Chapter 11
Markerless Tracking for Augmented Reality

Jan Herling and Wolfgang Broll

Abstract Augmented Reality (AR) tries to seamlessly integrate virtual content into the real world of the user. Ideally, the virtual content would behave exactly like real objects. This requires a correct and precise estimation of the user's viewpoint (respectively that of a camera) with respect to the coordinate system of the virtual content. This can be achieved by an appropriate 6-DoF tracking system.

In this chapter we will present a general approach for a computer vision (CV) based tracking system applying an adaptive feature based tracker. We will present in detail the individual steps of the tracking pipeline and discuss a sample implementation based on SURF feature descriptors, allowing for easy understanding of the individual steps necessary upon building your own CV tracker.

1 Introduction

The geometric registration of the virtual content within the real environment of the user is the basis for real Augmented Reality (AR) applications (in contrast to pseudo AR, where the real world content is just superimposed by floating bubbles, arrows and text messages). Correct geometric registration allows the observer to accept virtual content as enhancement of the real environment rather than a separated or overlaid layer of information. The virtual scene is geometrically registered by measuring or estimating the current pose of the user's view point or camera with respect to the coordinate system of the virtual content. This may either be done on a per object basis applying relative transformations, or all or at least several virtual objects share a coordinate system, thus the transformation to that is estimated.

J. Herling (✉)
Department of Virtual Worlds and Digital Games, Ilmenau University of Technology, Ilmenau, Germany
e-mail: jan.herling@tu-ilmenau.de

B. Furht (ed.), *Handbook of Augmented Reality*, DOI 10.1007/978-1-4614-0064-6_11,

This requires an appropriate tracking system, which either may be sensor based (e.g. applying inertial, gyroscopic or magnetic sensors, using ultrasonic sound, etc.) or computer vision (CV) based.

Computer vision based tracking can roughly be distinguished in marker based tracking and markerless tracking. While marker based tracking applies fiducial markers [8], which can be used to determine the camera pose in relation to each marker, providing a robust tracking mechanism, markers also come along with a couple of drawbacks. The major restriction is that they often cannot be attached to the objects which are supposed to be augmented. While this problem might be overcome by using two or more cameras (one for tracking the markers and one for the image to be augmented) this significantly reduces the overall tracking quality and by that the quality of the geometrical registration. Additionally such an approach is not feasible for most mobile devices like mobile phones. Further, attaching markers in certain environments may be considered as some type of environmental pollution. Finally, applying markers at all locations to be potentially augmented can be quite time-consuming, requiring a very large number of markers, in particular in outdoor scenarios.

Thus, markerless tracking approaches using natural features of the environment to be augmented for tracking is a much more promising approach. However, until recently, performance of suitable algorithms were not sufficient on mobile devices.

In this chapter we will provide an overview of current markerless tracking approaches and explain in detail the mechanisms involved for creating an appropriate computer vision based tracking system. We will also provide a more elaborated example based on the SURF feature descriptor to allow the reader to understand the steps involved and to show possible areas for optimization.

2 Feature Detection

Visual markerless pose trackers mainly rely on natural feature points (often also called interest points or key points) visible in the user's environment. To allow for an accurate pose determination such natural feature points must meet several requirements:

- **Fast computational time**
  It must be possible to calculate a rather large number of features points or even feature points and associated descriptors in real-time in order to allow for a pose estimation at an acceptable frame rate.
- **Robustness with respect to changing lighting conditions and image blurring**
  Set of feature points as well as their calculated descriptors must not vary significantly under different lighting conditions or upon image blurring. Both effects are quite common, in particular in outdoor environments, and thus any susceptibility against those would render the overall approach useless.

- **Robustness against observation from different viewing angles**
  In AR environments users are usually not very much restricted regarding their position and orientation, and by that regarding the viewing angle under which the feature points will be observed. Thus, any vulnerability with respect to the viewing angle will make such feature points pretty useless for AR.
- **Scale invariance**
  In AR, objects providing the necessary feature points are often observed from different distances. Tracking will have to work in a wide range and must not be limited to a certain distance only as users will typically want to spot AR content and approach it if necessary – in particular in outdoor environments. While a closer view will almost always reveal a higher number of feature points on a particular object, scale invariance refers to the fact that feature points visible from a rather large distance will not disappear when getting closer, allowing for a continuous tracking event without applying SLAM-like approaches.

In the last decade numerous feature detectors have been proposed providing different properties concerning e.g. detection robustness or detection speed. However, existing detectors basically can be distinguished into two different classes. First, very efficient corner detectors spotting corner-like features. Second, blob detectors not spotting unique corner positions but image regions covering blob-like structures with a certain size.

On the one hand, it holds that corner detectors are more efficient than blob detectors and allow for a more precise position determination. On the other hand, corner features typically lack of scaling information, resulting in an additional effort within the tracking pipeline.

2.1 Corner Detectors

The Harris detector [6] uses the approximated auto-correlation function for detecting interest point. The auto-correlation function determines the local difference in the image intensity for the direct neighborhood of pixels of interest. For each pixel of a frame an eigenvalue analyses of the auto-correlation matrix is applied to classify the pixel into different feature categories. Thus, strong corners can be distinguished from edges or homogenous image regions.

