Foundations and Trends® in
Computer Graphics and Vision
Vol. 1, No 1 (2005) 1–89

now
the essence of knowledge

Monocular Model-Based 3D Tracking of Rigid Objects: A Survey

Vincent Lepetit[1] and Pascal Fua[2]

[1] *Computer Vision Laboratory, CH-1015 Lausanne, Switzerland, Vincent.Lepetit@epfl.ch*
[2] *Computer Vision Laboratory, CH-1015 Lausanne, Switzerland, Pascal.Fua@epfl.ch*

Abstract

Many applications require tracking of complex 3D objects. These include visual servoing of robotic arms on specific target objects, Augmented Reality systems that require real-time registration of the object to be augmented, and head tracking systems that sophisticated interfaces can use. Computer Vision offers solutions that are cheap, practical and non-invasive.

This survey reviews the different techniques and approaches that have been developed by industry and research. First, important mathematical tools are introduced: Camera representation, robust estimation and uncertainty estimation. Then a comprehensive study is given of the numerous approaches developed by the Augmented Reality and Robotics communities, beginning with those that are based on point or planar fiducial marks and moving on to those that avoid the need to engineer the environment by relying on natural features such as edges, texture or interest. Recent advances that avoid manual initialization and failures due to fast motion are also presented. The survery concludes with the different possible choices that should be made when

implementing a 3D tracking system and a discussion of the future of vision-based 3D tracking.

Because it encompasses many computer vision techniques from low-level vision to 3D geometry and includes a comprehensive study of the massive literature on the subject, this survey should be the handbook of the student, the researcher, or the engineer who wants to implement a 3D tracking system.

1

Introduction

Tracking an object in a video sequence means continuously identifying its location when either the object or the camera are moving. There are a variety of approaches, depending on the type of object, the degrees of freedom of the object and the camera, and the target application.

2D tracking typically aims at following the image projection of objects or parts of objects whose 3D displacement results in a motion that can be modeled as a 2D transformation. An adaptive model is then required to handle appearance changes due to perspective effects or to deformation. It can provide the object's image position in terms of its centroid and scale or of an affine transformation [141, 26, 62]. Alternatively, more sophisticated models such as splines [16], deformable templates [142], 2D deformable meshes [112], or 2D articulated models [20] can be used. However, none of these methods involves recovering the actual position in space.

By contrast, 3D tracking aims at continuously recovering all six degrees of freedom that define the camera position and orientation relative to the scene, or, equivalently, the 3D displacement of an object relative to the camera.

1.1 Focus and Organization of the Survey

The 3D tracking literature is particularly massive both because there are almost as many approaches as target applications and because many different approaches to solving the same problem are possible. Here we focus on online model-based 3D tracking using a single camera. We will describe both marker-based techniques and marker-less natural features-based approaches for camera, scene, and object 3D tracking.

In particular, we will not consider batch methods to camera trajectory recovery: Because these methods can consider image sequence as a whole, they can rely on non-causal strategies that are not appropriate for online camera tracking. Furthermore, an excellent reference text for this topic already exists [54]. We will restrict ourselves to single-camera approaches because multi-camera systems require calibration of the stereo-rig and are therefore less popular. We will consider only rigid objects or scenes, as opposed to deformable [25, 89] or articulated objects such as human bodies [43, 121] that would take us too far afield.

We will first introduce the key mathematical tools required for 3D tracking. We will then present marker-based techniques that use either point fiducials or planar markers to ease the tracking task. Next we will focus on techniques that rely on natural features. Finally, we will discuss recent advances that seek to increase tracking robustness to disappearance and reappearance of the target object by replacing frame-to-frame tracking by detection in each frame individually.

1.2 Different Approaches for Different Applications

3D tracking is a very useful tool in many different fields, and we briefly review some of them below.

1.2.1 Augmented Reality Applications

Many potential Augmented Reality (AR) applications have been explored, such as medical visualization, maintenance and repair, annotation, entertainment, aircraft navigation and targeting. They all involve superposing computer generated images on real scenes, which must be done at frame-rate in online systems. 3D real-time tracking is

therefore a critical component of most AR applications. The objects in the real and virtual worlds must be properly aligned with respect to each other, and the system latency should also be low, lest the illusion that the two worlds coexist be compromised.

1.2.2 Visual Servoing

Visual servoing involves the use of one or more cameras and a Computer Vision system to control the position of a device such as a robotic arm relative to a part it has to manipulate, which requires detecting, tracking, servoing, and grasping. It therefore spans computer vision, robotics, kinematics, dynamics, control and real-time systems, and is used in a rich variety of applications such as lane tracking for cars, navigation for mobile platforms, and generic object manipulation.

The tracking information is required to measure the error between the current location of the robot and its reference or desired location from eye-in-hand cameras. As a consequence, the tracking algorithm must be robust, accurate, fast, and general.

1.2.3 Man–Machine Interfaces

3D tracking can be integrated into man–machine interfaces. For example, it could be used to continuously update the position of a hand-held object, which would then serve as a 3D pointer. This object would then become an instance of what is known as a *Tangible Interface*. Such interfaces aim at replacing traditional ones by allowing users to express their wishes by manipulating familiar objects and, thus, to take advantage of their everyday experience.

Eventually, this is expected to lead to more natural and intuitive interfaces. In this context, vision-based tracking is the appropriate technique for seamless interaction with physical objects.

1.3 Computer Vision-Based 3D Tracking

Many other technologies besides vision have been tried to achieve 3D tracking, but they all have their weaknesses: Mechanical trackers are accurate enough, although they tether the user to a limited working

volume. Magnetic trackers are vulnerable to distortions by metal in the environment, which are a common occurrence, and also limit the range of displacements. Ultrasonic trackers suffer from noise and tend to be inaccurate at long ranges because of variations in the ambient temperature. Inertial trackers drift with time.

By contrast, vision has the potential to yield non-invasive, accurate and low-cost solutions to this problem, provided that one is willing to invest the effort required to develop sufficiently robust algorithms. In some cases, it is acceptable to add fiducials, such as LEDs or special markers, to the scene or target object to ease the registration task, as will be discussed in Section 3. Of course, this assumes that one or more fiducials are visible at all times. Otherwise, the registration falls apart. Moreover, it is not always possible to place fiducials. For example, Augmented Reality end-users do not like them because they are visible in the scene and it is not always possible to modify the environment before the application has to run.

It is therefore much more desirable to rely on naturally present features, such as edges, corners, or texture. Of course, this makes tracking far more difficult: Finding and following feature points or edges can be difficult because there are too few of them on many typical objects. Total, or even partial occlusion of the tracked objects typically results in tracking failure. The camera can easily move too fast so that the images are motion blurred; the lighting during a shot can change significantly; reflections and specularities may confuse the tracker. Even more importantly, an object may drastically change its aspect very quickly due to displacement. For example this happens when a camera films a building and goes around the corner, causing one wall to disappear and a new one to appear. In such cases, the features to be followed always change and the tracker must deal with features coming in and out of the picture. Sections 4 and 5 focus on solutions to these difficult problems.

2

Mathematical Tools

This section introduces the mathematical tools commonly used for 3D tracking purposes. They are described in similar terms in many papers and we felt it more effective to present them all here. We begin with camera representation and pose parameterization. In particular, camera orientation can be formulated as a rotation in 3D space and we discuss its many possible representations. We then turn to the numerical optimization techniques required to estimate the pose from image data and discuss ways to increase their robustness when the data is noisy. Finally, we introduce filtering tools that enforce motion models and let us combine mutiple cues and estimate probability distributions over the space of poses.

2.1 Camera Representation

Here we focus on the standard pinhole camera model, including potential deviations from it, which is appropriate for most cameras currently used for tracking purposes. Note, however, that new camera designs, such as the so-called omnidirectional cameras that rely on hyperbolic or parabolic mirrors to achieve very wide field of views,

are becoming increasingly popular. The complete treatment of such cameras is beyond the scope of this survey. Nevertheless, they are amenable to mathematical treatment very similar to the one presented below [46, 126].

2.1.1 The Perspective Projection Model

Photogrammetrists have extensively researched the estimation of camera poses and more generally camera parameters from images well before Computer Vision scientists. In this survey, we focus on combining such estimation techniques with automated extraction of image features for 3D tracking rather than estimation itself. We therefore provide here only what must be undestood to carry out this integration. For additional details about camera models and an in-depth study of the numerical and geometric properties of camera calibration, we refer the interested reader to the photogrammetric texts [48, 88].

Mathematically, image formation can be defined as the projection from the 3D space to the image plane depicted by Figure 2.1. The coordinates of a 3D point $\mathbf{M} = [X, Y, Z]^T$ expressed in a Euclidean

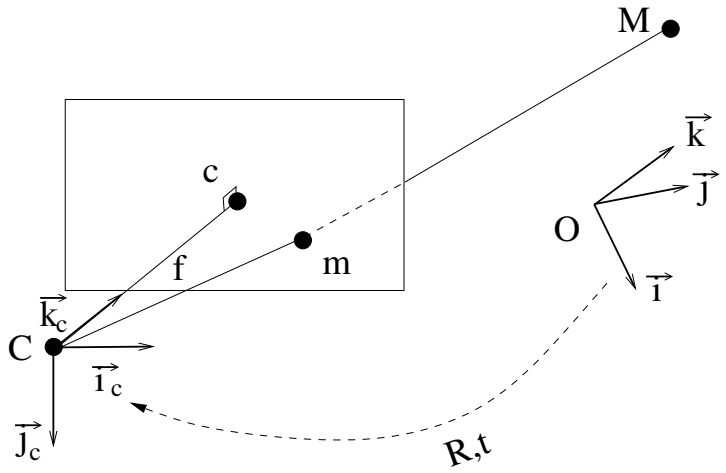


Fig. 2.1 The perspective projection model. $(\mathbf{O}, \vec{i}, \vec{j}, \vec{k})$ is the world coordinate system, $(\mathbf{C}, \vec{i_c}, \vec{j_c}, \vec{k_c})$ is the camera coordinate system, $\mathbf{M}$ is a 3D point, and $\mathbf{m}$ is its projection onto the image plane.

world coordinate system and the corresponding 2D point $\mathbf{m} = [u, v]^T$ in the image are related by the equation

$$s\tilde{\mathbf{m}} = \mathbf{P}\tilde{\mathbf{M}}, \tag{2.1}$$

where s is a scale factor, $\tilde{\mathbf{m}} = [u, v, 1]^T$ and $\tilde{\mathbf{M}} = [X, Y, Z, 1]^T$ are the homogeneous coordinates of points $\mathbf{m}$ and $\mathbf{M}$, and $\mathbf{P}$ is a 3 $\times$4 projection matrix. $\mathbf{P}$ is defined up to a scale factor, and thus depends on 11 parameters. It is usually taken to be a perspective projection matrix because it realistically describes the behavior of a reasonably good quality camera, while remaining relatively simple. Such perspective projection matrix can be decomposed as:

$$\mathbf{P} = \mathbf{K}\,[\mathbf{R} \mid \mathbf{t}]$$

where:

- $\mathbf{K}$ is the 3×3 camera calibration matrix that depends of the internal parameters of the camera such as the focal length;
- $[\mathbf{R} \mid \mathbf{t}]$ is the 3×4 external parameters matrix, and corresponds to the Euclidean transformation from a world coordinate system to the camera coordinate system: $\mathbf{R}$ represents a 3×3 rotation matrix, and $\mathbf{t}$ a translation. We describe both in more detail below.

2.1.2 The Camera Calibration Matrix

The $\mathbf{K}$ camera calibration matrix contains the intrinsic camera parameters, also called internal parameters. We write

$$\mathbf{K} = \begin{bmatrix} \alpha_u & s & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.2}$$

where

- α_u and α_v are respectively the scale factor in the u- and v- coordinate directions. They are proportional to the focal length f of the camera: $\alpha_u = k_u f$ and $\alpha_v = k_v f$, where k_u and k_v are the number of pixels per unit distance in the u and v directions;

- $\mathbf{c} = [u_0, v_0]^T$ represents the image coordinates of the intersection of the optical axis and the image plane, also called the principal point;
- s, referred as the skew, is non-zero only if the u and v directions are not perpendicular, which is exceedingly rare in modern cameras.

Taking the principal point $\mathbf{c}$ to be at the image center often is a very reasonable approximation. Similarly, if the pixels are assumed to be square, α_u and α_v can be taken to be equal. From now on, a camera will be said to be calibrated when its internal parameters are known.

2.1.3 The External Parameters Matrix

The 3×4 external parameters $[\mathbf{R} \mid \mathbf{t}]$ matrix defines the orientation and the position of the camera. It is formed of a $\mathbf{R}$ rotation matrix and a $\mathbf{t}$ translation vector and we will often refer to it as the *camera pose*. Tracking applications usually assume that the $\mathbf{K}$ calibration matrix is known and focus on estimating $\mathbf{R}$ and $\mathbf{t}$, or, equivalently, the target object's position and orientation with respect to the camera.

More formally, it corresponds to the Euclidean transformation from a world coordinate system to the camera coordinate system: A 3D point represented by the vector $\mathbf{M}_w$ in the world coordinate system will be represented by the vector $\mathbf{M}_c = \mathbf{R}\mathbf{M}_w + \mathbf{t}$ in the camera coordinate system. From this relation, we can easily recover the expression of the *camera center*, or *optical center* $\mathbf{C}$ in the world coordinate system. It must satisfy $\mathbf{0} = \mathbf{R}\mathbf{C} + \mathbf{t}$, which implies $\mathbf{C} = -\mathbf{R}^{-1}\mathbf{t} = -\mathbf{R}^T\mathbf{t}$.

2.1.4 Estimating the Camera Calibration Matrix

In most 3D tracking methods, the internal parameters are assumed to be fixed and known, which means that the camera cannot zoom, because it is difficult to distinguish a change in focal length from a translation along the camera Z-axis. These parameters can be estimated during an offline camera calibration stage, from the images themselves. A discussion of camera calibration techniques is beyond the scope of this survey. Let us simply say that classical calibration

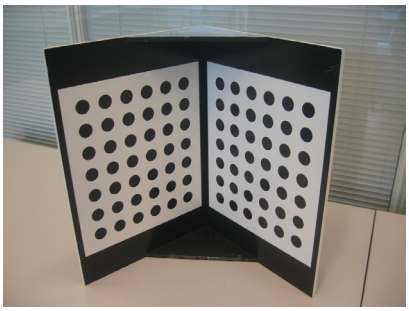


Fig. 2.2 A 3D calibration grid used for the estimation of the camera calibration matrix.

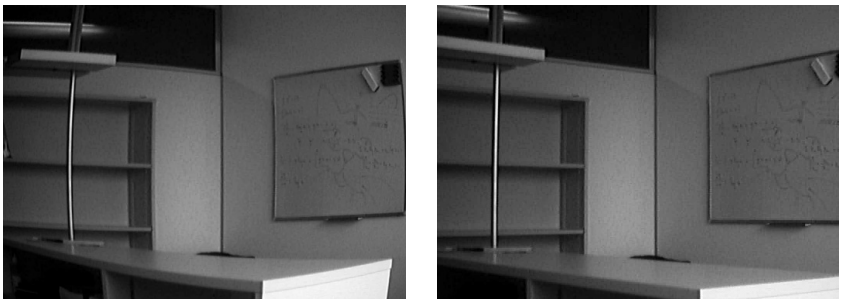
methods make use of a calibration pattern of known size inside the field of view. Sometimes it is a 3D calibration grid on which regular patterns are painted [133, 37], such as the black disks of Figure 2.2.

In this example, the 3D coordinates of the corners of the white squares with respect to the grid's upper left corner are exactly known. It is relatively easy to find those corners in the image and, from the correspondence between the 3D points and the 2D image points, to compute projection matrix parameters.

[144, 124] simultaneously introduced similar calibration methods that rely on a simple planar grid seen from several positions. They are more flexible since the pattern can be simply printed, attached to a planar object, and moved in front of the camera. A Windows-based tool can be found at [143], and another implementation is provided in [60].

