Multiple Target Detection and Tracking with Guaranteed Framerates on Mobile Phones

Daniel Wagner[1], Dieter Schmalstieg[2], Horst Bischof[3]

Graz University of Technology

ABSTRACT

In this paper we present a novel method for real-time pose estimation and tracking on low-end devices such as mobile phones. The presented system can track multiple known targets in real-time and simultaneously detect new targets for tracking. We present a method to automatically and dynamically balance the quality of detection and tracking to adapt to a variable time budget and ensure a constant frame rate. Results from real data of a mobile phone Augmented Reality system demonstrate the efficiency and robustness of the described approach. The system can track 6 planar targets on a mobile phone simultaneously at framerates of 23fps.

KEYWORDS: Pose estimation, 6DOF, mobile phone, natural features

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

1. INTRODUCTION AND RELATED WORK

Pose tracking for Augmented Reality (AR) applications has very demanding requirements: It must deliver full six degrees of freedom, give absolute measurements with respect to a given coordinate system, be very robust and run in real-time. Recently, mobile phones have become increasingly attractive for AR. With the built-in camera as the primary sensor, phones facilitate intuitive point-and-shoot interaction with the environment. However, the limited computational capabilities of the mobile phone CPU require that algorithms become 1-2 orders of magnitude more efficient. Moreover, the fidelity of the user interface demands that the real-time characteristics of the pose tracking should not be affected by the image content. We will first review other work on pose tracking suitable for AR, and then state our contribution.

Kanade-Lucas-Tomasi [16] is the most prominent real-time keypoint tracking method. Alternatively, areas can be tracked under projective transformation, as demonstrated by Benhimane and Malis [2]. Both tracking approaches are still too computationally expensive for mobile phones. Instead, simpler methods based on optical flow have been presented; e. g., TinyMotion [25] tracks in real-time using optical flow, but estimates only a coarse 2D image movement. Similar systems are reported in [8], [9].

Recent improvements in keypoint matching have made interest-point based approaches popular for real-time detection and tracking. The location invariance of interest point detectors is attractive for localization without prior knowledge. However, computation of descriptors that are invariant across large view point changes is expensive. For instance, Skrypnyk and Lowe [21] use SIFT descriptors [15] for object localization in AR. On-line selection of features can be done from a model [3] or mapped from the environment at runtime [6][12]. Lepetit et al. [13] recast matching as a classification problem using a decision tree and trade increased memory usage with avoiding expensive descriptor matching at runtime. Ozuysal et al. [17] improve the classification rates while further reducing necessary computational work. Takacs et al. [23] implemented the SURF [1] algorithm for mobile phones. They do not target real-time pose estimation, but maximum detection quality. In a more recent approach [22] Ta et al. improve speed by tracking SURF features in scale space.

Multiple target tracking deals with the problem of not mixing up multiple objects that interact and overlap in the camera image. Typically particle filters are applied for motion and appearance estimation [4][26][27]. Other approaches use online learning to optimally distinguish multiple objects [28][7][19]. Closest to our work is [18], which also amortizes detection of new targets over multiple frames, but without real-time guarantees.

The work in this paper builds upon our previous publication [24], where we described modified SIFT [15] and Ferns [17] approaches and created the first real-time 6DOF natural feature tracking system running on mobile phones. This paper extends that work. It deals with computationally efficient methods that permit simultaneous tracking and detection of multiple targets on computationally weak platforms (Section 2 presents an overview of the tracking method). Unlike previous approaches, we pose simultaneous tracking and detection as an optimization problem which has to be dynamically solved in real-time (Section 3). Several heuristics are applied to ensure tracking is not lost and additional targets are detected if feasible. To our knowledge we are the first to report a system that attempts to address this problem under tight real-time constraints of mobile AR. Section 4 presents several examples on a mobile phone achieving frame rates up to 23fps whilst tracking 6 targets simultaneously.

2. DETECTION AND TRACKING SYSTEM OVERVIEW

Our pose tracking pipeline is split into distinct detection and tracking parts, which are implemented using different techniques and exhibit different timing characteristics.

[1]wagner@icg.tugraz.at
[2]schmalstieg@icg.tugraz.at
[3]bischof@icg.tugraz.at

The tracking system (left block in Figure 1) is the driving force always running at full frame rate, tracking the pose of targets that have previously been detected by the detection system. The tracking system creates a mask that instructs the detection system where to look at for new targets, thereby speeding up the detection task. The detection task only works on the unmasked image areas and uses a SIFT like approach to match keypoints against a feature database. It then checks the matches using geometrical constraints against outliers and estimates the pose of each newly detected target.

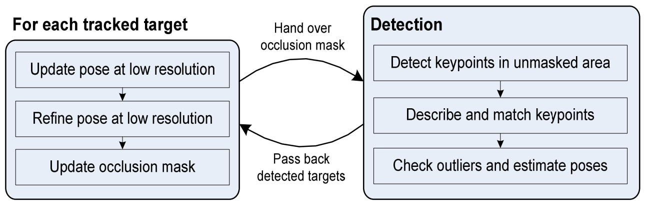

For each tracked target
Update pose at low resolution
Refine pose at low resolution
Update occlusion mask
Hand over occlusion mask
Pass back detected targets
Detection
Detect keypoints in unmasked area
Describe and match keypoints
Check outliers and estimate poses

Figure 1: Overview of the detection and tracking system.

In the following, we explain in detail how the detection and tracking systems work, how they complement each other in their strengths and weaknesses and how they collaborate for faster overall execution.

2.1. Detection

The aim of the detection system is to find new (previously learned) tracking targets in the camera image and to estimate a 6DOF pose, which is then forwarded to the tracking system. Our target detection method is based on a modified SIFT [15] implementation that replaces the slow parts of the original SIFT with simpler variants, yet keeping many of the attractive properties of the original approach. The detection system is similar to our work published in [24], but its working principles are repeated here for completeness and to report on improvements applied since then.