A different corner detector has been proposed by Rosten and Drummond [16] avoiding the eigenvalue analysis and thus performing significantly faster. The FAST feature detector compares the pixel value of the potential feature point with all pixels lying on a surrounding circle. The circle is selected to have a radius of 3 pixels resulting in 16 pixel values to be considered at most. A FAST feature is detected if the absolute intensity differences between the center pixel and at least 12 adjacent circle pixels are higher than a defined threshold. These intensity differences can be used to calculate the strength of features. Adjacent feature points are erased by a non-maximum-suppression search to determine unique feature positions. Due to the detection algorithm of FAST features, those cannot provide a dimension

parameter, but provide a unique pixel position. Thus, FAST features are not scale invariant being a drawback which must be compensated in the tracking pipeline with additional effort. However, the FAST detector is one of the fastest algorithms to find robust natural interest points. Thus, very often the detector is applied on mobile platforms with reduced computational power like e.g. mobile phones.

2.2 Blob Detectors

In contrast to corner detectors, blob detectors search natural feature points with a blob-like shape. Those interest points can be e.g. small dots or large blurred spots with similar color intensity. Most of these features are detected using a Gaussian filtering as bases. It has been shown that the Gaussian kernel performs best under the condition that features have to be scale invariant.

Commonly, blob detectors spot feature candidates within several consecutive scale spaces created by subsequent Gaussian filtering steps. The number of candidates is reduced by a non-maximum-suppression search within the neighborhood of three adjacent scale spaces. Finally, the exact feature position and scale is determined using the information of adjacent scale spaces. Thus, the feature positions between pixels and feature dimensions are characteristics of blob features.

A well-known feature detector has been proposed by Lowe. He proposed the SIFT [13] algorithm to detect scale invariant interest points. SIFT detects feature candidates by Laplacian of Guassian (LoG) filter responses within different scale spaces. Local extremes are extracted using a non-maximum-suppression search providing unique features. Lowe improved the computational performance of SIFT by approximating the LoG by a difference-of-Gaussian approach. Thus, filtered pyramid layers are reused to approximate the exact LoG result. Further, Lowe proposed an algorithm to determine the dominate orientation of a SIFT feature. This orientation can later be used to determine a feature descriptor for feature matching (see Sect. 3.2). In comparison to FAST the SIFT detector needs by far more computational time due to the several filtering iterations to extract the individual scale spaces.

Bay et al. [1] proposed the SURF detector which is in some aspects related to SIFT. Bay also uses different scale spaces to extract scale invariant features. Feature candidates are determined by the determinant of the Hessian matrix of a Gaussian convolution. However, SURF avoids the expensive Gaussian filtering steps to speed up the computational time for feature detection. Bay et al. apply an integral image in combination with rough approximated box filters of the Gaussian derivatives. Thus, the determinant of the Hessian matrix can be approximated with a few look-up operations using the integral image. Further, the computational time is independent from the filter size and thus is constant over all scale spaces. Cheng et al. [3] proved that SURF outperforms e.g. SIFT in performance and robustness.

Figure 11.1 shows a comparison between the FAST corner detector as described in Sect. 2.1 and the SURF feature detector as described in Sect. 2.2. The figure provides the unique characteristics of those detector classes.

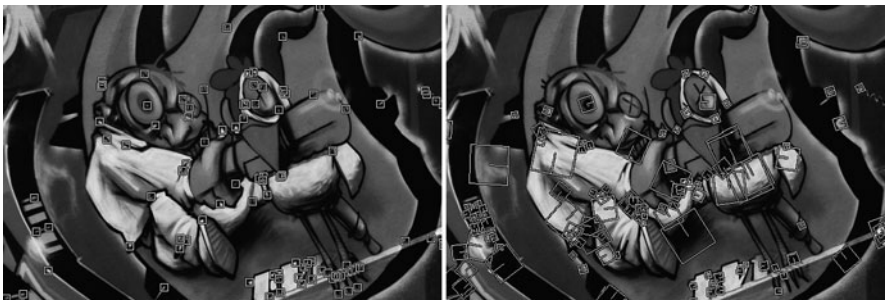

Fig. 11.1 Comparing FAST features without orientation and scale properties (*left*) and SURF features aligned to their dominant orientation (*right*)

3 Feature Matching

Once a set of robust features as been detected, they have to be matched against corresponding features inside a database to extract a respective 6DOF pose. Feature matching is one of the most critical tasks inside the entire tracking pipeline. The high number of potential feature correspondences results in a large computational effort, slowing down the tracking process significantly. Additionally, wrong feature pairs (so called outliers) would result in distorted or even wrong pose calculations and thus have to be handled explicitly. In the following sub-sections three different matching strategies are presented.