2.1.5 Handling Lens Distortion

The perspective projection model is not always sufficient to represent all the aspects of the image formation since it does not take into account the possible distortion from the camera lens, which may be non negligible, especially for wide angle cameras. Fortunately, the lens distortion can be modeled as a 2D deformation of the image. As shown in Figure 2.3, given an estimate of the distortion parameters, the distortion effect can be efficiently removed from the image captured by the camera at run-time using a look-up table, and the perspective projection



(a) (b)

Fig. 2.3 Undistorting an image. (a) Due to perspective distortion, projections of straight lines are curved. (b) In the undistorted image, the projections of straight lines are now straight.

model can then be applied. Among other things, this allows the use of wide angle cameras for tracking purposes, which can be beneficial since they make it easier to keep target objects within the field of view.

A common representation of the distortion parameters is as follow. Let $\breve{\mathbf{u}} = [\breve{u}, \breve{v}]^T$ be the observed, distorted pixel image coordinates, and $\breve{\mathbf{x}} = [\breve{x}, \breve{y}]^T$ the corresponding normalized coordinates such that $\breve{u} = u_0 + \alpha_u \breve{x}$ and $\breve{v} = v_0 + \alpha_u \breve{y}$, where u_0, v_0, α_u, and α_v are the intrinsic parameters of Equation (2.2). Let $\mathbf{u} = (u, v)$ and $\mathbf{x} = (x, y)$ be the corresponding undistorted values. The distortion is expressed as the sum of two components such that $\breve{\mathbf{x}} = \mathbf{x} + \mathbf{dx}_{\text{radial}} + \mathbf{dx}_{\text{tangential}}$. The radial distortion that can be approximated as

$$\mathbf{dx}_{\text{radial}} = (1 + k_1 r^2 + k_2 r^4 + \cdots)\mathbf{x},$$

where $r = \|\mathbf{x}\| = \sqrt{x^2 + y^2}$. The tangential distortion $\mathbf{dx}_{\text{tangential}}$ has much less influence. It can be expressed as

$$\mathbf{dx}_{\text{tangential}} = \begin{bmatrix} 2p_1 xy + p_2(r^2 + 2x^2) \\ p_1(r^2 + 2y^2) + 2p_2 xy \end{bmatrix},$$

but is usually ignored.

The software package of [60] allows the estimation of the distortion parameters using a method derived from [55]. This is a convenient

method for desktop systems. For larger workspaces, plumb line based methods [17, 42] are common in photogrammetry. Without distortion, the image of a straight line will be a straight line, and reciprocally the distortion parameters can be estimated from images of straight lines by measuring their deviations from straightness. This is a very practical method in man-made environments where straight lines, such as those found at building corners, are common.

2.2 Camera Pose Parameterization

For estimation and numerical optimization purposes, the camera pose must be appropriately parameterized. While representing translations poses no problem, parameterizing rotation in $\mathbb{R}^3$ is more difficult to do well.

It is well known that a rotation in $\mathbb{R}^3$ has only three degrees of freedom, and it would be awkward to directly use the nine elements of the 3×3 rotation matrix that represents it as parameters. Six additional non-linear constraints – three to force all three columns to be of unit length, and three to keep them mutually orthogonal – would have be added to ensure that the matrix is orthonormal.

We now briefly review the different parameterizations that have proved to be effective for 3D tracking: Euler angles, quaternions, and exponential maps. All three parameter representations have singularities, but these can be avoided by locally reparametrizing the rotation. Nevertheless, we will argue that, in general, the last one has the best properties for our purposes. However, if one restricts oneself to small rotations, they are all equivalent because they yield the same first order approximation. A more thorough comparison can be found in [47].

2.2.1 Euler Angles

A rotation matrix $\mathbf{R}$ can always be written as the product of three matrices representing rotations around the X, Y, and Z axes. There are several conventions on the order in which these rotations are carried

out. For example, taking α, β, γ to be rotation angles around the Z, Y, and X axis respectively yields

$$\mathbf{R} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix}.$$

The inverse operation, extracting the Euler angles for a given rotation matrix can be easily performed by identifying the matrix coefficients and their analytical expression.

Even though Euler angles can do a creditable job for a large range of camera orientations, they have a well known drawback: When two of the three rotations axes align, one rotation has no effect. This problem is known as gimbal lock, and this singularity can result in ill-conditioned optimization problems. The representations discussed below are designed to avoid this problem.

2.2.2 Quaternions

A rotation in the 3D space can also be represented by a unit quaternion [45]. Quaternions are hyper-complex numbers that can be written as the linear combination $a + bi + cj + dk$, with $i^2 = j^2 = k^2 = ijk = -1$. They can also be interpreted as a scalar plus a 3-vector $(a, \mathbf{v})$. A rotation about the unit vector $\vec{\omega}$ by an angle θ is represented by the unit quaternion

$$q = \left(\cos\left(\frac{1}{2}\theta\right), \vec{\omega}\sin\left(\frac{1}{2}\theta\right)\right).$$

To rotate a 3D point $\mathbf{M}$, we write it as a quaternion $p = (0, \mathbf{M})$ and take the rotated point p' to be

$$p' = q \cdot p \cdot \bar{q},$$

where $\cdot$ is the quaternion multiplication operator and $\bar{q} = \left(\cos\left(\frac{1}{2}\theta\right), -\vec{\omega}\sin\left(\frac{1}{2}\theta\right)\right)$ is the conjugate of q.

This representation avoids gimbal lock but estimation techniques must constrain the norm of q to remain equal to one. If q is estimated using numerical optimization techniques, this can be done by either adding the quadratic term $k(1 - \|q\|^2)$ to the objective function, where

k is a sufficiently large weight, or using a constrained optimization approach. This tends to increase the algorithmic complexity and is therefore not desirable in general. We now turn to the exponential-map representation that avoids both gimbal lock and additional constraints.

2.2.3 Exponential Map

Unlike the quaternion representation, the exponential map requires only three parameters to describe a rotation. It does not suffer from the gimbal lock problem and its singularities occur in a region of the parameter space that can easily be avoided.

Let $\vec{\omega} = [\omega_x, \omega_y, \omega_z]^T$ be a 3D vector and $\theta = \|\vec{\omega}\|$ be its norm. An angular rotation θ around an axis of direction $\vec{\omega}$ can be represented as the infinite series

$$\exp(\mathbf{\Omega}) = \mathbf{I} + \mathbf{\Omega} + \frac{1}{2!}\mathbf{\Omega}^2 + \frac{1}{3!}\mathbf{\Omega}^3 + \cdots \tag{2.3}$$

where $\mathbf{\Omega}$ is the skew-symmetric matrix

$$\mathbf{\Omega} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}. \tag{2.4}$$

This is the exponential map representation, which owes its name to the fact that the series of Equation (2.3) is of the same form as the series expansion of an exponential. It can be evaluated using Rodrigues' formula

$$\mathbf{R}(\mathbf{\Omega}) = \exp(\mathbf{\Omega}) = \mathbf{I} + \sin\theta\hat{\mathbf{\Omega}} + (1 - \cos\theta)\hat{\mathbf{\Omega}}^2, \tag{2.5}$$

where $\hat{\mathbf{\Omega}}$ is the skew-symmetric matrix corresponding to the unit vector $\frac{\vec{\omega}}{\|\vec{\omega}\|}$. At first sight, this formulation may seem to be singular when $\theta = \|\vec{\omega}\|$ goes to zero. However, this is not the case because Equation (2.5) can be rewritten as

$$\mathbf{R}(\mathbf{\Omega}) = \exp(\mathbf{\Omega}) = \mathbf{I} + \frac{\sin\theta}{\theta}\mathbf{\Omega} + \frac{(1 - \cos\theta)}{\theta^2}\mathbf{\Omega}^2, \tag{2.6}$$

which is not singular for small values of θ as can be seen by replacing $\frac{\sin\theta}{\theta}$ and $\frac{(1-\cos\theta)}{\theta^2}$ by the first two terms of their Taylor expansions.

This representation does have singularities that manifest themselves when $\|\vec{\omega}\| = 2n\pi$ with $n \geq 1$. In these cases, there is effectively no

rotation at all, no matter what the direction of $\vec{\omega}$. However, in practice, they can be easily avoided as follows. When $\|\vec{\omega}\|$ becomes too close to 2π, for example larger than π, $\vec{\omega}$ can be replaced by $(1 - \frac{2\pi}{\|\vec{\omega}\|})\vec{\omega}$, which represents the same rotation while having a norm smaller than π. Indeed, a rotation of θ radians about $\vec{v}$ is equivalent to the rotation of $2\pi - \theta$ radians about $-\vec{v}$.

In short, the exponential map represents a rotation as a 3-vector that gives its axis and magnitude. It avoids the gimbal lock problem of Euler angles and does not require an additional constraint to preserve the norm of a quaternion. It is therefore a very appropriate formulation and the rotation matrix corresponding to the vector can be computed according to Equation (2.6).

2.2.4 Linearization of Small Rotations

In 3D tracking applications, the camera motion between consecutive frames can often be assumed to remain small, along with the corresponding rotation angles. In such cases, it is appropriate to use a first order approximation of the rotation, which linearizes the pose estimation problem and simplifies the computations.

Let $\mathbf{M}'$ be the position of 3D point $\mathbf{M}$ after a rotation $\mathbf{R}$ about the origin by a *small* angle. All the formulations discussed above yield to the same first order approximation

$$\begin{aligned} \mathbf{M}' &= \mathbf{R}\mathbf{M} \\ &\approx (\mathbf{I} + \mathbf{\Omega})\mathbf{M} \\ &= \mathbf{M} + \mathbf{\Omega}\mathbf{M}, \end{aligned} \tag{2.7}$$

where the matrix $\mathbf{\Omega}$ is the skew-symmetric matrix of Equation (2.4).

2.3 Estimating the External Parameters Matrix

Continuously estimating the external parameters matrix from a video flux is considerably facilitated by a strong prior on the camera position given by the previously estimated camera positions, and this will be discussed in the next sections. Nevertheless, we present here several approaches to estimating these parameters without any prior knowledge of camera position but given some correspondences between 3D points

in the world coordinate system and their projections in the image plane. It is both a good introduction to other tracking approaches and useful in practice to initialize tracking algorithms. Estimating the external parameters when the internal parameters are known is often referred to as the pose estimation problem.

We assume here that we are given n correspondences between 3D points $\mathbf{M}_i$, and their projections $\mathbf{m}_i$. We are looking for the perspective projection matrix $\mathbf{P}$ that projects the points $\mathbf{M}_i$ on $\mathbf{m}_i$. In other words, we want to achieve for i $\mathbf{P}\tilde{\mathbf{M}}_i \equiv \tilde{\mathbf{m}}_i$ for all i, where $\equiv$ denotes the equality up to a scale factor.

2.3.1 How Many Correspondences are Necessary?

When the internal parameters are known, $n = 3$ known correspondences $\mathbf{M}_i \leftrightarrow \mathbf{m}_i$ produce 4 possible solutions. When $n = 4$ or $n = 5$ correspondences are known, [38] shows there are at least two solutions in general configurations, but when the points are coplanar and there is no triplets of collinear points, the solution is unique for $n \geq 4$. For $n \geq 6$, the solution is unique.

2.3.2 The Direct Linear Transformation (DLT)

The Direct Linear Transformation was first developed by photogrammetrists [125] and then introduced in the computer vision community [37, 54]. It can be used to estimate the whole matrix $\mathbf{P}$ of Equation (2.1) by solving a linear system even when the internal parameters are not known. Each correspondence $\mathbf{M}_i \leftrightarrow \mathbf{m}_i$ gives rise to two linearly independent equations in the entries $\mathbf{P}_{ij}$ of $\mathbf{P}$, that is

$$\frac{\mathbf{P}_{11}X_i + \mathbf{P}_{12}Y_i + \mathbf{P}_{13}Z_i + \mathbf{P}_{14}}{\mathbf{P}_{31}X_i + \mathbf{P}_{32}Y_i + \mathbf{P}_{33}Z_i + \mathbf{P}_{34}} = u_i,$$

$$\frac{\mathbf{P}_{21}X_i + \mathbf{P}_{22}Y_i + \mathbf{P}_{23}Z_i + \mathbf{P}_{24}}{\mathbf{P}_{31}X_i + \mathbf{P}_{32}Y_i + \mathbf{P}_{33}Z_i + \mathbf{P}_{34}} = v_i.$$

These equations can be rewritten in the form $\mathbf{Ap} = \mathbf{0}$, where $\mathbf{p}$ is a vector made of the coefficients $\mathbf{P}_{ij}$. The obvious solution $\mathbf{p} = \mathbf{0}$ is of course of no interest and the correct solution can be found from the Singular

Value Decomposition (SVD) of $\mathbf{A}$ as the eigen vector corresponding to the minimal eigen value.

The internal and external parameters can then be extracted from $\mathbf{P}$ [37, 54], but it would be dangerous to apply this approach for 3D tracking. In practice, the advisability of simultaneously estimating both internal and external parameters strongly depends on the geometry and the number of correspondences. In favorable configurations, 15 to 20 correspondences may suffice whereas, in unfavorable ones, even hundreds may not be enough. In such situations, it is always preferable to estimate the internal parameters separately. More specifically, for 3D tracking, using a calibrated camera and estimating only its orientation and position $[\mathbf{R} \mid \mathbf{t}]$ yields far more reliable results.

Once the camera calibration matrix $\mathbf{K}$ is known, the external parameters matrix can be extracted from $\mathbf{P}$ up to a scale factor as $[\mathbf{R} \mid \mathbf{t}] \backsim \mathbf{K}^{-1}\mathbf{P}$. The 3×3 matrix made of the first three columns of this matrix is not necessarily a rotation matrix, but a correction can easily be made [144].

In real conditions, pixel locations $\mathbf{m}_i$ are usually noisy. Because the actual error minimized by the DLT algorithm is an algebraic error as opposed to a meaningful physical quantity, the camera parameters estimated by this method should be refined by iterative optimization of the non-linear reprojection error. This will be discussed Subsection 2.4.

2.3.3 The Perspective-n-Point (PnP) Problem

The DLT method aims at estimating all 11 parameters of the projection matrix. It might also be applied to pose estimation, but it relies on an over-parameterization if the internal parameters are known. This results in reduced stability and requires more point correspondences than absolutely necessary. When the internal parameters have been estimated separately, a more satisfactory approach is to explicitly use this knowledge.

The 3 point pose problem, that is the estimation of the camera pose from 3 point correspondences also known as the perspective-3-point problem (P3P) has up to 4 real solutions in most cases. For 4 of more points, the solution is in general unique whether or not they are

coplanar, as long as they are not aligned. However in unfavorable cases there can be infinitely many solutions no matter how many points are supplied. Most notably, when there is a twisted cubic space curve of the particular type that can be inscribed on a circular cylinder, and which contains all of the points and the camera center, there are infinitely many solutions: There is one for each point along the twisted cubic and additional points on the cubic do not reduce the ambiguity. Some algorithms, notably linearized ones, fail for points lying on a cylinder, known to photogrammetrists as the *dangerous surface*, even if they fail to lie on a twisted cubic. The cubic can also degenerate to a circle attached orthogonally to a line, which can happen when the camera is directly above one corner of a square of points on a calibration grid. Wrobel's paper [48] lists the singular cases. In practice, one must also be aware that pose solutions can become unreliable even when the camera is quite far from these limiting cases.