The original SIFT algorithm uses Difference of Gaussians (DoG) to perform a scale-space search that not only detects features but also estimates their scale. Although several faster implementations of Lowe's approach have been proposed, all these approaches are inherently resource intensive and therefore not suitable for real-time execution on mobile phones. We therefore replaced the DoG with the FAST corner detector [20] with non-maximum suppression that is known to be one of the fastest corner detectors, but still provides a high repeatability.

Since FAST does not estimate a scale, we quantize the scale space into discrete steps of $1/\sqrt{2}$ and search over all scales. Our tests showed that the SIFT descriptor is robust enough to also match features that fall between two scales. The descriptor database contains features from all scale steps over a meaningful range (typically 1:5). By describing the same feature multiple times over various scales, we trade memory for speed to avoid a CPU-intensive scale-space search. This approach is reasonable because of the low memory required for each SIFT descriptor.

Although Lowe presents several versions of the SIFT descriptor, the most popular version uses 4x4 sub-regions with 8 gradient bins each, resulting in a 128-dimensional vector. Conversely, we constrain computational and memory requirements by using only 3x3 sub-regions with 4 bins each, resulting in a 36-dimensional vector. Lowe reports this variant to be only ~10% below optimum.

Since we have fixed-scale interest points, we fix the SIFT kernel to 15 pixels. To gain robustness we blur the patch with a 3x3 Gaussian kernel. Like in the original implementation we estimate feature orientations by calculating gradient direction and magnitude for all pixels of the kernel. The gradient direction is quantized to 36 bins and the magnitude, weighted using a distance measure, is added to the respective bin. We compensate for orientation by rotating the patch using sub-pixel accuracy. For each rotated patch, gradients are re-estimated, weighted by distance to the patch center and to the sub-region center, and finally written into the 4 bins of their sub-region.

The descriptors for all features in the new camera image are created and matched against the descriptors in the database. For feature matching the original SIFT uses a k-d Tree with a Best-Bin-First strategy, but our tests showed that some (usually 1-3) entries of the vectors vary strongly from those in the database, tremendously increasing the required tolerance for searching in the k-d Tree, making the approach infeasible on mobile phones. The higher robustness requirements arise from the simplifications made in the previous steps, most notably the coarsely quantized scale space.

We perform descriptor matching using a forest of spill trees [14]. A spill tree is a variant of a k-d tree that uses an overlapping splitting area for higher robustness against noisy data. While a single spill tree turns out to be insufficient, multiple trees with randomized dimensions for pivoting allow for a highly robust voting process. When matching descriptors, we only visit a single leaf in each tree and merge the resulting candidates. Descriptors that are found in only one tree are discarded; all others are matched using Euclidean distance.

Outlier removal operates in two steps. First, the relative orientations of all matched features are corrected to the absolute rotation using the feature orientations stored in the database. Since the tracker is limited to planar targets, all features should have a similar orientation. Entering all orientations into a histogram and searching for a peak, a main orientation is estimated in linear time and used to filter out those features which do not support this hypothesis. The remaining matches are used in a PROSAC scheme [5] to estimate a homography between the model points of the planar target and the input image. The final homography is estimated from the inlier set and used as a starting point in a 3D pose refinement scheme described next.

The pose extracted from the inliers is coarse and needs to be improved using non-linear refinement. We apply a Gauss-Newton iteration to minimize the reprojection error of the 3D-2D point correspondences. For increased robustness we use a robust M-estimator [10] that weights the influence of all correspondences by their variance using a Tukey function.

$$e(C) = \min \sum_{i=1}^{N} w(r_i) \cdot r_i^2 \quad (1)$$

$$r_i(C, x_i) = p_i - \mathrm{proj}(C, x_i) \quad (2)$$

$$w(r_i) = \left(1 - (r_i/c)^2\right)^2 \; if \; |r_i| \le c \quad \text{or} \quad 0 \text{ otherwise} \quad (3)$$

$$c = \left(1.48261 + 5/(N-5)\right) \cdot \mathrm{median}(r_i) \quad (4)$$

For a set of *N* observations p_i of model points x_i, we minimize the overall observation error *e(C)* for a camera pose *C*, which is a weighted sum of individual squared residuals r_i. The residuals are defined as the offset between the observation p_i and the respective projection $proj(C, x_i)$ of the model point x_i. Each squared residual is weighted by the Tukey function *w*.

2.2. Tracking

The target detector described above treats each image independently: Keypoints are detected, matched and used to estimate the camera pose. Frame-to-frame coherence is not exploited. In contrast to the detector, the tracker uses a purely active search approach: Based on a pose prior and a motion model, it estimates coarsely where to look for previously observed features and what locally affine feature transformation to expect. This approach removes the requirement for a descriptor that is invariant to affine changes. This is more efficient than tracking-by-detection because it makes use of the fact that both the scene and the camera pose change only slightly between frames.

The tracker uses a reference image of the tracking target, stored at multiple scales, as the only data source. During initialization keypoints are detected in the reference image using a corner detector. The image is stored at multiple scales to avoid aliasing effects during large scale changes.

Starting with a pose prior (either from the detector or from the previous frame) the tracker searches for known features at predicted locations in the camera image. The new feature locations are calculated by projecting the keypoints of the reference image into the camera image.

After the new feature positions have been estimated, they are searched within a predefined search region of constant size. Using the pose prior, the tracker creates affinely warped 8x8 pixel representations of the features from the reference image. The exact feature locations are determined using normalized cross correlation (NCC) over a predefined search area. For each good match, we perform a quadratic fit into the NCC responses of the neighboring pixel positions to achieve sub-pixel accuracy.

Template matching over a search window is fast, if the search window is small. However, a small search window limits the speed at which camera motion can be detected. To increase the search radius, we use a hierarchical approach. Similar to Klein [12], we estimate the new pose from an image scaled down by 50%. Only few, randomly selected feature positions are searched at this level, but with a large search radius. We employ the same pose refinement method as for the detector, using the pose prior as a starting point (rather than a pose from homography) for the iterative refinement. After a pose has been estimated for the low resolution image, it is further refined using the full resolution image with a larger number of feature points, but with a smaller search radius.