3.1 Image Patches

Feature matching by image patches is the most intuitive way to find valid correspondences between features inside an image and a huge database. All detected features are surrounded by a small image patch with constant size (e.g. 8×8 pixels) and compared with all patches inside the database. Often the summed square distance function (SSD) or the sum of absolute distance function (SAD) is used to find best matching patches. Especially the SAD computation performs quite efficiently and thus image patches are regularly applied for feature matching on mobile devices. However, image patches are very error-prone with respect to changing light conditions or viewing angles because they are not scale or rotation invariant. Thus, image patches commonly can only be used if enough a-priori information from e.g. a previous tracking iteration is known. Thus, reference image patches from database features may be replaced by image patches of the most recent camera frame to allow an almost perfect matching for the next camera frame. Further, a transformation based on the previous pose is applied to adjust the database patches to the current viewing angle [9, 10]. However, commonly feature patches

cannot be applied for pose initialization, e.g. because of the absence of a previous pose. Therefore, image patches often are used for feature matching between two consecutive camera frames only.

3.2 Feature Descriptors

Feature descriptors are designed to compensate the drawbacks of image patches. An abstract feature descriptor may hold arbitrary information to describe a feature and to allow the separation of two different features. Feature descriptors should be scale and orientation invariant, should allow for fast calculation and should describe the concerning interest point as unique as possible while needing as less information as necessary.

Several different feature descriptors have been proposed in the last decade. Additionally, many feature detectors have been suggested with own descriptors like e.g. SIFT or SURF. Most feature descriptors are defined by a vector of up to 128 scalar values defined by the direct neighborhood of the corresponding feature point. Feature matching between camera features and database features then can be applied by seeking the best matching descriptor pairs. The best pair may be found by using any distance function suitable for the specific descriptor (see Fig. 11.2). However, often a simple SSD or SDA search is sufficient to find corresponding descriptors.

One has to distinguish between feature descriptors, which are rotation invariant itself, and descriptors aligned to the dominant orientation of the corresponding feature point allowing for an adequate consideration of the current orientation and by that making the descriptor rotation invariant. The latter category is applied more commonly, although the extra time for orientation determination reduces the overall tracking performance considerably.

3.3 Trees and Ferns

Feature matching also can be solved by a classification problem. Instead of creating scale and orientation invariant descriptors, Lepetit et al. [11, 12] have shown that feature correspondences can be determined by classes specifying the characteristics of feature points. Each feature point represents a class holding all possible appearances of this feature point under changing lighting and viewing conditions. The classification is done by a large number of very simple tests like e.g. an image intensity comparison of two different positions in the direct feature neighborhood. Feature matching than can be formulated as a search for the best matching class traversing a tree-like structure. The advantage of this technique is that the dimension or the orientation of the feature render irrelevant for the matching process. Thus, matching can be performed much faster in particular on mobile devices with reduced

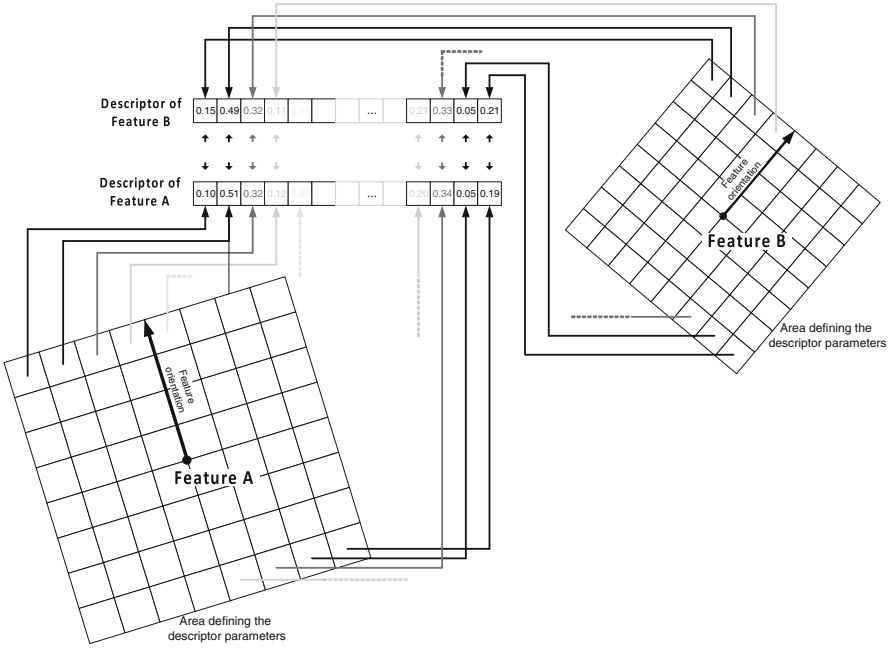

Fig. 11.2 Scheme of the determination of an abstract feature descriptor. The descriptor parameters of two features may be determined as depicted and element wise compared using an arbitrary distance function

computational power. On the other hand this performance speedup has to be paid with an expensive training process of all feature classes in an offline preprocessing step. Further, such classification trees need a huge amount of memory to store all feature classes. Therefore, Özuysal et al. [15] reformulated the problem to a Bayesian classification, grouping the simple tests to several sub classes. They call these groups *ferns* and proved that the memory amount can be reduced significantly while providing comparable classification accuracy as for common trees.

Normally, the feature classes have to be trained with several images of the tracking object from different viewpoints and with different lighting conditions. Although these images can be created automatically, the classification problem leads to significantly more time to setup a tracking system than necessary for trackers applying descriptor matching.