Different approaches to the P3P problem have been proposed within the Computer Vision community [38, 51, 104]. They usually attempt to first estimate the distances $x_i = \|\mathbf{CM}_i\|$ between the camera center $\mathbf{C}$ and the 3D points $\mathbf{M}_i$, from constraints given by the triangles $\mathbf{CM}_i\mathbf{M}_j$. Once the x_i are known, the $\mathbf{M}_i$ are expressed in the camera frame as ${\mathbf{M}^\mathbf{c}}_i$. Then, $[\mathbf{R} \mid \mathbf{t}]$ is taken to be the displacement that aligns the points $\mathbf{M}_i$ on the ${\mathbf{M}^\mathbf{c}}_i$ and can be found in closed-form solution using quaternions [57] or singular value decomposition (SVD) [4, 58]. Solving for the x_i requires finding the roots of a fourth degree polynomial.

To remove the ambiguity, one additional correspondence is needed. A simple approach is to solve the P3P problem for subsets of three of the four points, and retain the common solution.

POSIT [31] is another popular way to solve the pose estimation problem for $n \geq 4$. It involves first computing an approximate solution assuming a scaled orthographic projection for the camera model, which means that an initial estimation of the camera position and orientation can be found by solving a linear system. A weight is assigned to each point based on the estimated pose and applied to its coordinates. A new projection is estimated from these scaled coordinates, and the process is iterated until convergence. This method is very simple to implement [60], but is relatively sensitive to noise. In addition, it

cannot be applied when the points are coplanar. [101] provides a similar approach for the planar case, but the two cases – coplanar and non-coplanar points – have to be explicitly distinguished.

[104] gives a quasi-linear algorithm for the case $n \geq 5$, which involves solving a linear system of the form $\mathbf{AX} = \mathbf{0}$ where $\mathbf{X}$ is a vector made the first powers of the distance $\|\mathbf{CM}\|$ between the camera center $\mathbf{C}$ and one of the 3D points $\mathbf{M}$, and $\mathbf{A}$ is made of the coefficients of fourth degrees polynomials. This approach does not degenerate in the coplanar case.

2.3.4 Pose Estimation from a 3D Plane

The camera pose can also be estimated from a planar structure when the internal parameters are known. This property has often been used in 3D tracking because the projections of planar structures are relatively easy to recover in images.

The relation between a 3D plane and its image projection can be represented by a homogeneous 3 ×3 matrix, called a homography matrix. Let us consider the $Z = 0$ plane. The expression of the homography $\mathbf{H}$ that maps a point $\mathbf{M} = (X, Y, 0)^T$ on to this plane and its corresponding 2D point $\mathbf{m}$ under the perspective projection $\mathbf{P} = \mathbf{K}[\mathbf{R} \mid \mathbf{t}]$ can be recovered by writing

$$\begin{aligned}
\tilde{\mathbf{m}} &= \mathbf{P}\tilde{\mathbf{M}} \\
&= \mathbf{K}\begin{pmatrix}\mathbf{R}^1 & \mathbf{R}^2 & \mathbf{R}^3 & \mathbf{t}\end{pmatrix}\begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} \\
&= \mathbf{K}\begin{pmatrix}\mathbf{R}^1 & \mathbf{R}^2 & \mathbf{t}\end{pmatrix}\begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \\
&= \mathbf{H}\begin{pmatrix} X \\ Y \\ 1 \end{pmatrix},
\end{aligned} \tag{2.8}$$

where $\mathbf{R}^1$, $\mathbf{R}^2$ and $\mathbf{R}^3$ respectively are the first, second and third column of the rotation matrix $\mathbf{R}$.

Conversely, once $\mathbf{H}$ and $\mathbf{K}$ are known, the camera pose can be recovered. The matrix $\mathbf{H}$ can be estimated from four correspondences $\mathbf{M}_i \leftrightarrow \mathbf{m}_i$ using a DLT algorithm similar to the one introduced Subsection 2.3.2 to estimate projection matrices. Since $\mathbf{H}_w^t = \mathbf{K}\begin{pmatrix}\mathbf{R}^1 & \mathbf{R}^2 & \mathbf{t}\end{pmatrix}$, the translation vector $\mathbf{t}$ and the first two columns of the rotation matrix $\mathbf{R}$ of the camera pose can be retrieved from the product $\mathbf{K}^{-1}\mathbf{H}_w^t$. The last column $\mathbf{R}^3$ is given by the cross-product $\mathbf{R}^1 \times \mathbf{R}^2$ since the columns of $\mathbf{R}$ must be orthonormal. As before, the camera pose can then be refined by non-linear minimization as described below.

2.3.5 Non-Linear Reprojection Error

The strength of the previous methods is that they do not require initial estimates and are fast. However they are sensitive to noise and, therefore, lack precision. In particular, the DLT algorithm provides a solution that minimizes an algebraic error, while it is better to consider instead a geometric error. For PnP algorithms, it is not clear which error is actually minimized.

If the measurements $\mathbf{m}_i$ are noisy, the camera pose should then be refined by minimizing the sum of the reprojection errors, that is the squared distance between the projection of the 3D points and their measured 2D coordinates. We can therefore write

$$[\mathbf{R} \mid \mathbf{t}] = \underset{[\mathbf{R}|\mathbf{t}]}{\arg\min} \sum_i \text{dist}^2(\mathbf{P}\tilde{\mathbf{M}}_i, \mathbf{m}_i), \tag{2.9}$$

which is optimal assuming that the measurement errors are independent and Gaussian. This minimization cannot be solved in closed form, but requires an iterative optimization scheme. As will be discussed in the next subsection, such a scheme usually requires an initial camera pose estimate, which can be provided by the previously described methods or, in tracking conditions, by a motion model applied on the previously estimated pose.

2.4 Least-Squares Minimization Techniques

Most 3D tracking algorithms estimate the camera pose as the solution of a least-squares minimization problem similar to the one of

Equation (2.9). More generally, this can be written as finding the pose that minimizes a sum of residual errors r_i, that is

$$\mathbf{p} = \arg\min_{\mathbf{p}} \sum_i r_i^2. \tag{2.10}$$

It can also be written in a matrix form as

$$\mathbf{p} = \arg\min_{\mathbf{p}} \|f(\mathbf{p}) - \mathbf{b}\|^2,$$

where $\mathbf{p}$ is a vector of parameters that defines the camera pose, as discussed in Subsection 2.2, $\mathbf{b}$ is a vector made of measurements, and f is some function that relates the camera pose to these measurements.

The linear least-squares is well-known. In the non-linear case, because objective functions are expressed as sums of squares and there are few parameters, the Newton-based methods are well-adapted.

2.4.1 Linear Least-Squares

In some cases, the function f is linear, that is the camera pose parameters $\mathbf{p}$ can be written as the unknowns of a set of linear equations that can be written in matrix form as $\mathbf{Ap} = \mathbf{b}$. Usually the size of vector $\mathbf{b}$ is larger than the number of parameters in $\mathbf{p}$, and $\mathbf{p}$ can be estimated as $\mathbf{p} = \mathbf{A}^+\mathbf{b}$, where

$$\mathbf{A}^+ = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$$

is the pseudo-inverse of $\mathbf{A}$.

Here, the measurements all have the same influence, but one may want to attribute a different weight to each measurement, for example according to its quality. Then, the problem becomes a Weighted Least-Squares, and can be solved as

$$\mathbf{p} = (\mathbf{A}^T\mathbf{W}\mathbf{A})^{-1}\mathbf{A}^T\mathbf{W}\mathbf{b},$$

where $\mathbf{W}$ is a weight matrix, usually taken to be diagonal.

In the Iterative Re-weighted Least-Squares, the estimation is iterated, and the coefficients of matrix $\mathbf{W}$ are re-estimated at each iteration according to the residuals. As will be seen below Subsection 2.5, this allows robust estimation, even in presence of spurious measurements.

2.4.2 Newton-Based Minimization Algorithms

When the function f is not linear, which is often the case because of the non-linear behavior of perspective projection, one may use the Gauss-Newton or the Levenberg-Marquardt algorithms. We give here a description of these algorithms, but an in-depth discussion can be found in [132]. Both algorithms start from an initial estimate $\mathbf{p}_0$ of the minimum and update it iteratively:

$$\mathbf{p}_{i+1} = \mathbf{p}_i + \boldsymbol{\Delta}_i,$$

where the step $\boldsymbol{\Delta}_i$ is computed differently depending on the specific method being used. In the Gauss-Newton algorithm, $\boldsymbol{\Delta}_i$ is chosen to minimize the residual at iteration $i+1$, and estimated by approximating f to the first order. Writing $f(\mathbf{p}_i + \boldsymbol{\Delta}) \approx f(\mathbf{p}_i) + \mathbf{J}\boldsymbol{\Delta}_i$, where $\mathbf{J}$ is the Jacobian matrix of f computed at $\mathbf{p}_i$ leads to solving

$$\begin{aligned}\boldsymbol{\Delta}_i &= \arg\min_{\boldsymbol{\Delta}} \|f(\mathbf{p}_i + \boldsymbol{\Delta}) - \mathbf{b}\| \\ &= \arg\min_{\boldsymbol{\Delta}} \|f(\mathbf{p}_i) + \mathbf{J}\boldsymbol{\Delta} - \mathbf{b}\| \\ &= \arg\min_{\boldsymbol{\Delta}} \|\epsilon_i + \mathbf{J}\boldsymbol{\Delta}\| \\ &= -\mathbf{J}^+\epsilon_i\end{aligned} \tag{2.11}$$

where $\epsilon_i = f(\mathbf{p}_i) - \mathbf{b}$ denotes the residual at iteration i, and $\mathbf{J}^+$ is the pseudo-inverse of $\mathbf{J}$. We can therefore write

$$\boldsymbol{\Delta}_i = -(\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\epsilon_i. \tag{2.12}$$

In the Levenberg-Marquardt (LM) algorithm, $\boldsymbol{\Delta}_i$ is computed in a slightly different way and it is taken to be

$$\boldsymbol{\Delta}_i = -(\mathbf{J}^T\mathbf{J} + \lambda\mathbf{I})^{-1}\mathbf{J}^T\epsilon_i. \tag{2.13}$$

The additional term $\lambda\mathbf{I}$ stabilizes the behavior of the Gauss-Newton algorithm: If the value of $\boldsymbol{\Delta}_i$ obtained with the previous equation leads to a reduction in the error, then this increment is accepted, and the value of λ is reduced. If not, the value of λ is raised instead, and a new $\boldsymbol{\Delta}_i$ is computed. This process is iterated until a value of $\boldsymbol{\Delta}_i$ is found that decreases the objective function. The algorithm stops when no such value can be found. When λ is small, the LM algorithm behaves

similarly to the Gauss-Newton algorithm. When λ becomes large, it starts behaving more like a conventional steepest gradient descent to guarantee convergence.

Both algorithms involve the computation of the Jacobian matrix of the function to minimize. This can be carried out by a numerical differentiation, but whenever possible, it is preferable both for speed and convergence to use an analytical expression for the derivatives.

A more sophisticated alternative would be to use a full Newton method [132], which relies on a second order approximation. For small problems, this sometimes yields better results. However, for larger ones, the improvement is rarely worth the increase in both algorithmic and computational complexity.

Of course, there is no guarantee that a numerical optimization will converge to the actual solution as it can get trapped in a spurious local minimum. That is why it should be initialized with as "good" an estimate as possible which can be expected to be close to the global minimum.

2.5 Robust Estimation

Robust estimation is usually indispensable in 3D tracking to counter the influence of spurious data. For example, if the camera pose is directly estimated using Equation (2.9) from a set of correspondences $\mathbf{M}_i \leftrightarrow \mathbf{m}_i$ that contains even a single gross error, the recovered pose may be far away the actual pose. RANSAC and the M-estimators are two popular techniques to avoid this.

These two methods are complementary. The M-estimators are good at finding accurate solutions but require an initial estimate to converge correctly. RANSAC does not require such an initial estimate, but provides a solution that does not take into account all the available data, and thus lacks precision. Therefore, when an initial estimate is available, one should use M-estimators. Otherwise, RANSAC can be used to provide the required estimate, which can then be refined using M-estimators.

2.5.1 M-Estimators

It is well known that least-squares estimation can also be seen as the maximization of the log-likelihood under the assumption that the observations are independent and have a Gaussian distribution. Because this distribution does not account for potential outliers, the least-squares formulation is very sensitive to gross errors. More realistic error models have been introduced by statisticians [59], and this leads to the following reformulation of the objective functions. Instead of minimizing

$$\sum_i r_i^2,$$

where r_i are residual errors as in Equation (2.10), one can minimize its robust version

$$\sum_i \rho(r_i), \tag{2.14}$$

where ρ is an M-estimator that reduce the influence of outliers. Many such functions have been proposed. Two of the most popular ones are depicted by Figure 2.4. They are the

- Huber estimator

$$\rho_{\text{Hub}}(x) = \begin{cases} x^2/2 & \text{if } |x| \leq c \\ c(|x| - c/2) & \text{otherwise.} \end{cases}$$

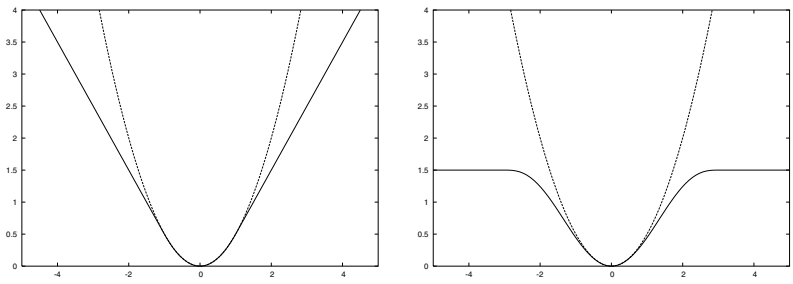


Fig. 2.4 Examples of robust estimators. Left: Graph of the Huber estimator for $c = 1$; Right: Graph of the Tukey estimator for $c = 3$. On both graphs, the graph of the least-squares estimator $\frac{x^2}{2}$ is also drawn in dash line for comparison.

- Tukey estimator

$$\rho_{\text{Tuk}}(x) = \begin{cases} \frac{c^2}{6}\left[1 - \left(1 - \left(\frac{x}{c}\right)^2\right)^3\right] & \text{if } |x| \leq c \\ c^2/6 & \text{otherwise.} \end{cases}$$

Both of these functions behave like the ordinary least-squares estimator $\frac{x^2}{2}$ when x is small. When $\|x\|$ is larger than c, the Huber estimator becomes linear to reduce the influence of large residual errors, while the Tukey estimator becomes flat so that large residual errors have no influence at all. Since the Huber estimator is convex, it makes the convergence to a global minimum more reliable. On the other hand, the Tukey estimator can be preferred to remove the influence of outliers when the minimization can be initialized close enough to the actual minimum.

The threshold c is usually taken to be proportional to the measured standard deviation of the residual errors for inlier data.

The Gauss-Newton and Levenberg-Marquardt algorithms can still be applied to minimize the sum (2.14) of residual errors after the introduction of M-estimators, even if the M-estimators can be complex functions. This is simply done by weighting the residuals r_i at each iteration step: Each r_i is replaced by $r'_i = w_i r_i$ such that:

$$(r'_i)^2 = w_i^2 r_i^2 = \rho(r_i),$$

therefore the weight w_i should be chosen as:

$$w_i = \frac{\rho(r_i)^{1/2}}{r_i}.$$

In the re-weighted least-squares estimation scheme, the matrix $\mathbf{W}$ can then be taken as $\mathbf{W} = \text{diag}(...w_i...)$. In the case of the Levenberg-Marquardt algorithm, $\boldsymbol{\Delta}_i$ can be computed as be changed as:

$$\boldsymbol{\Delta}_i = -(\mathbf{J}^T\mathbf{W}\mathbf{J} + \lambda\mathbf{I})^{-1}\mathbf{J}^T\mathbf{W}\epsilon_i.$$

In-depth details on using Gauss-Newton with robust cost functions can be found in Sections 10 and 11 of [132].

2.5.2 RANSAC

Even though it is a general method for robust estimation, RANSAC [38] has actually been developed in the context of camera pose estimation. It is very simple to implement, and has the advantage not to require an initial guess of the parameters to be estimated. On the other hand, it is usually slower than the minimization using M-estimators. By itself, it is also less accurate but can be followed by such minimization to refine the solution.