Additionally, we use a motion model for each target to predict the camera's pose (translation and rotation) from the previous frame. Our motion model is linear, calculating the difference between the poses of the previous two frames in order to predict the current pose. This model works well, as long as the camera's motion does not change drastically. As our tracker typically runs at 20Hz or more, this is a valid assumption in most cases.

2.3. Combined Detection and Tracking

The detection and tracking tasks described above are designed to work together and have complementary strengths (Table 1). The detection system can detect new targets in a camera image and estimates the camera's pose relative to these targets without prior knowledge on the exterior camera parameters.

System	Detection	Tracking
Detect targets	+	-
Initialize tracking	+	-
Speed	-	+
Robust to blur	-	+
Robust to tilt	~	+
Robust to illumination changes	~	+

Table 1: Strengths and weaknesses of the detection and tracking.

The tracking system works with a pose prior and therefore knows where to search; consequently it runs much faster than the detection task. The combination of avoiding a keypoint detector, affinely warping patches and normalized cross correlation for matching makes the tracker very robust: Due to using NCC, the PatchTracker is robust to global changes in lighting, while the independent matching of many features increases the chance of obtaining good matches, even under extreme local lighting changes and reflections (Figure 2a). Because of the affinely warped patches, it can track under extreme tilts close to 90° (Figure 2b), whereas the detector has problems handling tilts larger than ~45°. The absence of a keypoint detection step also makes it moderately robust to blur (Figure 2c). Finally, the tracker is very fast, requiring only ~1ms on an average PC and ~6ms on a fast mobile phone per target in typical application scenarios.

2.4. Speeding up Detection by Tracking

As noted above, the time required for detection largely depends on the camera resolution and number of keypoints detected in the camera image. More precisely, it depends on the number of pixels to be searched and the number of pixels that are classified as keypoints. Reducing these numbers by re-using information from the tracking part represents an efficient way to speed up the recognition task.

Our method is based on the simple idea of omitting areas covered by already tracked keypoints. A possible approach is to project the shape of the tracking target into the camera image in order to mask out the keypoints inside this area. However, such an approach does not work when tracking targets overlap in image space. For instance, a small target in front of a larger target cannot be detected, since the area of the large target completely surrounds the area of the small target.

Instead, we adopt a splatting technique that is based on the assumption that after a keypoint has been positively matched

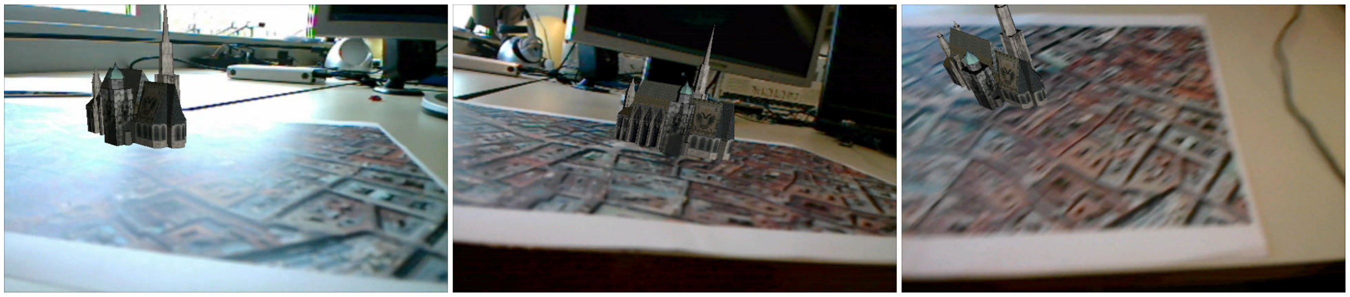


Figure 2: a) Tracking despite reflections; b) Tracking under severe tilt; c) Tracking despite strong blur

during tracking process, we can infer that a small area around that keypoint is visible in the camera image (see Figure 3). Hence, for each keypoint successfully matched, we paint (splat) a small square into the mask, covering the area around that keypoint in the camera image. The size of the square depends on the scale the feature originates from: features from level one (low-res) result splats twice as large as feature from level zero (high-res). Since all keypoints are randomly distributed over the target, this method nicely covers large areas of the tracked targets. Areas not covered are usually weak in texture and therefore do not create responses during keypoint detection.

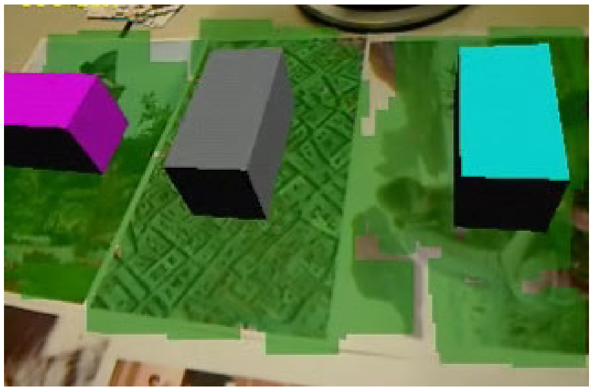


Figure 3: Masks (visualized in green color) created by the tracking task for speeding up the detection.

Using an OR operation, the masks from all tracked targets are combined and forwarded to the keypoint detector, which then only processes those pixels that are not masked. As can be seen in Figure 4 this method can vastly reduce the number of detected keypoints and therefore speed up the overall detection process.

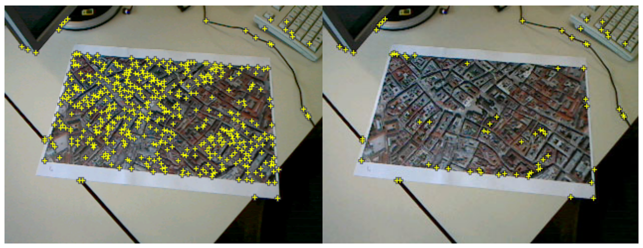


Figure 4: Keypoints detection without image mask (left, 486 keypoints) and with image mask (right, 70 keypoints).