4 Tracking Pipeline

The tracking pipeline covers the entire tracking process starting with camera frame grabbing and ending with the final 6DOF pose. In the following all important pipeline elements are presented in detail necessary to implement a functional feature

tracking pipeline. Without loss of generality the pipeline is explained for SURF features and their descriptors. However, the conceptual details hold for any feature detector and matching technique.

The tracking pipeline consists of some few tasks being applied for each new camera frame. First, reliable and strong SURF features must be detected in each new frame. Afterwards, those features have to be matched against a previously created SURF feature database holding reference features of the object to be tracked. Thus, SURF descriptors must be calculated for all camera features used for matching. If enough correspondences between camera and database features can be found, the associated 6DOF camera pose may be calculated. However, typically some of the found feature correspondences are faulty and thus must not be used for pose calculation to guarantee an accurate pose. Therefore, all correspondence outliers must be exposed during the pose determination. Afterwards, the pose can be refined using additional feature correspondences e.g. those found using the just extracted pose. Finally, the 6DOF pose is forwarded to the underlying system. If the next camera frame is available the tracking pipeline will restart. However, this time information from the previous camera frame – like e.g. the previous pose – can be used to improve tracking performance and accuracy.

4.1 Feature Detection

In each new camera frame new feature points have to be detected to be used for pose determination later. Depending on the applied detector this task can be one of the most time consuming parts inside the tracking pipeline. Obviously, the number of detected interest points is directly related to the frame dimension. The bigger the input frame the more feature points will be detected and vice versa. However, often the number of detected feature points can be reduced by a specific strength threshold all features must achieve to count as strong and reliable features. Thus, e.g. very strong corners will be detected while more homogenous regions or blurred corners are discarded as natural features. For SURF features the determinant of the Hessian matrix can directly be used as strength value and allows the separation of strong and robust features from weak and unreliable ones (see Fig. 11.3).

This strength threshold is an important instrument to balance the overall feature number to be considered in each frame and can be used to control the tracking performance and accuracy. Further, features can be sorted according to their strength parameters to use the most robust features for tracking only.

Commonly, available implementations of feature detectors allow for feature detection over the entire input frame only. However, often a-priori information from previous tracking iterations is given, which can be used to predict a small area of interest inside the input image. The exact area of interest can be determined by a projection of all 3D feature points into the camera coordinate system while using the pose of the previous frame for the extrinsic camera parameters. More efficient is an approximation of this area by projecting the 3D feature bounding box only.

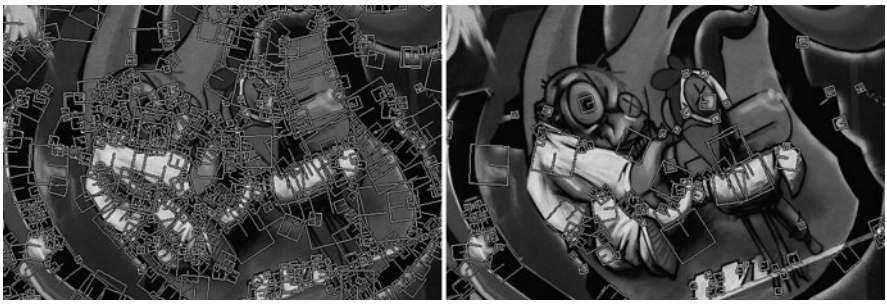

Fig. 11.3 SURF features with different minimal strength detection thresholds; the threshold in the *right* image is four times higher than for the *left* image

If the used implementation supports sub-image feature detection this approximated area of interest can be used to significantly improve the performance of the tracking pipeline. Additionally, the number of invalid interest points can be reduced right from the beginning, speeding up the matching process and reducing the number of matching outliers.

4.2 Feature Map

Apart from SLAM approaches (see Sect. 6), feature tracking needs a predefined set of feature points, which can be used as reference for the real object to be tracked. Thus, the tracking object and its real dimensions should be known in advance. The object then is used to create a corresponding database holding 3D feature points with exact proportions. This database often is referred to as feature map. If feature matching finally is based on descriptors, those have be determined for all features inside the map. Often an image or a bitmap of the tracking object is sufficient to create an appropriate map providing adequate details. Especially, if an image or a poster is used as tracking object, the feature map creation is trivial if the image is also available in digitally. Thus, commonly the creation of a feature map needs approximately the same time as the final tracker for one live camera image. Further, the feature map has to be created only once before tracking starts because the feature map typically does not change while the application is running.

4.3 Camera Distortion

Most camera devices, especially cheap webcams or integrated cameras of mobile phones, provide skewed images due to a weak lens quality resulting in a severe distortion. In contrast to an optimal pinhole camera model, images of real cameras

have to be corrected to handle so called *barrel* or *pincushion* distortion. Especially, with raising distance to the principal point, pixel errors can have a substantial influence on the final pose accuracy. Brown [2] has proposed a simple distortion model able to handle radial and tangential lens distortion. A low degree polynomial is appropriate to correct the distortion and provides sufficient accurate results for feature tracking. Thus, before feature tracking can start the used camera should be calibrated and the distortion parameters should be extracted explicitly. Several different calibration methods have been proposed in the last decade needing different types of calibration patterns. An implementation of a calibration method using a simple planar calibration pattern is part of the OpenCV library.