From the set of available data, RANSAC randomly extracts the smallest possible subsets required to generate model parameter hypotheses. This maximizes the odds that at least one subset will contain no gross errors and therefore produce a valid hypothesis. For example, the original RANSAC article uses a P3P algorithm to compute camera poses from randomly selected triplets of correspondences. For each pose, the number of 3D points that are reprojected close enough to their corresponding 2D points are treated as inliers. RANSAC retains the pose that gives rise to the largest number of inliers.

More formally, we want to estimate the parameters $\mathbf{p}$ of a model from a set S of measurements, some of them being erroneous. Assuming that the model parameters require a minimum of n measurements to be computed, N samples of n data points are randomly selected. Each sample is used to estimate model parameters $\mathbf{p}_i$ and to find the subset $S_i \subseteq S$ of points that are consistent with the estimate. The one that gives rise to the largest S_i is retained and refined by least-squares minimization using all the points in S_i. Alternatively, and even more effectively, one can perform a robust least-squares minimization using all the points in S, and use the estimate discussed above to initialize it: This allows to use inliers that may have initially been labeled as outliers.

Several parameters are involved in RANSAC. The error tolerance that is used to decide if a data point is consistent with a parameter estimate can be set as several times the standard deviations of the measurement errors. [38] provides a formula for the number of trials N: To ensure with a probability p that at least one of the sample is free from gross errors, N should be larger than $\log(1-p)/\log(1-w^n)$,

where w is the proportion of inlier data points. This value increases with the size of the subsets, and the outlier rate, but it is easy to see that it remains surprisingly small for reasonable values of p, w, and n.

2.6 Bayesian Tracking

Tracking algorithms, either explicitely or implicitely, aim at estimating the density of successive states $\mathbf{s}_t$ in the space of possible camera poses. In addition to the rotation and translation parameters, the $\mathbf{s}_t$ state vectors often include additional variables such as the translational and angular velocities. The state density that we note $p_t(\mathbf{s}_t)$ in the following is conditioned on the image measurements $\mathbf{z}_t$ performed up to time t i.e. we have $p_t(\mathbf{s}_t) = p(\mathbf{s}_t \mid \mathbf{z}_t \mathbf{z}_{t-1} \cdots \mathbf{z}_0)$. The measurements $\mathbf{z}_t$ depend on image features, such as the image locations at time t of specific feature points.

A common assumption is that the $\mathbf{z}_t$ measurements are mutually independent. Adding the natural assumption that the measurements have no influence on the dynamics of the states, it can be shown [61] that

$$p_t(\mathbf{s}_t) \propto p(\mathbf{z}_t \mid \mathbf{s}_t) p(\mathbf{s}_t \mid \mathbf{z}_{t-1} \cdots \mathbf{z}_0). \tag{2.15}$$

This gives us a propagation rule for the state density because it can be proven that the term $p(\mathbf{s}_t \mid \mathbf{z}_{t-1} \cdots \mathbf{z}_0)$ only depends on the previous density $p_{t-1}(\mathbf{s}_{t-1})$ and on the dynamics of the system [61]:

$$p(\mathbf{s}_t \mid \mathbf{z}_{t-1} \cdots \mathbf{z}_0) = \int_{\mathbf{s}_{t-1}} p(\mathbf{s}_t \mid \mathbf{s}_{t-1}) p_{t-1}(\mathbf{s}_{t-1}), \tag{2.16}$$

where $\int_{\mathbf{s}_{t-1}}$ is the integration over the set of possible values for the previous state $\mathbf{s}_{t-1}$. This term $p(\mathbf{s}_t \mid \mathbf{z}_{t-1} \cdots \mathbf{z}_0)$ can thus be interpreted as a prediction on $p_t(\mathbf{s}_t)$ made by applying the motion model on the previous density $p_{t-1}(\mathbf{s}_{t-1})$. This prediction is weighted in Equation (2.15) by the term $p(\mathbf{z}_t \mid \mathbf{s}_t)$, which is the observation density, and allows for taking into account the features observed in the incoming image.

Equation 2.15 is the equivalent of Bayes' rule for the discrete-time varying case. Therefore, formulating the problem in this generic manner is often referred to as "Bayesian tracking." In practice, many tracking systems ignore the probability density function altogether and

retain one single hypothesis for the camera pose, usually the maximum-likelihood one. However the Bayesian formulation can be useful to enforce temporal consistency, to provide error estimates, and to predict camera pose and feature locations in incoming frames. As we will see below, it also allows the fusion of multiple image cues in a statistically well-grounded manner.

This has most often been done in one of two ways characterized by the way the densities are represented. The first is Kalman filtering, which has a long history in 3D tracking but only considers Gaussian distributions. The other involves so-called Particle Filters. It allows for more general distributions but, while having largely been used in other tracking applications, it remains relatively marginal in the context of 3D tracking.

It should also be noted that, while the Bayesian formulation described here allows for recursive estimation of the current state density, it critically depends on the state density for the previous frame. This rescursive estimation is highly susceptible to error accumulation, especially when dealing with long sequences. [106, 105] derive a more general Bayesian framework to take into account not only one previous frame but a whole set of previous frames, in particular those with poses close to the current pose. [137] uses a similar idea but in a deterministic way.

2.6.1 Kalman Filtering

Kalman filtering is a generic tool for recursively estimating the state of a process and it has been often applied to 3D tracking. In fact, most of the algorithms described in the next sections can be used in conjunction with a Kalman filter. There are many extensive references to Kalman filtering [18, 140], and we only introduce it here in its most basic form. [45] is also a good reference dedicated to 3D tracking.

Linear Case The successive states $\mathbf{s}_t \in \mathrm{R}^n$ of a discrete-time controlled process are assumed to evolve according to a dynamics model of the form

$$\mathbf{s}_t = \mathbf{A}\,\mathbf{s}_{t-1} + \mathbf{w}_t\,, \tag{2.17}$$

where matrix $\mathbf{A}$ is called the *state transition matrix*, and $\mathbf{w}_t$ represents the process noise, taken to be normally distributed with zero mean. For tracking purposes, this relation is used to enforce a motion model, and is directly related to the term $p(\mathbf{s}_t \mid \mathbf{s}_{t-1})$ of Equation (2.16). For example, if the motion of the camera is assumed to have random, white noise accelerations with zero mean, the state vectors will comprise the 6 parameters of the camera pose, plus the translational and angular velocities.

The measurements $\mathbf{z}_t$, such as image location at time t of some feature points, are assumed to be related to the state $\mathbf{s}_t$ by a linear measurement model:

$$\mathbf{z}_t = \mathbf{C}\,\mathbf{s}_t + \mathbf{v}_t, \tag{2.18}$$

where $\mathbf{v}_t$ represents the measurement noise. This equation typically relates the camera pose stored in $\mathbf{s}_t$ to the considered image features, and corresponds to the term $p(\mathbf{z}_t \mid \mathbf{s}_t)$ in Equation (2.15). Note that, in fact, this relation is often non-linear, especially when modeling the camera as fully projective. As will be discussed in the next subsection, this can be handled by a natural extension of the approach presented here.

At each time step, the Kalman filter makes a first estimate of the current state called the *a priori* state estimate and that we denote $\mathbf{s}_t^-$. It is then refined by incorporating the measurements to yield the *a posteriori* estimate $\mathbf{s}_t$. $\mathbf{s}_t^-$, as well as its covariance matrix $\mathbf{S}_t^-$, are computed during a "time update" or prediction stage. Given knowledge of the process prior to time t and using the dynamic model, we can write

$$\begin{aligned}\mathbf{s}_t^- &= \mathbf{A}\,\mathbf{s}_{t-1},\\ \mathbf{S}_t^- &= \mathbf{A}\mathbf{S}_{t-1}\mathbf{A}^T + \mathbf{\Lambda}_\mathbf{w},\end{aligned}$$

where $\mathbf{S}_{t-1}$ is the *a posteriori* estimate error covariance for the previous time step, and $\mathbf{\Lambda}_\mathbf{w}$ is the process covariance noise that measures how well the motion model is respected in reality. The above expression of $\mathbf{S}_t^-$ is derived from the classical propagation formula. Next, the Kalman filter proceeds to a "measurement update" or correction. The *a posteriori* state estimate $\mathbf{s}_t$ and its covariance matrix $\mathbf{S}_t$ are now generated by incorporating the measurements $\mathbf{z}_t$ by writing

$$\begin{aligned}\mathbf{s}_t &= \mathbf{s}_t^- + \mathbf{G}_t(\mathbf{z}_t - \mathbf{C}\mathbf{s}_t^-),\\ \mathbf{S}_t &= \mathbf{S}_t^- - \mathbf{G}_t\mathbf{C}\mathbf{S}_t^-,\end{aligned}$$

where the Kalman gain $\mathbf{G}_t$ is computed as

$$\mathbf{G}_t = \mathbf{S}_t^- \mathbf{C}^T \left(\mathbf{C}\mathbf{S}_t^- \mathbf{C}^T + \mathbf{\Lambda}_\mathbf{v} \right)^{-1},$$

with $\mathbf{\Lambda}_\mathbf{v}$ being the covariance matrix of the measurements.

In the context of 3D tracking, the *a priori* state estimate $\mathbf{s}_t^-$ can be used to predict the location of an image feature. The predicted measurement vector $\mathbf{z}_t^-$ is indeed simply

$$\mathbf{z}_t^- = \mathbf{C}\mathbf{s}_t^-.$$

The uncertainty on this prediction can be represented by the covariance matrix $\mathbf{\Lambda}_\mathbf{z}$ estimated by propagating the uncertainty, which gives us

$$\mathbf{\Lambda}_\mathbf{z} = \mathbf{C}\mathbf{S}_t^- \mathbf{C}^T + \mathbf{\Lambda}_\mathbf{v}.$$

$\mathbf{z}_t^-$ and $\mathbf{\Lambda}_\mathbf{z}$ are useful to restrict the search for image features to a region of the image, as will be discussed in Subsection 4.1.3 and Subsection 4.5.2.

Extended and Iterated Extended Kalman Filter In the previous subsection, we assumed that the relationship between the state and the measurements was linear. As already noted, it is rarely the case in 3D tracking applications. The relationship should be expressed as

$$\mathbf{z}_t = c(\mathbf{s}_t, \mathbf{v}_t),$$

where c is a non-linear function. The Extended Kalman Filter (EKF) approximates this function c by its first order Taylor expansion, which allows the use of the formalism introduced below. This yields the following update equations

$$\begin{aligned} \mathbf{G}_t &= \mathbf{S}_t^- \mathbf{C}^T (\mathbf{C}\mathbf{S}_t^- \mathbf{C}^T + \mathbf{V}\mathbf{\Lambda}_\mathbf{v}\mathbf{V}^T)^{-1}, \\ \mathbf{s}_t &= \mathbf{s}_t^- + \mathbf{G}_t \left(\mathbf{z}_t - c(\mathbf{s}_t^-, 0) \right), \end{aligned} \tag{2.19}$$

where $\mathbf{C}$ is now the Jacobian of c with respect to the state $\mathbf{s}$ computed at $\mathbf{s}_t^-$. $\mathbf{V}$ is the Jacobian of c with respect to $\mathbf{v}$ and is often taken to be the Identity matrix in practice. The last update equation evaluates $\mathbf{S}_t$ with $\mathbf{C}$ computed at the updated state $\mathbf{s}_t$, which is usually considered

to be the best linearization point, giving self-consistent linearization estimates:

$$\mathbf{S}_t = \mathbf{S}_t^- - \mathbf{G}_t \mathbf{C} \mathbf{S}_t^-, \tag{2.20}$$

The Iterated Extended Kalman Filter (IEKF) involves iterating the updates of Equations (2.19) several times. It can be seen as the Gauss-Newton iterative method applied to the objective function the Kalman filter aims to minimize, whereas the EKF corresponds to performing only one Gauss-Newton step [70]. This reduces the inaccuracy resulting from the linearization of the function c.

Limitations The Kalman filter is a powerful and popular tool to combine noisy measurements from different image cues in a statistically well-grounded way. It is also useful to stabilize the camera trajectory using a motion model. However, this has a price. A simple motion model, such as one that assumes constant velocity, is fully justified in some applications such as visual servoing. But, as mentioned by several authors [6, 107], for applications that involve human motion that can be jerky, a low-order dynamical model is not very realistic. For Augmented Reality applications, this may result in some "lag" of the inserted virtual objects.

Another limitation comes from the fact that the measurements are often assumed to be mutually independent. While this assumption is not inherent to the Kalman filter formulation, it is difficult to avoid in practice without greatly increasing the complexity of the computation. In reality, the measurements are rarely independent, for example because they are projections in the image of points that all lie on the same 3D plane. This unwarranted assumption tends to result in an artificial reduction of the covariance of the *a posteriori* state estimate, thus making them unreliable.

2.6.2 Particle Filters

The probability distribution of states in Kalman filtering is restricted to be Gaussian, which is not optimal in ambiguous cases when multiple hypotheses may have to be considered.

One way out of this predicament is to use multiple Kalman filters [9] that represent the state density as a Mixture of Gaussiana. Particle Filters, such as Condensation [61, 16] or Monte Carlo filters [34], have been introduced as a more general representation by a set of weighted hypotheses, or particles. Another advantage is that they do not require the linearization of the relation between the state and the measurements. Each particle can be seen as an hypothesis for the state estimate. In other words particle filters can maintain several hypotheses over time, which gives them increased robustness.

Nevertheless, despite their strengths and even though they are popular for applications such as human body 3D tracking, there are surprisingly few papers on particle based 3D pose estimation. Exceptions are [115] that combines 2D alignment and 3D pose estimation and [128] that considers robot localization in mainly 2D environments.

We attribute this relative lack of popularity to two different causes. First a large number of particles – perhaps as many as several thousands when the motion is poorly defined – can be required, which slows down the tracking process. Next, for online applications, the system must provide a state estimate in each frame, usually taken to be the mean or the median of the particles. While robust, this estimate is not particularly accurate. This results in motion estimates that are not as smooth as they should be, which is a major drawback for many applications. However, maintaining several pose hypotheses in and representing the state density by a mixture of Gaussians certainly remains interesting, even if references are missing in rigid object 3D tracking.

3

Fiducial-Based Tracking

Vision-based 3D tracking can be decomposed into two main steps: first image processing to extract some information from the images, and second pose estimation itself. The addition in the scene of *fiducials*, also called *landmarks* or *markers*, greatly helps both steps: they constitute image features easy to extract, and they provide reliable, easy to exploit measurements for the pose estimation.

Here, we distinguish two types of fiducials. The first type is what we call "point fiducials" because each fiducial of this type give one point correspondence between the scene and the image. To obtain more information from each fiducial, it is possible to turn it into a planar shape with identifiable corners and we will refer to those as "planar fiducials": A single planar fiducial provides all six spatial constraints needed to define a coordinate frame.

3.1 Point Fiducials

Fiducials have been used for many years by close-range photogrammetrists. They can be designed in such a way that they can be easily detected and identified with an *ad hoc* method. Their image locations

can also be measured to a much higher accuracy than natural features. In particular, circular fiducials work best, because the appearance of circular patterns is relatively invariant under perspective distortion, and because their centroid provides a stable 2D position that can easily be determined with sub-pixel accuracy. The 3D positions of the fiducials in the world coordinate system are assumed to be precisely known: This can be achieved by hand, with a laser, or with a structure-from-motion algorithm. To facilitate their identification, the fiducials can be arranged in a distinctive geometric pattern. Once the fiducials are identified in the image, they provide a set of correspondences $\mathbf{M}_i \leftrightarrow \mathbf{m}_i$ and techniques similar to the ones we described in Subsection 2.3 can be applied to retrieve the camera pose.