2.5. Analysis of Speed and Predictability

We implemented different code paths for PCs and mobile phones. On the PC we use floating point, whereas on the mobile phone we always use a fixed point code path. Even though some modern mobile phones support floating point in hardware today, our tests showed that fixed point is still faster for our applications. We assume that this is related to a suboptimal intermixing of integer and floating point code in the current generation of compilers for mobile phones. We expect this situation to improve as mobile phones will increasingly support hardware floating point in the future. Except when noted otherwise, all tests and evaluations presented in this paper were performed in floating point on the PC and fixed-point on the mobile phone.

The speed of the detection system is mainly dependent on the camera resolution and the number of keypoints detected. Additionally, speed is influenced by the size of the feature database and the number of outliers.

The tracker's speed depends largely on the number of keypoints to match and the size of the search window; speed can therefore by be directly influenced. The tracking time for each target is independent of number of targets tracked. Typically, detection for one image takes about 3-5 times longer than tracking of a single target.

In order to detect and track while obeying a fixed time budget, we must be able to predict the time consumption of each algorithm. The keypoint detection using FAST corners can be well predicted, since the average speed per scan line is rather stable. Keypoint description using SIFT-like gradient histograms is also stable. Descriptor matching compares all descriptors against the feature database and assigns the matches to tracking targets. Again, the average time to match a descriptor can be well predicted. Outlier removal takes the matches of all those targets that are not tracked yet and performs various geometrical tests (consistent keypoint rotation and spatial tests based on homography). The time required for the outlier removal varies significantly, depending how early a tracking target is rejected. Finally, pose refinement (see section 2.1) takes the pose and the inlier set and performs a non-linear refinement. Since the prior pose is usually of good quality, only few iterations (typically 3) are required and the task therefore takes near constant time.

Since we keep the size of the search windows constant, the speed of the tracker directly depends on the number of keypoints to match. In our two-level approach, we always match 25 keypoints at half camera resolution and between 30 and 100 keypoints at full resolution. The time for matching at half resolution is therefore mostly constant while the time for matching at full resolution changes almost linearly with the number of keypoints to match.

In summary, the time consumption of most steps of the algorithm can be predicted well with the exception of the outlier removal step.

3. Combined Tracking and Detection at Guaranteed Frame Rates

A system that supports multiple tracking targets needs to run both the detection as well as the tracking task simultaneously: While the tracker estimates the poses of all previously detected targets until they are lost, the detector needs to look out for new targets, which are then forwarded to the tracker. Rendering, image acquisition and system overhead add up to the overall frame time.

The aim of the system is to keep the overall frame time constant, e.g. 50ms for a system running at 20Hz. The tracking system therefore has to adapt in order to fit into the time slice that is left after subtracting the application (rendering, system) and the image acquisition duration from the overall frame duration. While the application time can be accurately estimated, the image acquisition time is more problematic. Many cameras in mobile phones can only deliver the selected frame rate if the camera system process receives enough CPU time, by making the foreground task suspend to idle mode until the camera image is ready. The idling interval is hard to determine, as it depends on many factors including CPU, memory and bus load. If too little CPU time is dedicated to the camera, the camera frame rate drops. Only the effect of too little camera CPU time can be measured, whereas too much camera CPU time is simply spent with waiting for the next camera image to become available. Figure 5 shows an overview of this problem for a single frame. The aim is to minimize the "idle time for camera" to a level that does not reduce the camera frame rate.

To be able to measure the effect in both directions, we target a frame rate that is slightly lower than the maximum camera rate. The tracking time budget is then adapted dynamically to fit this frame rate. If the overall frame rate is higher than anticipated we can increase the tracking time; if it is lower then we have to decrease it.

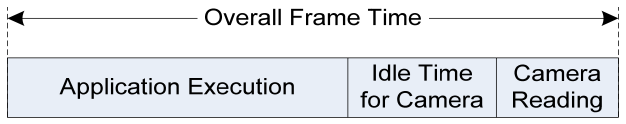


Figure 5: Execution of the application has to leave enough idle time for the system to read the camera image.

This problem can be outlined as follows: Once the overall tracking time budget T_v is determined, the tracking system can adapt its operations in order to detect and track within the available time frame. On a fast mobile phone, such as used in our tests (Section 4), tracking requires about 6-8ms per target, whereas the recognition process takes about 30ms for a full camera image. Consequently, given an overall budget of e.g. 30ms per frame (computed as 50ms per frame minus 20ms for the application), detection and tracking of multiple targets cannot both be executed without optimization.

Tracking previously detected targets has the highest priority and must be running at full frame rate if possible at all. Detection of new targets has lower importance and is not required to run at full frame rate. Instead, it is sensible to amortize the cost of detection over multiple frames.

We can formulate the task of scheduling the optimal detection and tracking within a certain time budget as a constraint optimization problem. More formally, we define Cost and Benefit heuristics for the tracking of target t_i and the detection task.

Cost(t_i, q_i, p_i) is the time it takes to track target t_i at quality q_i; p_i is the pose prior. The quality in this metric is assumed to be proportional to the number of feature points from the given target selected for warping and matching in the active search. The cost of not tracking target t_i is set to be very high (higher than detection). Thereby we ensure that tracking has always a higher priority than detection.

Benefit(t_i, q_i, p_i) is an estimate for the contribution of tracking target t_i at quality q_i; p_i is the pose prior. This benefit value is obviously higher (better accuracy) with increased quality, but also takes into account the target's number of features and the pose change rate derived from the motion model. Small pose changes require fewer iterations in the pose computation, while large pose changes often introduce blur, which makes the matching more computationally intensive.

Cost(d) is an estimate of the time it takes to perform a certain fraction *d* of a detection task. This is proportional to the number of features in the area not covered by the image mask derived from tracking the targets in the current frame.

Benefit(d) is an estimate of the contribution of performing a certain amount *d* of the detection task in this frame. This contribution depends on the number of frames still required to complete the detection, assuming the currently chosen amount of the task is kept over the next frames. For example, if n candidate feature pixels must be examined, choosing an amount of 0.3*n* leads to upper(1/0.3)=4 frames, whereas choosing *n*/3 leads to only 3 frames estimated completion time for the detection. Shorter completion times receive significantly higher benefit.