All feature positions should be corrected according to the distortion parameters before using them for pose determination to avoid accuracy errors.

4.4 Feature Description

Once stable features have been detected, SURF descriptors must be calculated for all features actually used for feature matching. Depending on information from previous tracking iterations, feature detectors may be calculated for a subset of the detected features only. Feature description is beside feature detection the most time consuming task in the tracking pipeline. The more SURF descriptors are calculated the more time is necessary for each new frame to proceed. However, obviously feature correspondences can only be found between features providing a calculated descriptor and thus the number of features to be descripted should be selected carefully.

Tracking systems not using descriptors for feature matching but using ferns instead, can avoid this description task and can directly start searching for correspondences. Therefore, tracking systems not applying descriptors but ferns may perform faster because they shift the expensive descriptor calculations to the training phase of the feature classes before the tracking starts.

4.5 Feature Matching

To extract valid matches between camera and map features, the feature descriptors are used to find feature pairs with nearby similar appearance. SURF features can be matched finding the two SURF descriptors with e.g. smallest summed square distance. Further, the SSD should lie below a specified threshold to ensure a high matching quality with less correspondence outliers. The threshold has to be defined carefully for not rejecting valid feature correspondences on the one hand, while accepting too many wrong feature pairs on the other hand. Each type of feature detector will need its individual threshold to achieve an adequate ratio between accepted and rejected feature correspondences.

If no information can be used to reduce the number of potential feature candidates – e.g. from previous tracking iterations – an initial mapping can be extracted by a brute-force search. Obviously, a brute-force search between n camera features and m map features would need time $O(nm)$ and thus should be applied as rare as possible.

Often a kd-tree is used to speed-up the search for the best matching descriptors. Further, some of the descriptors presented allow for a significant faster matching compared to other descriptor types. All SURF features e.g. can uniquely be categorized into two different kinds of blob-like features. Dark features with brighter environment or bright features with darker environment. Thus, a huge amount of potential correspondence candidates can be discarded very early if two SURF features belong to different categories. In the following more complex culling methods are presented using geometrical information to cull possible correspondence candidates.

4.6 Feature Culling

If the feature tracker can rely on information from previous tracking iterations like e.g. the pose calculated for the previous camera frame, the number of wrong feature correspondences can significantly be reduced by applying culling approaches based on geometrical constraints. We can expect only minor changes in the camera pose for two consecutive frames. Thus, the previous pose can e.g. be used to predict the 2D camera position of all 3D features defined in the feature map.

One efficient culling method is the usage of a culling cone. For each 2D image feature the cone allows to discard all invalid 3D object features upon their 3D position using the previous pose as reference. The implicit form of a cone with apex lying in (m_x, m_y, m_z), having a dihedral angle α and defined along the z-axis is given by the following quadric:

$$F(x,y,z,) = (x-m_x)^2 + (y-m_y)^2 - tan^2\alpha(z-m_z)^2$$

A point $p = (x,y,z)$ is inside the cone if $F(p) < \mathbf{0}$, outside if $F(p) > \mathbf{0}$, and lies on cone surface if $F(p) = \mathbf{0}$. The implicit form of the cone can be expressed in matrix notation:

$$\begin{aligned} F(p) &= p^T \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & -m_x \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & -m_y \\ \mathbf{0} & \mathbf{0} & -tan^{\mathbf{2}}\alpha & m_z tan^{\mathbf{2}}\alpha \\ -m_x & -m_y & m_z tan^{\mathbf{2}}\alpha & m_x^{\mathbf{2}} + m_z^{\mathbf{2}} - m_z^{\mathbf{2}} tan^{\mathbf{2}}\alpha \end{bmatrix} p \\ &= p^T Q p \end{aligned}$$

The cone Q can be transformed by a homogenous transformation matrix M into a new cone Q':

$$\begin{aligned} F_M(p) &= (M^{-1}p)^T Q(M^{-1}p) \\ &= p^T(M^{-T}QM^{-1})p \\ &= p^T Q' p \end{aligned}$$

Thus, only a single vector-matrix multiplication followed by a vector scalar product is sufficient to determine whether a 3D point lies outside an arbitrary culling cone. The amount of culled 3D features is defined by the cone's dihedral. The angle has to be carefully chosen to handle the difference between the previous pose and the new pose to be determined for the current frame. The smaller the angle the smaller the allowed difference between the two poses and the more 3D features will be discarded.

4.7 Pose Calculation and Pose Refinement

All extracted feature correspondences will be used for determination of the final pose related to the current camera frame. Pose determination from a set of n feature correspondences is known as the *perspective n-point problem (PnP)*. At least three valid feature correspondences between 2D image features and 3D map features are necessary to proceed. The so called *P3P* [5] provides at most four different poses due to geometric ambiguities. Thus, further feature correspondences are necessary to determine the unique pose. Several approaches have been proposed to solve the *P4P*, *P5P* or even *PnP* [12]. However, the more feature correspondences are known the more accurate the extracted pose. Further, commonly a set of detected feature correspondences contains invalid feature pairs due to matching errors leading to inaccurate or even invalid pose calculations. Thus, invalid features correspondences must be exposed explicitly. The well-known RANSAC [5] algorithm can be used to determine the subset of valid correspondences from a large set of feature pairs containing outliers. RANSAC randomly chooses few feature pairs to create a rough pose using e.g. the *P3P* algorithm and tests this pose geometrically for all remaining pairs. Several iterations are applied using different permutations of feature pairs. This results in the pose where most feature correspondences could be confirmed.

