] Y. Park, V. Lepetit, and W. Woo. Multiple 3D object tracking for augmented reality. In *Proc. Int. Symp. on Mixed and Augmented Reality*, pages 117–120, 2008.

[18] M. Pollefeys, R. Koch, and L. V. Gool. Self-calibration and metric reconstruction inspite of varying and unknown intrinsic camera parameters. *Int. Journal of Computer Vision*, 32, 1999.

[19] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *Proc. Int. Conf. on Computer Vision*, volume 2, pages 1508–1511, 2005.

[20] M. Salzmann, J. Pilet, S. Ilic, and P. Fua. Surface deformation models for nonrigid 3D shape recovery. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 29(8):1481–1487, 2007.

[21] G. Schall, E. Mendez, E. Kruijff, E. Veas, S. Junghanns, B. Reitinger, and D. Schmalstieg. Handheld augmented reality for underground infrastructure visualization. *Personal and Ubiquitous Computing*, 13(4):281–291, 2009.

[22] G. Schweighofer and A. Pinz. Robust pose estimation from a planar target. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 28(12):2024–2030, 2006.

[23] G. Simon, A. Fitzgibbon, and A. Zisserman. Markerless tracking using planar structures in the scene. *Proc. Int. Symp. on Mixed and Augmented Reality*, pages 120–128, 2000.

[24] T. Thormählen, H. Broszio, and P. Mikulastik. Robust linear auto-calibration of a moving camera from image sequences. In *Proc. Asian Conf. on Computer Vision*, pages 71–80, 2006.

[25] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg. Pose tracking from natural features on mobile phones. In *Proc. Int. Symp. on Mixed and Augmented Reality*, pages 125–134, 2008.

[26] D. Wagner and D. Schmalstieg. ARToolKitPlus for pose tracking on mobile devices. *Proc. Comp. Vision Winter Workshop*, 2007.

[27] B. Williams, G. Klein, and I. Reid. Real-time SLAM relocalisation. *Proc. Int. Conf. on Computer Vision*, 2007.