Using this notation, the optimization can be stated as

$$\max_{q_i,d} \sum_i Benefit(t_i,q_i,p_i) + Benefit(d) \quad s.t. \quad \sum_i Cost(t_i,q_i,p_i) + Cost(d) \leq T_V \tag{5}$$

This constrained optimization problem is a variant of the NP-complete Multiple Choice Knapsack Problem [11]. This kind of Knapsack Problem allows to select ("pack") only one from a number of options for each item, in our case the quality of the tracking and the amount of detection. Since we have very few items (typically <10 tracking targets), a simple greedy heuristic suffices:

1. Maximize the cumulative relative value (*Benefit/Cost*) for all tracking targets while staying within frame time
2. Assign the remaining time to detection
3. Try to increase detection time at the expense of tracking quality until the overall benefit does no longer increase. This is done in discrete steps that reduce the number of frames until completion.

Unfortunately, some of the variables used in the optimization procedure cannot be estimated reliably. In particular, the later steps in the detection such as the outlier removal depend on the keypoint detection in the earlier steps. The overall outlier removal time varies strongly depending on how early a potential target is rejected.

Therefore we repeat or refine the planning at several stages within one frame's work. After each step, the system will determine its used time and make sure that only as much work is scheduled as will likely fit into the time left, e.g., a limited number of image scan lines to search for keypoints. Likewise, any amount of detection left over after time runs out will be re-scheduled to continue at the next frame's detection slot.

4. EXPERIMENTAL RESULTS

The following experiments were all run with the same configuration for detection and tracking. The keypoint detector was configured to adjust its threshold dynamically to detect keypoints for 0.5% of the number of pixels in the unmasked areas. The maximum allowed Euclidean distance for descriptor matching was set to 0.12 (using normalized vectors). We created a feature dataset of 8 targets with 15313 descriptors altogether, indexed by 3 spill trees resulting in an overall detection dataset size of 1.75 megabytes (777 KB for the descriptors and 340 KB for each tree).

For the tracker we used grayscale reference images, sized at 480 pixels for the longer side summing up to 1.23 megabytes of image data. We configured the tracker to match 25 points at half resolution with a search radius of 5 pixels and up to 100 points at full resolution with a search radius of 2 pixels only. Hence, for a frame rate of 20Hz, the system can track interest points at a speed of up to 5·2·20 = 200 pixels (5 pixels radius at half resolution at 20Hz) per second plus the compensation provided by the motion model of the tracker.

For both the tracker and detector the pose refinement was configured to iterate at least three times and then break as soon as the reprojection error improves less than 5%.

4.1. Artificial Transformations

To test the accuracy of the tracker, we created a sequence of 61 frames rendered in 3dsMAX showing a target coming into the image from the right side and leaving to the left (see Figure 6). The renderings were done a using a virtual camera with a field of view of 45° at a resolution of 320x240 pixels. The camera was set at the center of the scene, whereas the target sized 480x272 units moved on a circular path at a distance of 750 units.

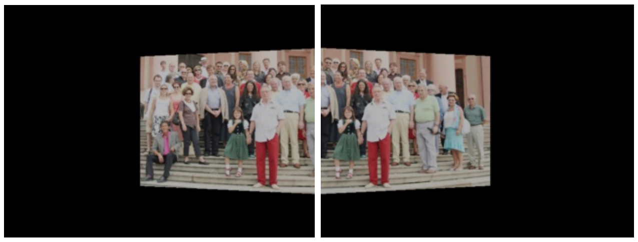


Figure 6: Artificial transformation for accuracy tests. Left: Frame 17; Right: Frame 51.

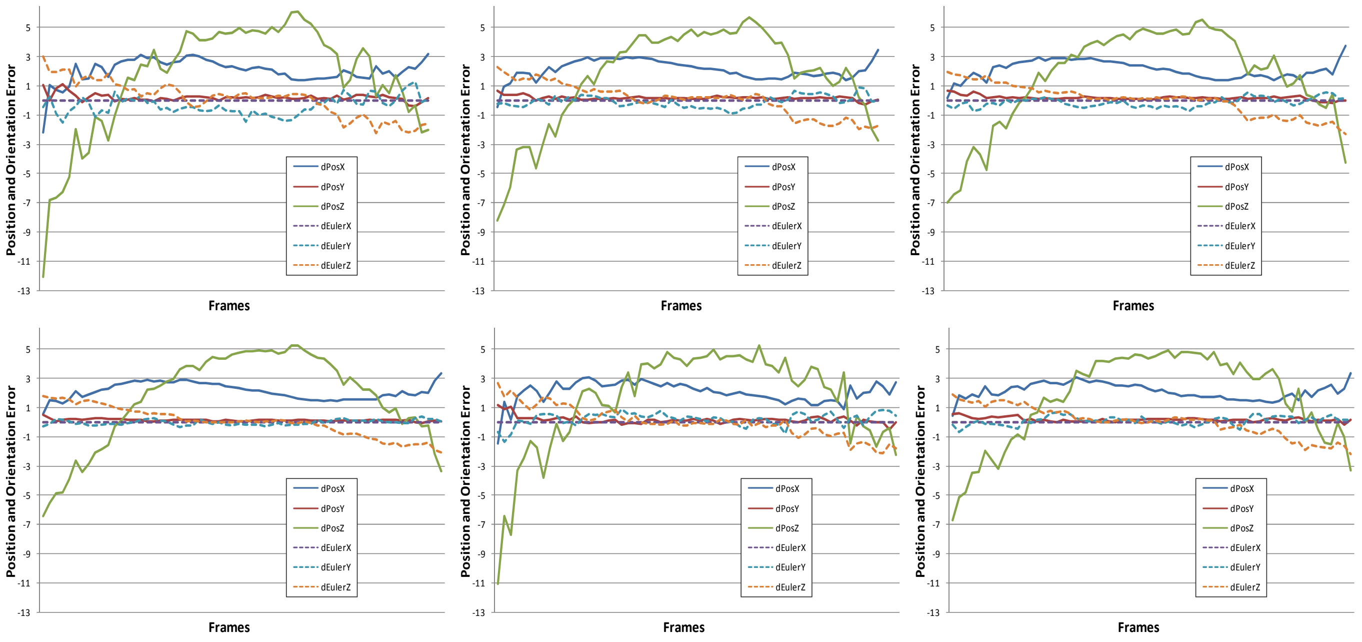
Figure 7 shows the position and orientation errors of various configurations with respect to the ground truth from the scene settings in 3dsMAX. As to be expected the z-coordinate suffers from the largest positional error, whereas the x-values are good and the y-values are flawless with an error of 0.0 (note that the target moved horizontally only at the middle of the screen). Although the average errors are similar, the setups with more correspondences suffer less from jitter, noticeable as smoother lines in the charts of Figure 7. One can clearly see that in the middle of the sequence, where the target is fully visible in the camera image, the jitter is minimal for all configurations.