However, RANSAC does not optimize the final pose for all valid feature correspondences. Therefore, the pose should be refined further to reduce the overall error regarding all applied feature correspondences. To optimize the pose with respect to the total projection error, nonlinear models must be applied. Commonly, several iterations of nonlinear least squares algorithms like e.g. the Levenberg-Marquardt (LM) algorithm [14] are sufficient to converge to the final pose. Further, different feature correspondences may be weighted e.g. according to their expected position accuracy to increase the pose quality. Often, an M-estimator [14] is applied automatically neglecting the impact of outliers or inaccurate feature positions.

5 Overall System of an Adaptive Tracker

A naive implementation of a feature tracker would apply the above presented elements of a tracking pipeline simply in a sequential manner. However, the entire tracking pipeline provides space for potential optimizations. As indicated above, a feature tracker may reuse information from previous camera frames to improve tracking performance and accuracy significantly. Further, the tracker should be able to choose the number of necessary features for each new frame in advance to allow a fast tracking iteration, while being as accurate as possible. In the following a SURF tracker adapting to the environment, the current lighting, and the occlusion conditions is presented in detail using the base elements of the above described tracking pipeline as proposed in our previous work [7].

The adaptive tracker provides three different process paths, which are selected depending on the availability of information received from previous tracking iterations. Figure 11.4 shows the overall scheme of the adaptive tracker and its individual paths and the transitions between those paths. The adaptive tracker starts with an initial number of n features to be used and a lower and upper boundary for n.

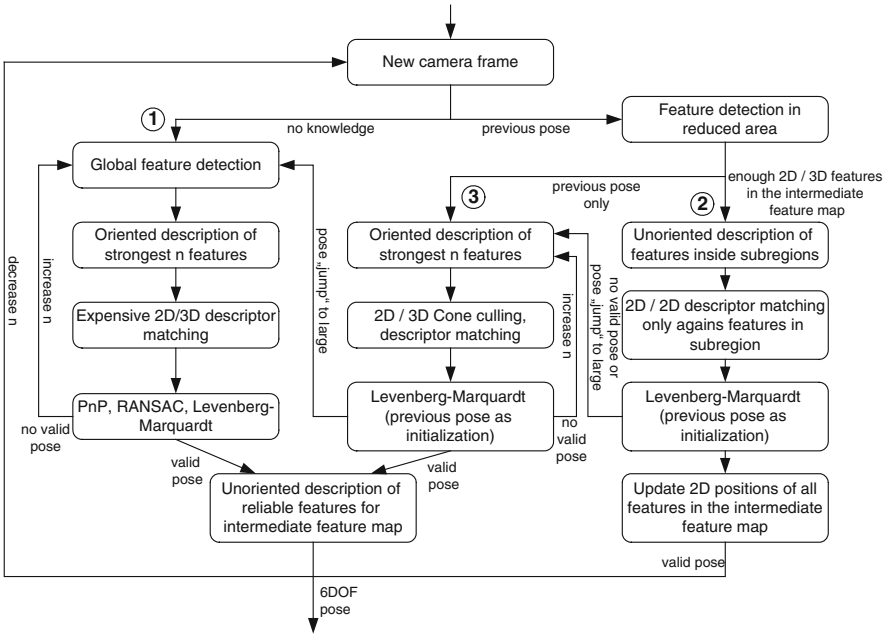

Fig. 11.4 Scheme of an implementation for an adaptive feature tracker

5.1 Tracking without Previous Knowledge

The first path is chosen if the tracker starts for the first time or the tracking has been interrupted in the previous frame. This path needs the most computational effort and therefore is used as rare as possible. The SURF features are detected in the entire image frame and sorted according to their strength (as a result of the hessian determinant) and for the first n strongest features the oriented SURF descriptors are calculated. Afterwards these n features are matched against all features inside the reference feature map of the object to be tracked. Only unidirectional feature correspondences are accepted with a SSD bellow a defined threshold. Additionally, the ratio between best and second best SSD matching must be below a specified ratio e.g. 0.6 to guarantee strong correspondences. Then, the 2D/3D feature correspondences detected are taken to determine the exact camera 6-DOF pose by applying several Perspective-n-Point (PnP) iterations, followed by a pose refinement step with a non-linear-least-square algorithm i.e. the Levenberg-Marquardt algorithm. If a sufficient number of feature correspondences have been detected and validated by the RANSAC iterations, the number of used features n will be decreased as long as it is higher than the lower boundary and the determined 6-DOF pose is returned. Afterwards, the algorithm waits for the next camera frame. If the pose estimation fails, the number of used (oriented and described) features n will be increased and again (better) feature correspondences will be estimated. If n reaches the upper boundary, the object to be tracked is expected to be invisible in the current frame and thus the tracker returns no 6-DOF pose.