For high-end applications, as found by close-range photogrammeters who have a long experience in this area, fiducial locations should be estimated carefully. In particular, there should be uniform lighting and a strong foreground-background contrast. Most of the professional solutions use circular or spherical fiducials made from retro reflective material, and cameras instrumented with a ring flash or other symmetric lighting coming from within a few degrees. Images are exposed so that the background is suppressed and the fiducials can be detected automatically due to their high contrast. It then become easier to estimate their center of gravity with subpixel accuracy. The targets should be at least 4-5 pixels across in the image and appear against a clear background with diameter at least 3 times the foreground diameter. Larger targets give similar, but not better results. All forms of saturation, blooming, and camera nonlinearities, as well as electronics that attempt to make the image look sharper, should be avoided. A small amount of optical defocus is best - say 0.5 pixel, just enough to remove visible jaggies. In these conditions one can easily get 1/10 pixel accuracy, and 1/100 is achievable with care and a good photogrammetric-quality camera. For high-end results, these requirements are not superfluous. For example, a jitter of amplitude as small as 1/10 of a pixel is easily visible in AR applications. Companies such as Geodetic services, Advanced Real-time Tracking, Metronor, ViconPeak, and AICON 3D Systems all propose commercial products based on this approach.

Low-cost, and lower-accuracy solutions, have also been proposed by the Computer Vision community. For example, the Concentric Contrasting Circle (CCC) fiducial [56] is formed by placing a black ring on a white background, or vice-versa. To detect these fiducials, the image is first thresholded, morphological operations are then applied to eliminate too small regions, finally a connected component labeling operation is performed to find white and black regions, as well as their centroids. In [56], four CCC's are placed in a flat rectangular pattern, and a fifth CCC is added on a side of the rectangle to remove ambiguities. The three collinear CCC's can be found by testing each subset of three points for collinearity.

[123] uses color-coded fiducials for a more reliable identification. Each fiducial consists of an inner dot and a surrounding outer ring, four different colors are used, and thus 12 unique fiducials can be created and identified based on their two colors. Some heuristics are also introduced: During tracking the fiducials should remain close to their predicted position; if the centroids of the outer and the inner regions are not close enough, the fiducial may be partially occluded, and is not taken into account in the pose computation.

Because the tracking range is constrained by the detectability of fiducials in input images, [23] introduces a system that uses several sizes for the fiducials. They are composed of several colored concentric rings, where large fiducials have more rings than smaller ones, and diameters of the rings are proportional to their distance to the fiducial center, to facilitate their identification. When the camera is close to fiducials, only small size fiducials are detected. When it is far from them, only large size fiducials are detected.

The previous extraction methods involve thresholds to segment the images, and can usually not be used under different lighting conditions without adjusting them. To handle variable lighting, [23] uses a rule-based approach that groups samples of similar color that are likely to all belong to the same fiducial. [98] uses homomorphic image processing, which is designed to eliminate the effect of non-uniform lighting. The thresholding operation is applied not on the image itself, but on the gradient of the logarithm of the image. This allows a robust detection

of the fiducials, even in presence of very non-uniform lighting, including blooming effects.

In order to expand the number of uniquely identifiable fiducials, [98] adds "data rings" between the traditional outer and inner rings. These additional rings are composed of sectors that are black or white, and can be used as a bar code, to encode the fiducial index. With this design, they can have as many as 3×2^{15} different fiducials.

While all the previous methods for fiducial detection use *ad hoc* schemes, [24] uses a machine learning approach which delivers significant improvements in reliability. The fiducials are made of black disks on white background, and sample fiducial images are collected under varying perspective, scale and lighting conditions, as well as negative training images. A cascade of classifiers is then trained on these data: The first step is a fast Bayes decision rule classification, the second one a powerful but slower nearest neighbor classifier on the subset passed by the first stage. At run-time, all the possible sub-windows in the image are classified using this cascade. This results in a remarkably reliable fiducial detection method.

3.2 Planar Fiducials

The fiducials presented above were all circular and only their center was used. By contrast, [71] introduces squared, black on white, fiducials, which contain small red squares for identification purposes. The corners are found by fitting straight line segments to the maximum gradient points on the border of the fiducial. Each of the four corners of such fiducials provides one correspondence $\mathbf{M}_i \leftrightarrow \mathbf{m}_i$, and the pose is estimated using an Extended Kalman filter.

[108, 67, 68] also use planar, rectangular fiducials, and show that a single fiducial is enough to estimate the pose. Their approach has become popular, both because it yields a robust, low-cost solution for real-time 3D tracking, and because a software library called ARToolKit is publicly available [3].

As most of the fiducials seen before, the fiducials of ARToolKit have a black border on a white background to facilitate the detection. An inner pattern allows the identification of the different fiducials. As

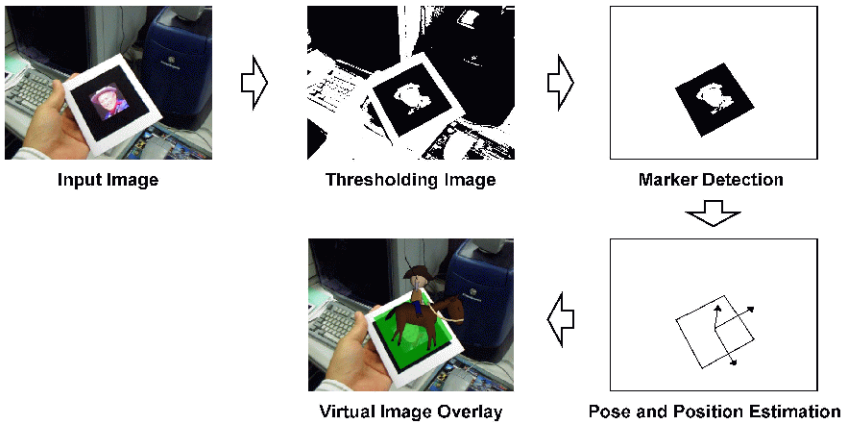


Fig. 3.1 Processing flow of ARToolKit: The marker is detected in the thresholded image, and then used to estimate the camera pose. (From [68], figure courtesy of H. Kato, M. Billinghurst, I. Poupyrev, K. Imamoto and K. Tachibana.)

before, the image is first thresholded, and the system looks for connected regions of black pixels. Regions whose outline contour can be fitted by four line segments are kept. Then each region is corrected to remove the perspective distortion and compare it by template matching with the known patterns. The correction is done by estimating the homography $\mathbf{H}$ that maps the pattern coordinates $\tilde{\mathbf{m}}_p$ on the screen coordinates $\tilde{\mathbf{m}}_s$

$$\tilde{\mathbf{m}}_s = \mathbf{H}\tilde{\mathbf{m}}_p$$

The coefficients of $\mathbf{H}$ can be estimated from the fiducial corners detected in the image and their coordinates in the pattern coordinate system [54]. If the internal parameters of the camera are known, the camera pose can be recovered from $\mathbf{H}$ as described in Subsection 2.3.4.

The whole process, the detection of the fiducials and the pose estimation, runs at a reported 30 frames-per-second, and therefore can be applied in every frame: The 3D tracking system does not require any hand-initialization, and is robust to fiducial occlusion. In practice, under good lighting conditions, the recovered pose is also accurate enough for Augmented Reality applications. These characteristics make ARToolKit a good solution for 3D tracking whenever engineering the scene is possible. Because it has a low CPU requirement, such markers-

based applications begin to be implemented on mobile devices such as PDA and mobile phones [139]. Free Computer Vision libraries on Symbian OS, which is the dominant operating system for smart phones are also in development [94], and lets hope for the development on such techniques on mobile devices.

4

Natural Features, Model-Based Tracking

Using markers to simplify the 3D tracking task requires engineering the environment, which end-users of tracking technology do not like or is sometimes even impossible, for example in outdoor environments. Whenever possible, it is therefore much better to be able to rely on features naturally present in the images. Of course, this approach makes tracking much more challenging and some 3D knowledge is often used to make things easier. The 3D knowledge can come in the form of a CAD model of a scene object, a set of planar parts, or even a rough 3D model such as an ellipsoid. Such models can be created using either automated techniques or commercially available products.

For completeness sake, we also present methods that do not require 3D scene models and simultaneously recover both camera trajectory and 3D structure. We will see that these methods can be make to work reliably in-real time. However, they cannot eliminate error accumulation and are not adequate in cases where the model semantics are important. For example, for automated grasping purposes, one must not only recover camera trajectory but also decide where exactly to grasp. Furthermore, the model-based techniques can usually be made to be more robust and fail-safe. In short there exists a trade-off between

the inconvenience of building the 3D model and the increased reliability it affords and choosing one approach over the other depends on the application at hand.

To organize the massive literature on the subject, we distinguish here two families of approaches depending on the nature of the image features being used. The first one is formed by edge-based methods that match the projections of the target object 3D edges to area of high image gradient. The second family includes all the techniques that rely on information provided by pixels inside the object's projection. It can be derived from optical flow, template matching or interest point correspondences.

Of course, during tracking, knowledge of the pose in the previous frames considerably simplifies the task: Once initialized, usually by hand or using an *ad hoc* method, a motion model is often used to predict the pose in the coming frame to help look for image features. The simplest such model is to assume that the camera does not move very much from one frame to the next one.

4.1 Edge-Based Methods

Historically, the early approaches to tracking were all edge-based mostly because these methods are both computationally efficient, and relatively easy to implement. They are also naturally stable to lighting changes, even for specular materials, which is not necessarily true of methods that consider the internal pixels, as will be discussed later. These methods can be grouped into two categories:

- One approach is to look for strong gradients in the image around a first estimation of the object pose, without explicitly extracting the contours [52, 2, 84, 35, 27, 136]. This is fast and general.
- Another approach is to first extract image contours, such as straight line segments and to fit the model outlines to these image contours [79, 45, 70, 73, 109]. The loss in generality can be compensated by a gain in robustness.

We discuss both kinds of approach below.

4.1.1 RAPiD

Because of its low computational complexity, RAPiD [52] was one of the first 3D tracker to successfully run in real-time. Even though many improvements have been proposed since, we describe it here in detail because many of its basic components have been retained in more recent systems. The key idea is to consider a set of 3D object points, called control points, that are most likely to project on high-contrast image edges. As shown in Figure 4.1, the control points can be sampled along the 3D model edges and in the areas of rapid albedo change. They can also be generated on the fly as points on occluding contours. The 3D motion of the object between two consecutive frames can be recovered from the 2D displacement of the control points.

Once initialized, the system performs a simple loop: For each frame, the predicted pose, which can simply be the pose estimated for the previous frame, is used to predict which control points will be visible and what their new locations should be. The control points are matched to the image contours, and the new pose estimated from these correspondences. For each control point, the system looks for its projection $\mathbf{m}'$ in the new image around $\mathbf{m}$, its projection in the previous frame. Because of the aperture problem, the position $\mathbf{m}'$ cannot be completely determined. As depicted by Figure 4.2 only the perpendicular distance l of $\mathbf{m}$ from the appropriate image edge is measured. Assuming that the orientations of the image edge and the model edge are nearly the same, a one-dimensional search for the image edge is conducted by looking in

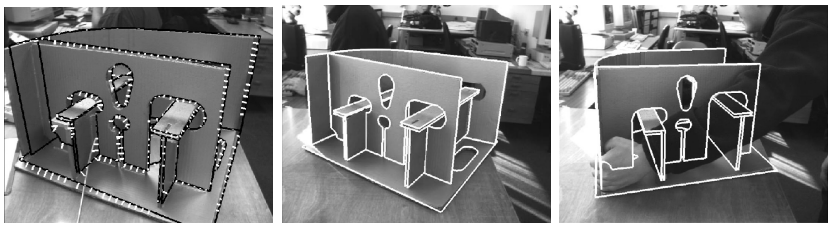


Fig. 4.1 In RAPiD-like approaches, control points are sampled along the model edges. The small white segments in the left image join the control points in the previous image to their found position in the new image. The pose can be inferred from these matches, even in presence of occlusions by introducing robust estimators. (From [35], figure courtesy of T. Drummond and R. Cipolla).

the direction of a vector $\vec{\mathbf{n}}$ from $\mathbf{m}$ where $\vec{\mathbf{n}}$ is a unit vector orthogonal to the projected object contour at point $\mathbf{m}$. For a fast implementation, the search is often performed in the horizontal, vertical, or diagonal direction, closest to the actual edge normal. [27] uses a precomputed convolution kernel function of the contour orientation to find only edges with an orientation similar to the reprojected contour orientation, as opposed to all edges in the scan-line.

In the original RAPiD formulation, the motion is estimated in the object coordinate system whose origin is located at $\mathbf{T} = (T_x, T_y, T_z)^T$ in camera coordinates, and whose axes are aligned with the camera coordinate system. A control point $\mathbf{P} = (P_x, P_y, P_z)^T$ in object coordinates is expressed as $\mathbf{M} = \mathbf{T} + \mathbf{P} = (X, Y, Z)^T$ in the camera frame. Its projection in the image is then $\mathbf{m} = \mathbf{KM}$. For simplicity, its projection is expressed in the following into the normalized image, that is one corrected to have focal length and aspect ratio unity, and origin at the optical center, as $\mathbf{m} = \left(\frac{X}{Z}, \frac{X}{Z}\right)^T$.

After a motion $\delta\mathbf{p}$ rotating the object about the object origin by $\Delta\mathbf{R}$ and translating it by $\delta\mathbf{t}$, the control point location in camera coordinates becomes $\mathbf{M}' = \mathbf{T} + \delta\mathbf{t} + \Delta\mathbf{RP}$. RAPiD assumes that the new image is acquired after a small motion, which makes it possible to linearize the object projection with respect to motion. As shown Subsection 2.7, the rotation $\Delta\mathbf{R}$ is approximated as $\Delta\mathbf{R} \approx \mathbf{I} + \mathbf{\Omega}$, where $\mathbf{\Omega}$ is a skew-symmetric matrix. Thus, we have $\mathbf{M}' \approx \mathbf{M} + \delta\mathbf{t} + \mathbf{\Omega P}$. The expression of the projection $\mathbf{m}'$ of $\mathbf{M}'$ can then be expanded in $\mathbf{\Omega}_x$, $\mathbf{\Omega}_y$, $\mathbf{\Omega}_z$ and $\delta\mathbf{t}$ and, by retaining only terms up to first order, becomes

$$\begin{aligned} u' &= u + \tfrac{1}{T_z + P_z}\left(\delta\mathbf{t}_x + \mathbf{\Omega}_y P_z - \mathbf{\Omega}_z P_y - u\left(\delta\mathbf{t}_z + \mathbf{\Omega}_x P_y - \mathbf{\Omega}_y P_x\right)\right), \\ v' &= v + \tfrac{1}{T_z + P_z}\left(\delta\mathbf{t}_y + \mathbf{\Omega}_z P_x - \mathbf{\Omega}_x P_z - v\left(\delta\mathbf{t}_z + \mathbf{\Omega}_x P_y - \mathbf{\Omega}_y P_x\right)\right). \end{aligned} \tag{4.1}$$

This can be written in matrix form as

$$\mathbf{m}' = \mathbf{m} + \mathbf{W}\delta\mathbf{p}, \tag{4.2}$$

where $\delta\mathbf{p} = (\mathbf{\Omega}_x, \mathbf{\Omega}_y, \mathbf{\Omega}_z, \delta\mathbf{t}_x, \delta\mathbf{t}_y, \delta\mathbf{t}_z)^T$ is a six-vector made of the rotation coefficients and the translation components, and $\mathbf{W}$ a 2×6 matrix that is a function of the coordinates of $\mathbf{T}$ and $\mathbf{P}$.