It is interesting to notice that for all configurations the error trend of the z-coordinate is highly similar: In the beginning, the target is estimated too far away and then transitions to a pose that is too close. Similar trends can also be found other attributes dPosX, dPosY, dEulerX, dEulerY and dEulerZ. We don't have an explanation for this behavior, but it is obviously system immanent rather than an issue due to limited precision in pose estimation.

4.2. Visual Jitter

We created a real setup (see Figure 8) that exaggerates the jitter an Augmented Reality user would perceive using our tracker. In this setup, we placed a virtual pole on top of a tracked printout and placed the camera such that the tip of the virtual pole is close to the camera. Due to the large distance of the target and the close distance to the camera, the visual jitter at the tip of the virtual object is strongly amplified. Since the scene is static, the pole should remain perfectly still in an ideal case. While the camera feed is 320x240 pixels, we rendered the virtual pole at 640x480 for improved measurement accuracy. The results can be viewed in the supplemental video.

Additionally we estimated the reprojection error at the tip of the pole by sampling 1000 camera frames, calculating the average position, standard deviation and maximum derivation. As before, all calculations were done for a screen resolution of 640x480



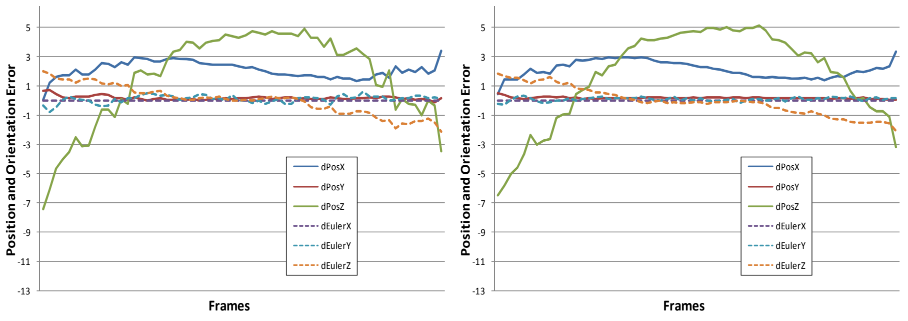


Figure 7: Position and orientation (in degrees) errors from a moving synthetic target over 61 frames.
1st line: 30, 65 and 150 floating point correspondences. 2nd line 300 floating point correspondences and fixed point with 30 and 65 correspondences. 3rd line: 150 and 300 fixed point correspondences.

pixels. The whole sequence of tests was repeated 3 times and the averages of all measurements were taken.

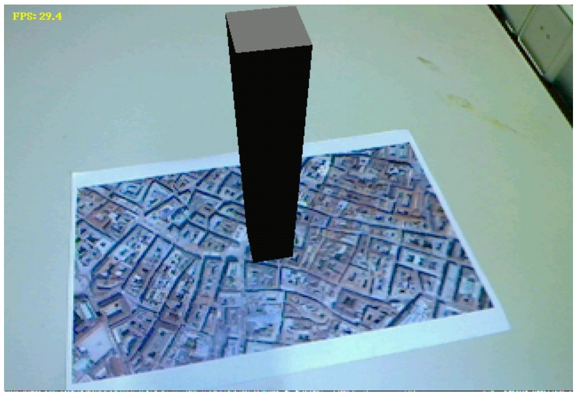


Figure 8: Setup for visual jitter estimation.

The chart in the right image of Figure 9 shows the results for 1000, 300, 100, 65 and 30 correspondences. At 100 correspondences or below, the quality of fixed point decreases. In all cases, standard deviation of the reprojection error is <0.6 pixels (fixed) and <0.2 pixels (floating).

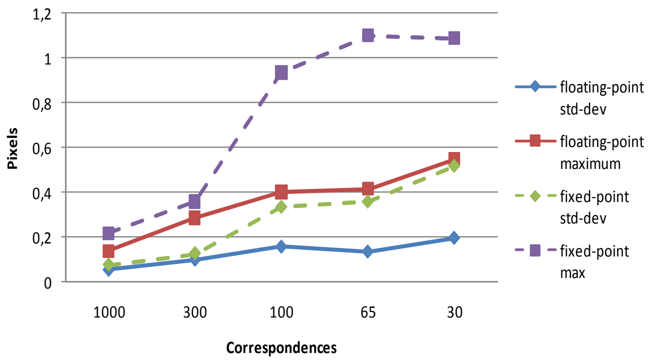


Figure 9: Reprojection error at the tip of the pole in the left image.

4.3. Tests on a Mobile Phone

We tested our approach on the Asus P565, a Windows Mobile phone with 800MHz and a camera delivering up to 30 images per second at a resolution of 320x240 pixels. The target frame rate of the system was set to 23 frames per second. The screenshots in Figure 10 show the progress of detecting and tracking one to six independent targets concurrently. The green transparent areas encode those parts of the screen that are tracked and therefore not searched for during the detection step. Each detected target is augmented with a cuboid. All targets are treated independent of each other.

At the bottom of the screen the application displays feedback data. Line 1 reports how much time of the estimated vision budget was used in the current frame. Line 2 tells how much time was spent for tracking, how many targets were tracked and which quality setting was used. Line 3 reports how much time was spent for the detection task and over how many frames this work was distributed.