5.2 Using Existing Feature Correspondences

The algorithm uses the second path, if a sufficient number of feature correspondences are available from the previous camera frame. Thus, the intermediate feature map contains 2D image features with un-oriented descriptors and their corresponding 3D object points. This enables the tracker to apply a significantly more efficient 2D/2D tracking. First, SURF features are detected inside the smallest area enclosing all previous 2D features with a small extra boundary. Then, the un-oriented descriptors are calculated for all features lying inside small sub regions of each 2D feature from the intermediate map. The size of the sub regions depends on the camera resolution and lies between 30 and 60 pixels. Afterwards, the 2D/2D mapping is applied to features lying inside the same sub region only. This allows for significantly reducing the number of false correspondences, while increasing the overall performance. If an insufficient number of feature correspondences can be recovered or the difference or gap between the previous pose and the new pose is too large, the algorithm proceeds with the third path (see below). However, if a sufficient number of correspondences can be recovered, the exact 6-DOF pose is determined by a Levenberg-Marquardt iteration using the previous pose as initial guess. Afterwards, the intermediate map is updated by the current un-oriented

descriptors and the new 2D positions of all successfully recovered correspondences. The remaining features in the intermediate map (those with invalid correspondences in the recent camera frame) receive an updated 2D position by the projection of the known 3D object position and the 6-DOF pose just extracted. Thus, those features are preserved for further usage in forthcoming frames. However, if a feature has not been recovered for several frames, it is removed from the intermediate map. Finally, the 6-DOF pose is returned and the tracker waits for the next frame.

5.3 Using the Previous Pose as Basis

The third path is used if no or an insufficient number of reliable 2D/3D correspondences are available from the previous frame. In this situation the intermediate map is empty and the tracker can use the pose of the previous tracking iteration only. However, a pose from the previous frame still provides a pretty good guess for the current pose, providing a significant advantage compared to a calculation of a pose without any prior knowledge as used by path one. First, the previous pose is used to project the bounding volume of the entire 3D feature map into the current camera image. This bounding box (with a small extra boundary) defines the region of interest. Features useful for tracking may only occur in this region. The reduced 2D tracking area speeds up the detection process and avoids unnecessary descriptor comparison of features not part of the stored object feature map. The feature detection is followed by a feature sorting according to their individual strength. The oriented SURF descriptors are then determined for the strongest n features.

After all geometrically invalid candidates for 3D features have been eliminated feature matching can be applied between the n strongest 2D image features and those having passed the specific *cone culling*. If an insufficient number of correspondences can be determined, n will be increased and the third path restarts with additional features to be matched. On the other side if enough correspondences can be extracted, a Levenberg-Marquardt iteration uses the previous pose as initial guess and uses the new correspondences for the accurate pose determination. Afterwards, the intermediate feature map will be filled with the new reliable correspondences and their un-oriented descriptors (similar to the first path) and n will be decreased. The extracted 6-DOF pose is returned and the tracker starts with a new camera frame.

6 SLAM

Simultaneous localization and mapping (SLAM) approaches explicitly avoid the application of feature maps created before the actual start of the tracking. Rather than using a feature map holding reference features for 6DOF pose determination,

SLAM techniques create and extend their own feature map during the tracking process ad-hoc. Thus, a SLAM based tracking algorithm starts without any previous knowledge of the current environments, gathering all information required – such as the positions of 3D feature points – during the actual tracking process. Commonly, the initialization takes only a few seconds for creating a very rough initial feature map, allowing for an initial pose calculation. Afterwards, this initial feature map is enlarged and improved with new feature points detected in subsequent camera frames.

Originally, SLAM techniques were proposed for autonomous robots for independent exploration of an unknown environment [4]. Typically, robots have access to further sensor input such as odometry or gyro compass data. In contrast to this, handheld single camera Augmented Reality applications completely rely on visual input information. Within the AR context SLAM approaches are also known as Parallel Tracking and Mapping (PTAM) techniques. A well-known PTAM approach has been proposed by Klein et al. [9]. They initialize their tracking system by a stereo image of a small tracking area using a single handheld camera only. Klein et al. apply FAST features in combination with simple image patches for feature tracking. The size of the feature map increases with each new frame and the positions of already found 3D feature points are improved using a global optimization approach. During initialization all found feature points are used to determine the dominant 3D plane. This plane is then used as basis for the virtual objects in the AR application. Due to the high performance when calculating FAST features, PTAM even can be applied on mobile phones as shown by Klein et al. [10]. On the one hand, SLAM approaches do not need any previously created feature maps, on the other hand, the stereo image initialization with a single camera does not allow for an exact definition of the dimensions of the environment. In contrast to this, a unique feature map created previously allows for augmenting the real environment with virtual objects with precise object dimensions.