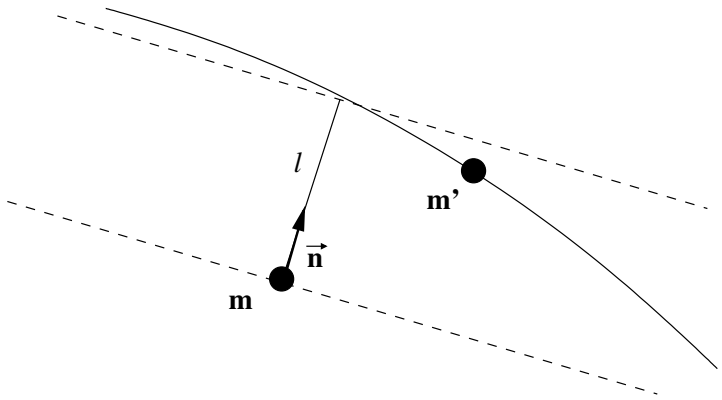


Fig. 4.2 Searching for the new control point position in RAPiD-like approaches. $\mathbf{m}$ is the predicted control point position, $\mathbf{m}'$ the actual control point position. The search is performed along the direction of vector $\vec{\mathbf{n}}$, and only the perpendicular distance l is measured.

The distance l of Figure 4.2 can be written as

$$l = \tilde{\mathbf{n}}^T \left(\mathbf{m}' - \mathbf{m} \right). \tag{4.3}$$

From Equations (4.3) and (4.2), each control point $\mathbf{M}_i$ then yields one equation of the form

$$\tilde{\mathbf{n}}_i \mathbf{W}_i \delta\mathbf{p} = l_i.$$

Given enough control points, $\delta\mathbf{p}$ can then be computed by minimizing the sum of squares of the perpendicular distances, which we write as

$$\delta\mathbf{p} = \arg\min_{\delta\mathbf{p}} \sum_i \left(\tilde{\mathbf{n}}_i \mathbf{W}_i \delta\mathbf{p} - l_i \right)^2. \tag{4.4}$$

Therefore, it can be recovered by solving a least-squares problem of the form

$$\mathbf{l} = \mathbf{A}\delta\mathbf{p},$$

where $\mathbf{l}$ is the vector made of the distances l_i and $\mathbf{A}$ depends on the $\tilde{\mathbf{n}}_i$ and $\mathbf{W}_i$. The solution can then be found using the pseudo-inverse of matrix $\mathbf{A}$, and taking $\delta\mathbf{p}$ to be

$$\delta\mathbf{p} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}\mathbf{l}.$$

Finally the pose $\mathbf{p}$ is incremented by $\delta\mathbf{p}$, which yields $\mathbf{p}^t = \mathbf{p}^{t-1} + \delta\mathbf{p}$.

In [52], some enhancements to this basic approach are proposed. When the edge response at a control point becomes too weak, it is not taken into account into the motion computation, as it may subsequently incorrectly latch on to a stronger nearby edge. As we will see below, this can also be handled using a robust estimator. An additional clue that can be used to reject incorrect edges is their polarity, that is whether they correspond to a transition from dark to light or from light to dark. A way to use occluding contours of the object is also given. In [36], integrating a Kalman filter into RAPiD is proposed.

The control points can be defined on the fly. [52] shows how profile edge points can be created along occluding contours defined by the model projection. [84] also discusses the discretization of the model edges visible at time t to produce the control points for the estimation of the pose at time $t + 1$.

4.1.2 Making RAPiD Robust

The main drawback of of the original RAPiD formulation is its lack of robustness. The weak contours heuristics is not enough to prevent incorrectly detected edges from disturbing the pose computation. In practice, such errors are frequent. They arise from occlusions, shadows, texture on the object itself, or background clutter.

Several methods have been proposed to make the RAPiD computation more robust. [35] uses a robust estimator and replaces the least-squares estimation by an iterative re weighted least-squares to solve the new problem. [84] uses a framework similar to RAPiD to estimate a 2D affine transformation between consecutive frames, but substitutes a robust estimator for the least-squares estimator of Equation (4.4). The affine transformation is used to infer an approximate 3D pose, which is then refined as will be discussed Subsection 4.1.4.

In fact, when using a more powerful minimization algorithm, linearizing the problem is not required, instead one can minimize the actual distances between the detected features and the reprojected 3D primitives. Let the $\mathcal{M}_i$ be those primitives, and let $\left\{\mathbf{m}'_{i,j}\right\}$ be the set

of associated image features, the pose can now be estimated as:

$$\mathbf{p} = \arg\min_{\mathbf{p}} \sum_{i,j} \rho\left(\text{dist}\left(\mathbf{P_p}\mathcal{M}_i, \mathbf{m}'_{ij}\right)\right), \tag{4.5}$$

where $\mathbf{P_p}$ is the projection defined by parameters $\mathbf{p}$ and $\mathbf{P_p}\mathcal{M}_i$ denotes the 2D curve obtained projecting $\mathcal{M}_i$. For example, [86] discusses the computation of the relevant Jacobian matrices when the 3D primitives such as straight lines segments, circles or occluding boundaries of cylinders can be defined analytically. [117] considers free-form curves and uses an approximation of the distance.

In the approaches described above, the control points were treated individually, without taking into account that several control points are often placed on the same edge, and hence their measurements are correlated. By contrast, in [2, 117] control points lying on the same object edge are grouped into primitives, and a whole primitive can be rejected from the pose estimation. In [2], a RANSAC methodology is used to detect outliers among the control points forming a primitive. If the number of remaining control points falls below a threshold after elimination of the outliers, the primitive is ignored in the pose update. Using RANSAC implies that the primitives have an analytic expression, and precludes tracking free-form curves. By contrast, [117] uses a robust estimator to compute a local residual for each primitive. The pose estimator then takes into account all the primitives using a robust estimation on the above residuals.

When the tracker finds multiple edges within its search range, it may end-up choosing the wrong one. To overcome this problem, in [35], the influence of a control point is inversely proportional to the number of edge strength maxima visible within the search path. [136] introduces another robust estimator to handle multiple hypotheses and retain all the maxima as possible correspondents in the pose estimation.

4.1.3 Explicit Edge Extraction

The previous approaches rely on matching points sampled on edges. An alternative approach is to globally match model primitives with primitives extracted from the image [79, 45, 70, 73, 109], as depicted by Figure 4.3. In these specific examples, the primitives are straight

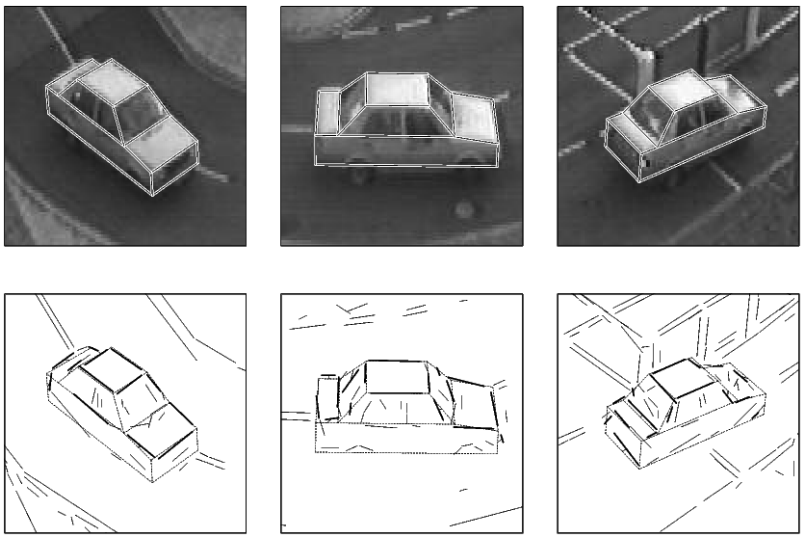


Fig. 4.3 Pose estimation from correspondences between 3D model edge segments M_i and 2D image segments D_i. (From Figure 13 from [70], reproducted with kind permission of Springer Science and Business Media).

line segments, but, in theory, they could be more complex parametric curves.

For each image, straight line edge segments are extracted, while the model edge segments are projected with respect to the predicted pose. The matching is based on the Mahalanobis distance of line segment attributes. For example, in [70] segments are represented by $\mathbf{X} = (c_x, c_y, \theta, l)$ defined by the coordinates of the middle point, the orientation and the length of the segment [32]. Given the attribute vector $\mathbf{X}_m$ of a model segment and the attribute vector $\mathbf{X}_d$ of an extracted segment, the Mahalanobis distance between $\mathbf{X}_m$ and $\mathbf{X}_d$ can be then defined as

$$d = (\mathbf{X}_m - \mathbf{X}_d)^T (\Lambda_m + \Lambda_d)^{-1} (\mathbf{X}_m - \mathbf{X}_d), \tag{4.6}$$

where Λ_d is the covariance matrix of $\mathbf{X}_d$, and depends on the extraction procedure. The covariance matrix Λ_m of a model segment depends on the covariance matrix of the predicted pose estimation. [73] also integrates the uncertainty in the Hough transform used for segment

extraction and limits its search to the uncertainty region predicted by the Λ_m matrices.

An iterative procedure is used to find the best correspondences between 3D model edge segments M_i and 2D image segments D_i, while estimating the pose. In [70], a model segment M_i is matched with the closest data segment D_i according to the Mahalanobis distance of Equation (4.6), if this distance is lower than a threshold. The pose $\mathbf{p}$ is then estimated by minimizing

$$\sum_i \left(\mathbf{X}_d^i - \mathbf{X}_m^i(\mathbf{p})\right)^T \Lambda_d^i \left(\mathbf{X}_d^i - \mathbf{X}_m^i(\mathbf{p})\right), \tag{4.7}$$

with respect to $\mathbf{p}$ where $\mathbf{X}_m^i(\mathbf{p})$ is the attribute vector of the model segment M_i projected with respect to the pose $\mathbf{p}$. An additional term can be added to the criterion of Equation (4.7) to account for a motion model. The minimization is performed using the Levenberg-Marquardt algorithm. The process is repeated until a stable pose is found.

Such approaches, which are only adapted to polyhedral object tracking, have been applied to vehicle and robot arm tracking, but they seem to have fallen out of use and been replaced by RAPiD like algorithms. We believe this can be attributed to bottom-up nature of the edge-extraction process, which makes it unreliable. The RAPiD approach both avoids this drawback thanks to the local search around an *a priori* pose and tends to be significantly faster.

4.1.4 Direct Optimization on Gradients

[72, 84, 8] propose to recover the pose by fitting the model projection directly to the image gradients. A simple approach is to maximize the gradient norm along the model reprojection but there is no guarantee that the model edges should correspond to high intensity gradient values.

It is better to take into account the expected direction of the projected contour: [84] proposes to minimize the sum of the values $\frac{\nabla I.\vec{\mathbf{n}}}{\|\nabla I\|^2}$, where ∇I denotes the spatial gradient of the image I, and $\vec{\mathbf{n}}$ the expected direction. This measure tends to support locations where the gradient is both strong and in the expected direction. [72] maximizes

the correlation between the predicted and the measured gradient norm plus an additional term to constrain the motion.

Such approaches require a very good initial estimate to converge to the correct pose. Therefore, they are best used as a refinement step.

4.2 Optical Flow-Based Methods

Optical flow is the apparent motion of the image projection of a physical point in an image sequence, where the velocity at each pixel location is computed under the assumption that projection's intensity remains constant. It can be expressed as

$$\mathbf{m}' = \mathbf{m} + \begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} \mathrm{d}t,$$

where $\mathbf{m}$ the projection of a point in an image I at time t, $\mathbf{m}'$ its corresponding location in the next image, captured at time $t + dt$, and $(\dot{u}, \dot{v})^T$ the apparent speed of the 2D motion at $\mathbf{m}$. The vector field of the $(\dot{u}, \dot{v})^T$ is the optical flow. It can be computed using the Lucas-Kanade method [83] for example, which adopts a multiscale approach and assumes that the optical flow varies smoothly.

4.2.1 Using Optical Flow Alone

An early attempt to use optical flow as a cue for 3D tracking relied on the well-known normal optical flow constraint [75]

$$\left(\frac{\partial I}{\partial u}, \frac{\partial I}{\partial v} \right) \left(\mathbf{m}' - \mathbf{m} \right) + \frac{\partial I}{\partial t} = 0 \,, \tag{4.8}$$

where $\frac{\partial I}{\partial u}$ and $\frac{\partial I}{\partial v}$, and $\frac{\partial I}{\partial t}$ are respectively the spatial and temporal derivatives of the image computed at location $\mathbf{m}$. Equation (4.8) is similar to Equation (4.3) used by RAPiD. That means that if we know the 3D correspondent $\mathbf{M}$ for some $\mathbf{m}$ on the model projection, we can infer the model displacement using the same approach as in RAPiD by simply replacing Equation (4.3) by Equation (4.8).

Nevertheless, this approach has two important drawbacks: First, for large motions, there exists a large linearizing error in the normal optical flow constraint that affects the estimation. Second, while the control

points on edges provide absolute information and act as anchors, relying on optical flow causes error accumulation, which quickly results in drift and tracking failure.

To avoid error accumulation, [75] uses a render-feedback loop. Instead of applying the previous method to the actual image, it applies it to a synthetic image of the object rendered under a pose predicted using a motion model. If the motion model is valid, the synthetic image is closer to the new image, the error due to the linearity assumption is smaller and the computed motion more reliable. The process can then be iterated to suppress the drift effect. Unfortunately, this method requires a motion model to handle fast motion and is sensitive to lighting effects, since it is difficult to render a synthetic image that takes illumination into account. It is also relatively slow since several image synthesis per frame are required. Nevertheless, it was one of the first successful methods to use Computer Graphics techniques to solve a Computer Vision problem.

[10] applies the regularized optical-flow method developed in [15] to 3D tracking to handle fast motion. The optical flow is computed using a generic algorithm, the displacement $\delta\mathbf{p}$ of the model between images t and $t+1$ is taken to be the best displacement that best matches the optical flow in a least-squares sense. It is taken to be

$$\delta\mathbf{p} = \arg\min_{\delta\mathbf{p}} \sum_i \rho(\|\mathbf{F_M}(\mathbf{m}_i, \delta\mathbf{p}) - \mathbf{F_I}(\mathbf{m}_i)\|), \tag{4.9}$$

where the $\mathbf{m}_i$ are pixel locations on the model reprojection, $\mathbf{F_I}(\mathbf{m}_i)$ is the optical flow computed at $\mathbf{m}_i$, and $\mathbf{F_M}(\mathbf{m}, \delta\mathbf{p})$ is the expected flow at $\mathbf{m}$ for an object displacement $\delta\mathbf{p}$ from pose $\mathbf{p}$. This approach handles faster motion, and can run at several frames per second, but is still prone to drift.

4.2.2 Combining Optical Flow and Edges

Several authors combine edge and optical flow information to avoid error accumulation. For example, [49] uses a Kalman filter to combine the two cues. The edges are taken into account by a term similar to the one of Equation (4.5), the optical flow by a term analogous to the one of Equation (4.9).

A different combination is proposed in [29], where the optical flow information is treated as a hard constraint. In this work, the deformations of the tracked shape, a human face, are also estimated but we will drop here the deformation terms. Equation (4.8) is applied at several locations to obtain a linear system that can be written compactly as

$$\mathbf{B}\delta\mathbf{p} + \mathbf{I_t} = \mathbf{0}, \tag{4.10}$$

where matrix $\mathbf{B}$ depends of the pose $\mathbf{p}$ and the image spatial gradients at time t, and $\mathbf{I_t}$ is a vector made of the temporal gradient at the chosen locations. A similar linear system is derived from the edges in image locations expected to have high gradient values. It is written as

$$\delta\mathbf{p} = \mathbf{L}\mathbf{f}, \tag{4.11}$$

where vector $\mathbf{f}$ is made of image forces directly related to the image gradients at these locations. Instead of solving simultaneously Equations (4.10) and (4.11), which is problematic since it would require estimating relative weights for the two contributions, Equation (4.10) is treated as a hard constraint, resulting in the constrained system

$$\delta\mathbf{p} = \mathbf{L}\mathbf{f} \text{ , subject to } \mathbf{B}\delta\mathbf{p} + \mathbf{I_t} = \mathbf{0}.$$

This is solved by using Lagrange multipliers, which transforms the problem into a new, larger, unconstrained linear system. This method is not naturally robust because it relies on matrix $\mathbf{B}$ that depends on possibly noisy image gradients. Robustness is obtained by reformulating the new system using an iterated extended Kalman filter.