One can notice in the top left image of Figure 10 that tracking a single target leaves enough time left for detection of new targets within a single frame only. This is due to a large area of the camera image being masked out by the tracking system. The 2nd image shows tracking of 2 targets, which requires about 20ms, still leaving enough time to the detection system to finish its task in a single frame most of the time. The 3rd image shows tracking of 3 targets, now heavily reducing the detection time budget. In the first image of the second row, the tracking system starts reducing tracking quality in order to keep up the frame rate. There is only little time left for detection, but most of the image is covered by the tracked area. The next screenshot shows that the tracking system has degraded the tracking quality even more to maintain the target frame rate. Finally, in the last image, six targets are tracked, which leaves only little time for detection despite the strongly reduced tracking quality. The system would not be able to track another target once detected (please see supplemental video).

The tests show that the mobile phone can track and detect up to six targets while maintaining a frame rate of 23 fps or more. In this test we deliberately chose simple virtual objects (cuboids) that are fast to render in order to estimate how far the tracking system can go.

5. Conclusions

We have presented a new method for detection and tracking of multiple planar targets in real-time on low end devices. Our system detects and tracks up to six targets with 23 frames per second a mobile phone in 6DOF while augmenting them at the same time. Splitting up detection and tracking in two separate subsystems that work at different frame rates allowed the creation of a highly efficient system that is several orders of magnitude faster than any other 6DOF tracker reported so far.

We observe that the level of CPU performance on phones has not increased a lot in the last three years, probably because of a certain market saturation and the very tight power budget afforded by cell phone batteries. Instead, mobile phones with programmable GPUs are now becoming available. Especially the tracking subsystem is made up of basic operations (warping and template matching) that are suitable even for the rather simple shading units of first generation programmable mobile phone GPUs.

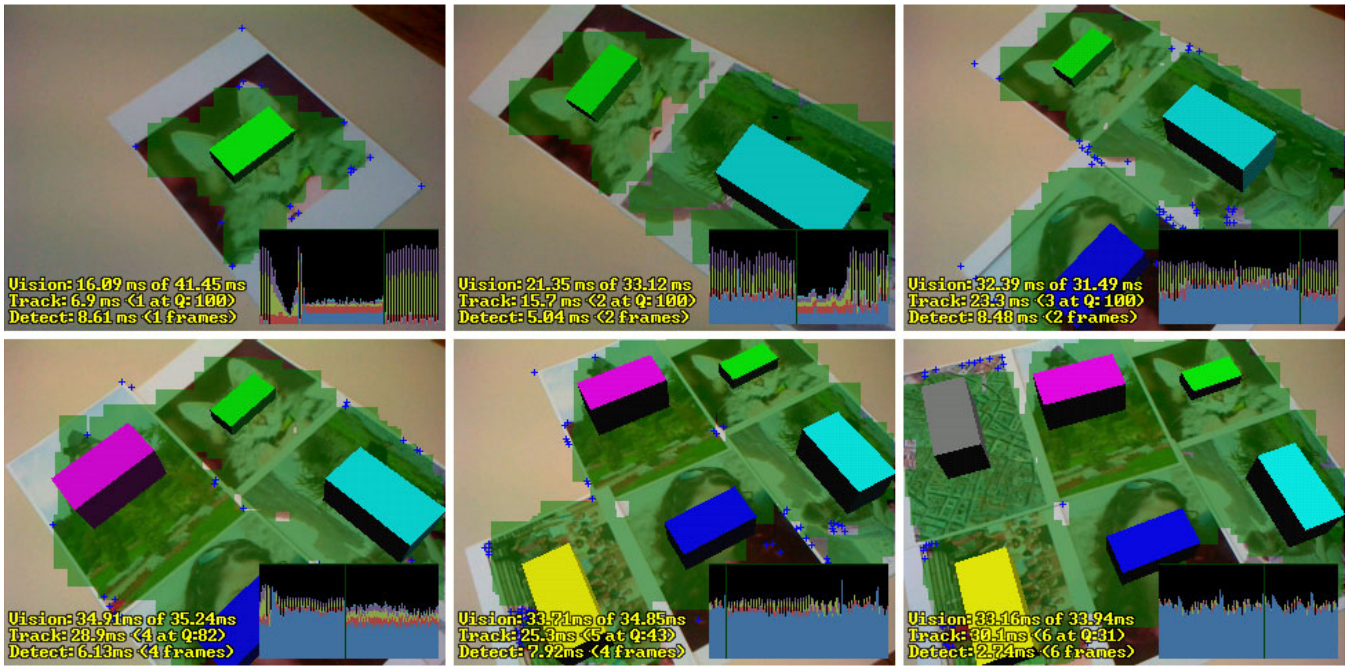
In the future we plan to extend our system to true 3D targets thereby allowing tracking of more complicated targets such as real 3D objects or the environment of a mobile user. Estimating a homography would then not suffice anymore. Furthermore, it would be necessary to cope with self-occlusions of the tracking target.

6. Acknowledgements

The authors thank Gerhard Reitmayr for discussions regarding the pose refinement step and the reviewers for their many suggestions. This research was funded by the Austrian Science Fund FWF under contracts Y193 and W1209-N15, and the Christian Doppler Research Association.