7 Conclusions and Future Directions

In this chapter we reviewed the current state-of-the-art in feature based tracking for Augmented Reality (Fig. 11.5), providing the reader with the information necessary to decide on the appropriate components of his tracking approach. We investigated into general requirements of feature detectors for real-time capable vision based tracking with a particular focus on corner detectors, such as the Harris or FAST feature detectors, and blob detectors, such as SIFT or SURF detectors. Once, the features detectors have been terminated, matching them against corresponding features for extracting the 6DOF pose is required. We reviewed the three basic approaches: use of image patches, use of feature descriptors and how best matching pairs may be found, and finally the use of trees and ferns. In order to show the functionality of a feature based tracker the individual stages of the tracking pipeline were explained in detail: feature detection, feature mapping, dealing with camera

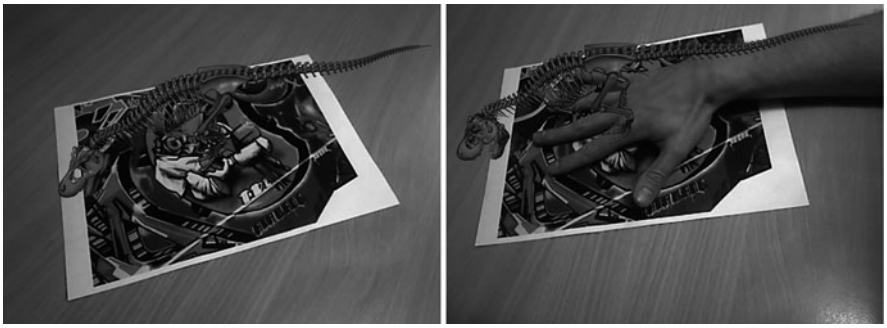

Fig. 11.5 Feature tracking result of the adaptive feature tracker augmenting a virtual dinosaur

distortion, feature description, feature matching, fast feature culling, and finally the calculation of the pose and/or its refinement. We further showed how an adaptive tracker can be realized. The three different tracking approaches applied here were discussed: Firstly, the tracking without any previous knowledge completely based on the information of the current frame. Secondly, tracking using existing feature correspondences from the previous frame, and finally, an approach using the previous pose as a basis for tracking. While the approach presented, relied on SURF feature descriptors, the general architecture may also be applied to other feature descriptors, or may easily be extended to combine several feature descriptors and/or matching approaches.

While selected current approaches may already be used on recent mobile phones and tablets, it can be expected that those mechanisms will become widely used upon availability of multi-core mobile phones and tablets. Further, we expect online 2D and 3D data to be used for this purpose in combination with SLAM like approaches, relying on ad-hoc information from the environment. Combining those will provide a basis for universal mobile tracking.

References

1. H. Bay, A. Ess, T. Tuytelaars, and L. van Gool, "Speeded-up robust Features (SURF)," Computer Vision and Image Understanding (CVIU), Volume 110, No. 3, pages 346–359, 2008
2. D. C. Brown, "Close-range camera calibration," Photogrammetric Engineering, Volume 37, Number 8, pages 855–866, 1971
3. D. Cheng, S. Xie, and E. Hämmerle, "Comparison of local descriptors for image registration of geometrically-complex 3D scenes," 14th International Conference on Mechatronics and Machine Vision in Patrice (M2VIP), pages 140–145, 2007
4. A. J. Davison, and N. Kita, "3D simultaneous localisation and map-building using active vision for a robot moving on undulating terrain," In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Kauai, 2001

5. M. A. Fischler, and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” Communications of the ACM 24, Volume 6, 1981
6. C. Harris, and M. Stephens, “A Combined Corner and Edge Detector,” Proceedings of the 4th Alvey Vision Conference, pages 147–151, 1988
7. J. Herling and W. Broll, “An Adaptive Training Free Tracker for Mobile Phones”, In Proc. of the 17th ACM Conference on Virtual Reality Systems and Technology (VRST 2010), ACM, New York, NY, USA, pages 35–42
8. H. Kato, and M. Billinghurst, “Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System”, *in 2nd IEEE and ACM International Workshop on Augmented Reality*, 1999
9. G. Klein, and D. Murray, “Parallel Tracking and Mapping for Small AR Workspaces”, Proc. of IEEE *ISMAR*, 2007
10. G. Klein, and D. Murray, “Parallel Tracking and Mapping on a Camera Phone”, Proc. of IEEE *ISMAR*, 2009
11. V. Lepetit, P. Lagger, and P. Fua, “Randomized Trees for Real-Time Keypoint Recognition,” Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), pages 775–781, 2005
12. V. Lepetit, F. Moreno-Noguer, and P. Fua, “EPnP: An Accurate O(n) Solution to the PnP Problem,” International Journal of Computer Vision, Volume 81, Issue 2, 2009
13. D. G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” International Journal of Computer Vision. Volume 60 Issue 2, 2004
14. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B.P. Flannery, “Numerical Recipes: The Art of Scientific Computing,” Cambridge University Press, Third Edition, 2007
15. M. Özuysal, P. Fua, and V. Lepetit, “Fast Keypoint Recognition in Ten Lines of Code,” Proceedings of IEEE Conference on Computing Vision and Pattern Recognition, pages 1–8, 2007
16. E. Rosten, and T. Drummond, “Fusing Points and Lines for High Performance Tracking,” Proceedings of the IEEE International Conference on Computer Vision, pages 1508–1511, 2005