This approach yields impressive results especially because it estimates not only the motion but also the deformations of a face model. Nevertheless, it still depends on the brightness constancy assumption during optical flow computation, and major lighting changes can cause tracking failure. Because of the linearization in the optical flow equation cue, the range of acceptable speeds is also limited.

4.3 Template Matching

The Lucas-Kanade algorithm [83, 7] was originally designed to compute the optical flow at some image locations, but in fact has a more general

purpose and can be used to register a 2D template to an image under a family of deformations. It does not necessarily rely on local features such as edges or interest points, but on global region tracking, which means using the whole pattern of the object to be tracked. Such a method can be useful to treat complex objects that are difficult to model using local features. While it can be computationally expensive, [50] showed that under some conditions, it can be effectively formulated. Since then it has been extended by several authors and applied to 3D tracking [19, 65, 66].

4.3.1 2D Tracking

The general goal of the Lucas-Kanade algorithm is to find the parameters $\mathbf{p}$ of some deformation $\mathbf{f}$ that warps a template T into the input image I_t, where the $\mathbf{f}$ deformation can be a simple affine warp as well as a much more complex one. This is done by minimizing

$$O(\mathbf{p}) = \sum_j \left(I_t(\mathbf{f}(\mathbf{m}_j; \mathbf{p})) - T(\mathbf{m}_j)\right)^2, \tag{4.12}$$

the sum of squared errors computed at several $\mathbf{m}_i$ locations.

The Lucas-Kanade algorithm assumes that a current estimate of $\mathbf{p}$ is known, which is reasonable for tracking purposes. It iteratively solves for $\mathbf{p}$ by computing $\mathbf{\Delta}_i$ steps that minimize

$$\sum_j \left(I_t(\mathbf{f}(\mathbf{m}_j; \mathbf{p}_j + \mathbf{\Delta})) - T(\mathbf{m}_j)\right)^2.$$

As in the Gauss-Newton algorithm of Subsection 2.4.2, the $I_t(\mathbf{f}(\mathbf{m}_j; \mathbf{p}+\mathbf{\Delta})$ term is linearized by performing a first order Taylor expansion. This lets us write

$$\mathbf{\Delta}_i = \mathbf{A}\delta\mathbf{i}, \tag{4.13}$$

where $\mathbf{A}$ is the pseudo-inverse of the Jacobian matrix $\mathbf{J}$ of $I_t(\mathbf{f}(\mathbf{m}_j; \mathbf{p}))$ computed at $\mathbf{p}_j$, and $\delta\mathbf{i} = [T(\mathbf{m}_j) - I_t(\mathbf{f}(\mathbf{m}_j; \mathbf{p})]$ is the vector of intensity differences between the template and the warped input image. $\mathbf{J}$ depends both on the image gradients computed at pixel locations

$\mathbf{f}(\mathbf{m}_j; \mathbf{p})$ and on the Jacobian of the warping function. It can be computed as

$$\mathbf{J} = \sum_j \left(\frac{\partial I_t}{\partial \mathbf{f}}\right)\left(\frac{\partial \mathbf{f}}{\partial \mathbf{p}}\right)(\mathbf{m}_j).$$

In this derivation, the function $\mathbf{f}$ can be arbitrarly complex. In the general case, this means that the the pseudo-inverse of $\mathbf{J}$ must be recomputed at each iteration, which is computationally expensive. As we will see below, the method can be made to be much more efficient by restricting $\mathbf{f}$ to a specific class.

4.3.2 Jacobian Formulation

[50] shows that for some classes of displacements, including linear models, the previous iteration step can be replaced by

$$\begin{aligned}\boldsymbol{\Delta}'_j &= \mathbf{A}'\delta\mathbf{i},\\ \boldsymbol{\Delta} &= \Sigma(\mathbf{p})^{-1}\boldsymbol{\Delta}'_j,\end{aligned}$$

where $\Sigma(\mathbf{p})$ is a matrix that only depends on the displacement $\mathbf{p}$. This time, the matrix $\mathbf{A}'$ does not depend on time-varying quantities and can be computed beforehand. [50] restricts the estimation to a single step, but it could be iterated as in the Gauss-Newton algorithm. The final algorithm requires about a hundred image accesses and subtractions to compute $\delta\mathbf{i}$, a few hundred of multiplications and a matrix inversion to compute the displacement $\boldsymbol{\Delta}$.

[30] gives a similar derivation but also proposes an efficient sampling of the target region to reduce even further the online computation cost without losing too much image information. Ideally the subset that reduces the expected variance of recovered motion should be retained. In practice, the combinatorics are too large, and [30] proposes to compute the expected for individual pixels and constructs a random subset of the best pixels, constrained to be well spread on the target image.

A limitation of the formulation given above is sensitive to changes in illumination of the target region. To handle such variations, [50] adds a linear combination of basis vectors to the expression of $\mathbf{I}_t(\mathbf{p} + \boldsymbol{\Delta}_j)$. These basis vectors can be learned from a set of training images of the

target region, taken under varying illumination. It is shown that the motion can still be computed using no more online computation than before.

Another drawback is that it does not handle potential partial occlusions of the tracked regions, which could result in tracking failure. To solve this problem, [50] turns the least-squares formulation of Equation (4.12) into a robust one by introducing a robust estimator. The motion is then recovered using an iteratively re-weighted least-squares approach.

This method was originally used to track in real-time human faces assuming a 2D affine transformation for the motion model.

4.3.3 Hyperplane Approximation

The approach presented in [66] relies on a reinterpretation of Equation (4.13) that yields a faster implementation than the Jacobian formulation discussed above. It treats the equation as an approximation by hyperplanes. This allows the estimation of the $\mathbf{A}'$ matrix during a learning stage. It is done by producing random small disturbances $\boldsymbol{\Delta}$ around the reference position, for which the change of brightness $\delta\mathbf{i}$ can be measured by virtually moving the region of interest. A matrix $\delta\mathbf{p}$ that maps the $\delta\mathbf{i}$ to the $\delta\mathbf{p}$ can then be estimated in the least-squares sense. The paper [66] experimentally shows that such a matrix gives a more reliable approximation of the relation between image differences and the motion. This can be explained by the fact that the objective function of Equation (4.12) has many local minima, which this approach allows the algorithm to skip.

4.3.4 From 2D to 3D Tracking

The template matching approach of Subsection 4.3.3 has been used to track 3D objects. As depicted by Figure 4.4, [65] uses a homography to model the object pose $\mathbf{p}$ to track 3D planar objects. A homography maps two views of the same 3D plane seen from different camera positions and can be represented by a 3×3 matrix. As shown in Subsection 2.3.4, once the homography is known for a view of the plane,

Fig. 4.4 The book cover is modeled as a plane, and tracked using a template matching-based approach. (From [66], figure courtesy of F. Jurie and M. Dhome.)

the camera position with respect to that plane can be recovered if the internal camera parameters are known.

A few improvements can be added to the original method described in [66]. The locations considered to compute the pose should be taken following the method described in [30]. At least a simple heuristics is to retain locations to strong gradients, with some care to take well spread locations. These locations cannot be taken on the border of the object: in the contrary case, they could lie on the background once the object has moved, and disturb the intensity difference computation. The locations intensities should be normalized to be robust to most of lighting changes. Finally, while the original paper only discusses single iteration step estimation, several iterations can be performed to allow for fast motion. It can be done using a cascade of $\mathbf{A}'$ matrices instead of a single one, where the first matrices of the cascade are trained from large motions, and the last ones capture finer and finer motions.

[19] also uses this approach to track the position and orientation of a human head using a non-planar surface model. The head is modeled

as a generalized cylinder. It can thus be described as a parametric surface, where a 3D point $\mathbf{M}$ on the model surface is a function of two coordinates (s, t) in the surface's parametric coordinate system $\tilde{\mathbf{M}} = \mathbf{x}(s, t)$. This allows the definition of a relatively simple warping between the image and the texture map on the head model, and a template matching approach similar to the one of [50] is used to recover the six degrees of freedom of the 3D model. A confidence map is also introduced is to account for the fact that pixels are not equally informative due to perspective distortion. They are assigned a confidence level proportional to the image area of the triangle they belong to. A motion model is also used to regularize the recovered motion. The resulting tracker overcomes the biggest problem of using a planar approximation for the face, that is instability in presence of out of plane rotations.

4.4 Interest Point-Based Methods

We now turn to methods that use localized features instead of global ones, which has several advantages. In the same way as edge-based methods, they rely on matching individual features across images and are therefore easy to robustify against partial occlusions or matching errors. Illumination invariance is also simple to achieve. But, unlike edges methods, they do not get confused by background clutter and exploit more of the image information, which tends to make them more robust. They are similar in some ways to optical flow approaches, but they allow faster motions since no small inter-frame motion assumption is made.

The local features can be patches manually selected in several registered views of the target object during a preliminary stage [107, 135]. An object feature is then defined by its 3D object location and by a template image that captures its image appearance. Once initialized, the algorithms perform a simple loop similar to the one of edge-based methods. For each frame, the object features are matched by localizing feature templates in search windows around hypothesized locations using steerable filters to compensate for image-plane rotations, and normalized cross-correlation for insensitivity to lighting changes. The

pose is then obtained from these model to image correspondences. Note that in [107], aspect changes are handled by initially building an aspect table that associates object features with discrete viewpoints in which they can be expected to be seen.

Of course, the above algorithms require both manual intervention and actual expertise to select appropriate patches. A far more effective and practical approach is to have the system itself choose the features for optimal performance. We refer to these automatically chosen features as interest points. In the remainder of this subsection, we first discuss their extraction and matching. We then present ways of using them for 3D tracking.

4.4.1 Interest Point Detection

Matching only a subset of the image pixels reduces computational complexity while increasing reliability if they have been correctly chosen. This task usually falls on an "interest operator," which should select points with the following properties [40]: The patches surrounding them should be textured so that they can be easily matched. They should be different from their immediate neighbors to eliminate edge-points that can give rise to erroneous matches. Similarly, pixels on repetitive patterns should also be rejected or at least given less importance to avoid ambiguous matches. Finally, the selection should be repeatable, which means that the same points should be selected in different images of the same scene, despite perspective distortion or image noise. This last property is important because the precision and the pose estimation directly depends on the invariance of the selected position.

Such operator were already in use in the 1970's for tracking purposes [97, 96]. In these early works, pixels with the largest minimum variance of intensity differences in the four directions were retained. Numerous other methods have been proposed since and [33, 122] give good surveys. Most of these techniques involve second order derivatives, and results can be strongly affected by noise. Currently popular interest point detectors, sometimes called the Förstner operator [40], the

Plessey operator, the Harris-Stephen detector [53], or the Shi-Tomasi detector [116], all rely on the auto-correlation matrix

$$\mathbf{Z} = \begin{pmatrix} \sum I_u^2 & \sum I_u I_v \\ \sum I_u I_v & \sum I_v^2 \end{pmatrix},$$

computed at each pixel location. The coefficients of $\mathbf{Z}$ are the sums over a window of the first derivatives I_u and I_v of image intensities with respect to (u, v) pixel coordinates. The derivatives can be weighted using a Gaussian kernel to increase robustness to noise [111]. The derivatives should also be computed using a first order Gaussian kernel. This comes at a price since it can reduce localization accuracy.

As discussed in [40], the pixels can be classified from the behavior of the eigen values of $\mathbf{Z}$: Pixels with two large, approximately equal eigen values are good candidates for selection. [53] defines a "textureness" measure from the trace and the determinant of $\mathbf{Z}$ to avoid explicit computation of the eigen values. [116] shows that locations with two large eigenvalues of $\mathbf{Z}$ can be tracked reliably especially under affine deformations. It therefore focuses on locations where the smallest eigen value is higher than a threshold. Interest points can then taken to be the locations $\mathbf{m}_i$ that are local maxima of the chosen measure above a predefined threshold. It should be noted that these measures have a relatively poor localization accuracy and are computationally demanding. However they are widely used they have proved effective and are easy to implement.

4.4.2 Interest Point Matching

To estimate motion, one can then match sets of interest points $\{\mathbf{m}_i\}$ and $\{\mathbf{m}'_j\}$ extracted from two images taken from similar, and often successive, viewpoints. A classical procedure [145] runs as follows. For each point $\mathbf{m}_i$ in the first image, search in a region of the second image around location $\mathbf{m}_i$ for point $\mathbf{m}'_j$. The search is based on the similarity of the local image windows centered on the points, which strongly characterizes the points when the images are sufficiently close. The similarity can be measured using the zero-normalized cross-correlation

that is invariant to affine changes of the local image intensities, and make the procedure robust to illumination changes. To obtain a more reliable set of matches, one can reverse the role of the two images, and repeat the previous procedure. Only the correspondences $\mathbf{m}_i \leftrightarrow \mathbf{m}'_j$ between points that chose each other are kept. For matching purposes we typically detect 500 interest points per image in 320×240 images. A good choice is often to use 7×7 correlation windows. This represents a reasonable trade-off between being large enough to be statistically significant and small enough to be able to ignore undesirable variations due to perspective, occlusions, or background changes. Zero-normalized cross-correlation makes the matching procedure very robust to illumination changes. In practice, we reject matches for which this measure is less than 0.8 as unreliable matches. We also limit the search of correspondents for a maximum image movement of 50 pixels. In terms of code optimization, [100] discusses some efficient implementations using MMX instructions for both point extraction and matching.

An alternative to matching points across images is to use the Kanade-Lucas-Tomasi (KLT) tracker [83, 129, 116], which extracts points from an initial image and then tracks them in the following images by mostly relying on the optical flow. Both approaches have their strengths: KLT handles continuity better and keeps tracking points that cannot be detected as interest points. By contrast, performing detection in every frame naturally handles the appearance and disappearance of interest points due to aspect changes and occlusions.

Since these phenomena are perennial causes of tracking failure, many authors, including ourselves, give preference to the detection-in-every-frame approaches. In essence, the interest points replace the patches of [107, 135] and serve much the same purpose [97, 40, 116, 119, 44, 21, 131, 76, 137]. The 3D coordinates of corresponding points can be obtained by back-projecting them to the 3D model. Alternatively, the transfer function that will be introduced in Subsection 4.4.4 can be used to avoid having to compute them explicitly. This results in implementations that are both robust and fast because extraction and matching can now be achieved in real-time on modern computers.

4.4.3 Pose Estimation by Tracking Planes

An alternative way [119] to use interest points is to track 3D planar structures as opposed to full 3D models. This choice is justified by the fact that it is a common special case that makes the 3D model acquisition problem trivial. For example, in man-made environments such as the one of Figure 4.5, the ground plane and one or more walls are often visible throughout the scene. Furthermore, the resulting method is efficient and precise.

The homography $\mathbf{H}_w^0$ that relates the tracked plane with its projection in the first image is estimated from several correspondences, which can be given by hand. The idea of [119] is to exploit the fact that the relation between two views of the same plane is also an homography. The relation between the plane and a frame of the sequence can be retrieved by chaining the homographies, and used to estimate the camera pose as shown in Subsection 2.3.4.