References

[1] Bay, H., Tuytelaars, T., Gool, L. V., Surf: Speeded up robust features, In Proc. ECCV 2006, 2006.
[2] Benhimane, S., Malis, E., Real-time image-based tracking of planes using Efficient Second-order Minimization, In Proc. of International Conference on Intelligent Robots and Systems, Vol. 1, pp. 943-948, 2004
[3] Bleser, G., Stricker, D., Advanced tracking through efficient image processing and visual-inertial sensor fusion. In Proc. IEEE VR 2008, pp. 137-144 2008
[4] Cai, Y., Freitas, N., Little, J.J., Robust Visual Tracking for Multiple Targets. In Proc. ECCV2006, pp. 107-118, 2006



```
Vision: 16.09 ms of 41.45 ms
Track: 6.9 ms (1 at Q: 100)
Detect: 8.61 ms (1 frames)
```

```
Vision: 21.35 ms of 33.12 ms
Track: 15.7 ms (2 at Q: 100)
Detect: 5.04 ms (2 frames)
```

```
Vision: 32.39 ms of 31.49 ms
Track: 23.3 ms (3 at Q: 100)
Detect: 8.48 ms (2 frames)
```

```
Vision: 34.91ms of 35.24ms
Track: 28.9ms (4 at Q:82)
Detect: 6.13ms (4 frames)
```

```
Vision: 33.71ms of 34.85ms
Track: 25.3ms (5 at Q:43)
Detect: 7.92ms (4 frames)
```

```
Vision: 33.16ms of 33.94ms
Track: 30.1ms (6 at Q:31)
Detect: 2.74ms (6 frames)
```

Figure 10: Screenshots from the test application running on a mobile phone. The graph in the right bottom of each screen shows where the vision pipeline spends its time. From bottom to top: Tracking (blue), keypoint detection (red), keypoint description (green), matching (purple) and removal and pose estimation (cyan). Black represents the time left for the rest of the system. The yellow text reports: 1st line: how much time of the overall time budget has been used; 2nd line: how long tracking took, the number of tracked targets and the tracking quality; 3rd line: detection time split of number of frames.

[5] Chum, O., Matas, J., Matching with PROSAC Progressive Sample Consensus, In Proc. CVPR'05 - Volume 1, 220 - 226, 2005

[6] Davison, A.J., Mayol, W.W., Murray, D.W., Real-time localisation and mapping with wearable active vision. In Proc. of IEEE ISMAR 2003, pp. 18-27, 2003

[7] Grabner, M., Grabner, H., Bischof, H., Real-time tracking via on-line boosting. In Proc. of British Machine Vision Conference (BMVC), Volume I, pp. 47-56, 2006

[8] Hannuksela J., Sangi P. and Heikkilä J., A Vision-Based Approach for Controlling User Interfaces of Mobile Devices, In Proc. CVPR, Workshop on Vision for Human-Computer Interaction (V4HCI), 2005

[9] Haro, A., Mori, K., Setlur, V., Capin, T., Mobile Camera-based Adaptive Viewing, In Proc of 4th International Conference on Mobile Ubiquitous Multimedia, MUM 2005

[10] Huber, P.J., Robust estimation of a location parameter. In Annals of Mathematical Statistics, pp 73-101, 1964

[11] Ibaraki, T., Hasegawa, T., Teranaka, K., and Iwase J. The Multiple Choice Knapsack Problem. J. Oper. Res. Soc. Japan 21, pp. 59-94, 1978

[12] Klein, G., Murray, D., Parallel tracking and mapping for small ar workspaces. In Proc. of ISMAR 2007, pp. 225-234, 2007

[13] Lepetit, V., Lagger, P., Fua, P., Randomized trees for real-time keypoint recognition. In Proc. CVPR 2005, pp. 775-781, 2005

[14] Liu, T., Moore, A.W., Gray, A., Yang, K., An investigation of practical approximate nearest neighbor algorithms. In Advances in Neural, In Information Processing Systems, MIT Press, pp. 825-832, 2004

[15] Lowe, D., Distinctive image features from scale-invariant keypoints. Int. Journal of Computer Vision, Volume 60, Issue 2, pp. 91-110, 2004

[16] Lucas, B.D., Kanade, T., An iterative image registration technique with an application to stereo vision. In Proc. of 7th International Conference on Artificial Intelligence, pp. 674-679, 1981

[17] Ozuysal, M., Fua, P., Lepetit, V., Fast keypoint recognition in ten lines of code. In of Proc. CVPR 2007, pp. 1-8, 2007

[18] Park, Y., Lepetit, V., Woo, W., Multiple 3D Object tracking for augmented reality, In Proc. of ISMAR2008, pp. 117-120, 2008

[19] Pernkopf, F., Tracking of Multiple Targets Using On-Line Learning for Appearance Model Adaptation, In Proc. of International Conference on Image Analysis and Recognition (ICIAR07), pp. 602-614, 2007

[20] Rosten, E., Drummond, T., Machine learning for high-speed corner detection. In Proc. of 9th European Conference on Computer Vision (ECCV2006), pp. 430-443, 2006

[21] Skrypnyk, I., Lowe, D., Scene modeling, recognition and tracking with invariant image features. In Proc. of ISMAR 2004, pp. 110-119, 2004

[22] Ta, D.-N., Chen, W.-C., Gelfand, N., Pulli, K., SURFTrac: Efficient Tracking and Continuous Object Recognition using Local Feature Descriptors, Conference on Computer Vision and Pattern Recognition (CVPR'09), pp. 2937-2943, 2009

[23] Takacs, G., Chandrasekhar, V., Gelfand, N., Xiong, Y., Chen, W.-C., Bismpigiannis, T., Grzeszczuk, R., Pulli, K., and Girod, B., Outdoors Augmented Reality on Mobile Phone using Loxel-Based Visual Feature Organization, In Proc. of the 1st ACM international conference on Multimedia information retrieval, pp. 427-434, 2008

[24] Wagner, D., Reitmayr, G., Mulloni, A., Drummond, T., Schmalstieg, D., Pose Tracking from Natural Features on Mobile Phones, In Proc. IEEE ISMAR2008, pp. 125-134, 2008

[25] Wang, J. Zhai, S., Canny, J., Camera Phone Based Motion Sensing: Interaction Techniques, Applications and Performance Study, In ACM UIST 2006, pp. 101-110, 2006

[26] Yang, C., Duraiswami, R. Davis, L., Fast Multiple Object Tracking via a Hierarchical Particle Filter, In Proc. ICCV'05, pp. 212-219, 2005

[27] Yu, T., Wu, Y., Collaborative Tracking of Multiple Targets, In Proc. CVPR'04, pp.834-841, Volume 1, 2004

[28] Yuan, X., Li, S.Z., Learning Feature Extraction and Classification for Tracking Multiple Objects: A Unified Framework, In Proc. of the IEEE International Conference on Video and Signal Based Surveillance (AVSS'06), pp.22-27, 2006