For each incoming frame, captured at time t, the homography $\mathbf{H}_{t-1}^t$ that maps the plane projection in frame at time $t-1$ to frame at time t is computed from interest point matches between these frames. The

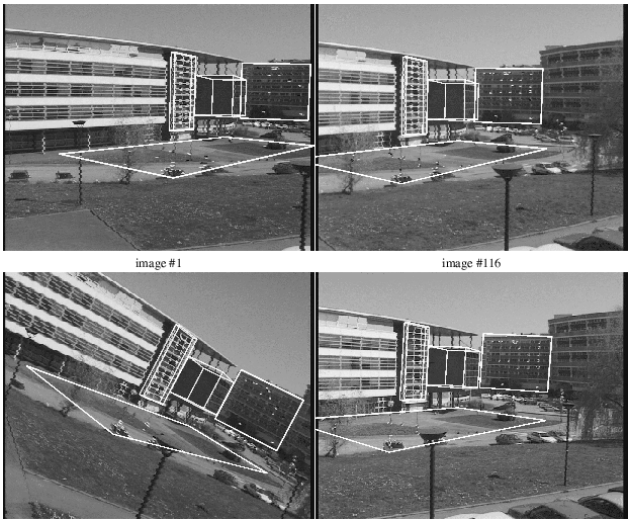


Fig. 4.5 Tracking planar structures using interest points. (From [118], figure courtesy of G. Simon and M.-O. Berger.)

computation is performed robustly using RANSAC and considering subsets of four matches to generate hypotheses on the actual homography as discussed in Subsection 2.5. Then, the homography $\mathbf{H}_w^t$ that maps the tracked plane to the new frame is obtained by chaining the successive transformations, which can be written as

$$\mathbf{H}_w^t = \mathbf{H}_{t-1}^t \mathbf{H}_{t-2}^{t-1} \dots \mathbf{H}_0^1 \mathbf{H}_w^0.$$

Here, the formulas were derived for one single plane, but this method has been extended to multiple planes in [118].

In this approach the jittering effect is minimal because the homographies between consecutive, close views can be computed very accurately. Nonetheless, because the motion is computed recursively by chaining transformations, one can expect error accumulation and drift after a while, even if this is delayed by the accuracy of the computed homographies.

4.4.4 Eliminating Drift

In the absence of points whose coordinates are known *a priori*, all methods are subject to error accumulation, which eventually results in tracking failure and precludes of truly long sequences.

A solution to this problem is to introduce one or more *keyframes* such as the one in the upper left corner of Figure 4.6, that is images

Fig. 4.6 Face tracking using interest points and one reference image shown on the top left. (From [137].)

of the target object or scene for which the camera has been registered *beforehand*. At runtime, incoming images can be matched against the keyframes to provide a position estimate that is drift-free [107, 44, 131]. This, however, is more difficult than matching against immediately preceding frames as the difference in viewpoint is likely to be much larger. The algorithm used to establish point correspondences must therefore both be fast and relatively insensitive to large perspective distortions, which is not usually the case for those used by the algorithms of Subsections 4.4.2 and 4.4.3 that need only handle small distortions between consecutive frames.

In [137], this is handled as follows. During a training stage, the system extracts interest points from each keyframe, back-projects them to the object surface to compute their 3D position, and stores image patches centered around their location. During tracking, for each new incoming image, the system picks the keyframe whose viewpoint is closest to that of the last known viewpoint. It synthesizes an *intermediate image* from that keyframe by warping the stored image patches to the last known viewpoint, which is typically the one corresponding to the previous image. The intermediate and the incoming images are now close enough that matching can be performed using simple, conventional, and fast correlation methods. Since the 3D position of keyframe interest has been precomputed, the pose can then be estimated by robustly minimizing the reprojection error of Equation (2.9). This approach handles perspective distortion, complex aspect changes, and self-occlusion. Furthermore, it is very efficient because it takes advantage of the large graphics capabilities of modern CPUs and GPUs.

However, as noticed by several authors [107, 21, 131, 137], matching only against keyframes does not, by itself, yield directly exploitable results. This has two main causes. First, wide-baseline matching as described in the previous paragraph, is inherently less accurate than the short-baseline matching involved in frame-to-frame tracking, which is compounded by the fact that the number of correspondences that can be established is usually less. Second, if the pose is computed for each frame independently, no temporal consistency is enforced and the recovered motion can appear to be jerky. If it were used as is by an Augmented Reality application, the virtual objects inserted in the scene

would appear to *jitter*, or to tremble, as opposed to remaining solidly attached to the scene.

Temporal consistency can be enforced by some dynamical smoothing using a motion model. Another way proposed in [137] is to combine the information provided by the keyframes, which provides robustness, with that coming from preceding frames, which enforces temporal consistency. This does not make any assumption on the camera motion and improves the accuracy of the recovered pose. It is still compatible with the use of dynamical smoothing that can be useful to in case where the pose estimation remains unstable, for example when the object is essentially fronto-parallel.

The tracking problem is reformulated in [137] in terms of bundle-adjustment. In theory, this could be done by minimizing a weighted sum of the reprojection errors computed both for the 3D keyframe interest points and for points $\mathbf{N}_i$ tracked from frame to frame, with respect to the camera poses up to time t, and to the 3D locations of the points N_i. In practice, this would be too time consuming and [137] restricts the estimation to the current and previous frames. The problem then becomes minimizing

$$\min_{\mathbf{P}^t, \mathbf{P}^{t-1}, \mathbf{N}_i} \left(r^t + r^{t-1} + \sum_i s_i^t \right), \tag{4.14}$$

with r^t and r^{t-1} being the total residuals of the points from the *keyframes* reprojected in frames at time t and $t-1$. In Equation (4.14),

$$s_i^t = \text{dist}^2(\mathbf{P}^{t-1}\tilde{\mathbf{N}}_i, \mathbf{n}_i^{t-1}) + \text{dist}^2(\mathbf{P}^t\tilde{\mathbf{N}}_i, \mathbf{n}_i^t)$$

is the residual for point $\mathbf{N}_i$ in frames $t-1$ and t, with the interest point $\mathbf{n}_i^t$ detected in the current frame is matched against the point $\mathbf{n}_i^{t-1}$ detected in the previous frame. This formulation would still result in a computationally intensive algorithm if the 3D coordinates of the $\mathbf{N}_i$ are treated as optimization variables. However, as shown in [114], one can exploit the fact that the $\mathbf{N}_i$ are on the surface of the 3D model and approximate the s_i^t terms using a transfer function that involves only the point projections. Given a point $\mathbf{n}$ in the first frame and the poses $\mathbf{P}$ and $\mathbf{P}'$ of the two frames, such a transfer function $\Psi(\mathbf{n}, \mathbf{P}, \mathbf{P}')$ returns the point $\mathbf{n}'$ such that there is a 3D point $\mathbf{N}$ belonging to the

model surface that satisfies $\tilde{\mathbf{n}} = \mathbf{P}\tilde{\mathbf{N}}$ and $\tilde{\mathbf{n}'} = \mathbf{P}'\tilde{\mathbf{N}}$. s_i^t can then be approximated as

$$\text{dist}^2\left(\Psi\left(\mathbf{n}_i^{t-1}, \mathbf{P}^{t-1}, \mathbf{P}^t\right), \mathbf{n}_i^t\right) + \text{dist}^2\left(\Psi\left(\mathbf{n}_i^t, \mathbf{P}^t, \mathbf{P}^{t-1}\right), \mathbf{n}_i^{t-1}\right), \quad (4.15)$$

where the actual 3D position $\mathbf{N}_i$ does not appear anymore. Finally, the combination of the information from wide baseline matching and from preceding frames in Equation (4.14) results in a real-time tracker that does not jitter or drift and can deal with significant aspect changes. In [136], this method has been extended so that it can also take advantage of edge information, still in real-time.

4.5 Tracking Without 3D Models

All the methods presented up to here estimate the pose given an *a priori* 3D model. However, it is possible to simultaneously estimate both camera motion and scene geometry, without any such model. The recovered trajectory and 3D structure are expressed in an arbitrary coordinate system, for example the one corresponding to the initial camera position. This problem is known as Simultaneous Localization and Mapping (SLAM) by roboticists who are mainly interested in the self-localization of a moving robot. By contrast to Structure from Motion (SFM) that focuses more on local reconstruction issues, SLAM is usually more about hand-off between local reconstructions and how to merge them to produce a global representation. This involves finding effective approximations to the huge fully connected covariance matrices that would result from a naive "all frames at once" batch bundle adjustment.

We present here two different classes of approaches that both rely on interest points.

4.5.1 n-Images Methods

The first class of approaches relies on projective properties that provide constraints on camera motion and 3D point locations from 2D correspondences. While such approaches have long been used for offline

camera registration in image sequences [130, 39, 103, 54], only recent improvements in both algorithms and available computational power have made them practical for real-time estimation.

For example, [100] shows how to recover in real-time the trajectory of a moving calibrated camera over a long period of time and with very little drift. The algorithm first estimates the relative poses between three consecutive frames from point correspondences established as described in Subsection 4.4.2. This is done by robustly estimating the essential matrix $\mathbf{E}$ between image pairs. The essential matrix is a 3×3 matrix that relates corresponding points $\mathbf{m}$ and $\mathbf{m}'$ in two images:

$$\mathbf{m}'^T \mathbf{K}^{-T} \mathbf{E} \mathbf{K}^{-1} \mathbf{m} = 0.$$

When the camera calibration matrix $\mathbf{K}$ is known, $\mathbf{E}$ can be computed from 5 point correspondences, and RANSAC is used to handle spurious matches. The relative motion between the two images can then be extracted from $\mathbf{E}$. Using two images only yields up to ten solutions and the third image is used to choose the correct one. Furthermore, the translation can be recovered up to a scale factor only. However, this relative motion must still be expressed in the same coordinate system as the previously recovered camera trajectory, which means keeping the scale consistent. To this end, the 2D point matches are linked into tracks over time and the tracks are then triangulated into 3D points using the estimated motion. The scale is then taken to be the one that best aligns these points against the current reconstruction.

As direct application of this approach would quickly in drift, two techniques are used in [100] to mitigate this problem. First, the pose is refined once in a while by minimizing the reprojection error of the already reconstructed 3D points over sets of frames. Second, the system is made to occasionally "forget" the 3D coordinates and to recompute them from scratch to avoid error accumulation.

This system has been tested with a vehicle-mounted camera and yields results very close to that of a GPS, even for trajectories of several hundreds of meters. In other words, error accumulation is not completely avoided, but considerably reduced.

4.5.2 Filter-Based Methods

Pose and structure can also be recursively estimated using the Extended Kalman filter [5, 12, 22, 63, 28] discussed in Subsection 2.6.1. In particular, [28] shows that it can yield very good results in real-time.

While [100] proposes a bottom-up approach – interest points are tracked in 2D then reconstructed to 3D, here the pose estimation is done in a top-down manner. The camera is supposed to move smoothly, with unlikely large accelerations. The filter state therefore contains the camera pose parameters, and the linear and angular velocities used to predict the camera pose over time. The filter state also contains the 3D locations of some points detected in the images. In each coming frame, the position of a feature point is predicted and its uncertainty is estimated using uncertainty propagation, using the 3D location stored in the filter state, the predicted camera pose and its uncertainty. This constraints the search for the point position in the current image, retrieved using sum-of-squared difference correlation. This position is then given to the Kalman filter to update the point 3D location. These hypotheses are tested in subsequent images by matching them against the images, and their probabilities are re-weighted

One issue is the initialization in the filter of appearing feature points, since the depth of such a point cannot be estimated from one measurement. [28] proposes to represent the initial probability density over point depth by a equally-weighted particle set. Discrete depth hypotheses are made along the semi-infinite line stated at the estimated camera position heading along the point viewing direction. The hypotheses are tested in the subsequent time steps by projecting them into the images, and re-weighted according to their likelihood. After some times, the distribution becomes closely Gaussian. A covariance matrix can then be enough to represent the distribution, and the feature point can be integrated into the filter.

To handle the distortion on the point appearances due to perspective, this method was extended in [95] to also estimate the orientation of the local surface. The orientation is initialized to be parallel to the current viewing direction. For each coming frame, the current orientation

estimate is used to predict the point appearance to help finding the point projection, and updated from the actual image.

4.6 Natural Features Methods in Short

Edge-based methods have a fairly low computational complexity because they only work for a small fraction of the image pixels. However, they can become confused in the presence of texture or of a cluttered background. In such cases, area-based methods come into their own and justify their increased computational requirements, which remain quite manageable on modern computers. Nevertheless, optical flow-based methods depend on brightness constancy assumption during optical flow computation, and major lighting changes can cause tracking failure. Because of the linearization in the optical flow equation cue, the range of acceptable speeds is also limited. The template-based approach is attractive because it has low computational requirements, and is simple to implement. But it has some disadvantages. It loses some of its elegance when occlusions must be taken into account, and handling illumination changes requires an offline stage where appearance variations are learned. The class of objects that can be tracked is also limited and a 3D object of general shape under general perspective view have never been handled in this way. All these drawbacks disappear when using a local, feature-based approach. Interest points give information similar to optical flow, but with no need for assumption on the brightness constancy or linearity assumptions. Computers have now become powerful enough to make them practical for real-time applications. As a result, they are now popular and yield the most successful 3D tracking techniques. As the next section shows, they are also a powerful tool for object detection in individual images.

5

Tracking by Detection

The recursive nature of traditional 3D tracking approaches provides a strong prior on the pose for each new frame and makes image feature identifications relatively easy. However, it comes at a price: First, the system must either be initialized by hand or require the camera to be very close to a specified position. Second, it makes the system very fragile. If something goes wrong between two consecutive frames, for example due to a complete occlusion of the target object or a very fast motion, the system can be lost and must be re-initialized in the same fashion. In practice, such weaknesses make purely recursive systems nearly unusable, and the popularity of ARToolKit [68] in the Augmented Reality community should come as no surprise: It is the first vision-based system to really overcome these limitations by being able to detect the markers in every frame without constraints on the camera pose.

However, achieving the same level of performance *without* having to engineer the environment remains a desirable goal. Pose estimation from natural features without prior on the actual position is closely related to object detection and recognition. Object detection has a long history in Computer Vision, mostly focused on 2D detection even for

3D objects [99, 138]. Nevertheless, there has been longstanding interest in simultaneous object detection and pose estimation. Early approaches were edge-based [78, 64], but methods based on feature points matching have become popular since [111] shows that local invariants work better than raw patches for such purpose. [111] uses invariants based on rotation invariant combination of image derivatives but other local invariants have been proposed. Considering feature point appear to be a better approach to achieve robustness to scale, viewpoint, illumination changes and partial occlusions than edge- or eigen-image- based techniques.

During an offline training stage, one builds a database of interest points lying on the object and whose position on the object surface can be computed. A few images in which the object has been manually registered are often used for this purpose. At runtime, feature points are first extracted from individual images and matched against the database. The object pose can then be estimated from such correspondences, for example using RANSAC to eliminate spurious correspondences.

The difficulty in implementing such approaches comes from the fact that the database images and the input ones may have been acquired from very different viewpoints. As discussed in Subsection 4.4.2, unless the motion is very quick, this problem does not arise in conventional recursive tracking approaches because the images are close to each other. However, for tracking-by-detection purposes, the so-called *wide baseline* matching problem becomes a critical issue that must be addressed.

In the remainder of this subsection, we discuss in more detail the extraction and matching of feature points in this context. We conclude by discussing the relative merits of tracking-by-detection and recursive tracking.

5.1 Feature Point Extraction

To handle as wide as possible a range of viewing conditions, feature point extraction should be insensitive to scale, viewpoint, and illumination changes.

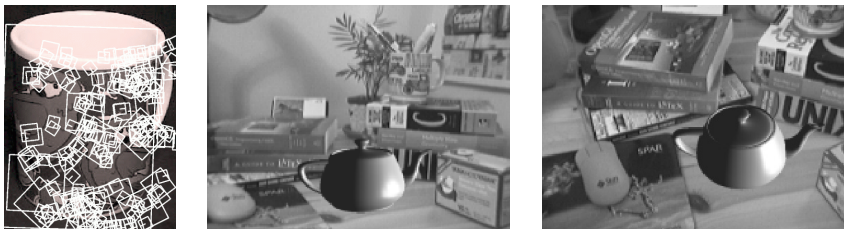
As proposed in [77], scale-invariant extraction can be achieved by taking feature points to be local extrema of a Laplacian-of-Gaussian pyramid in scale-space. To increase computational efficiency, the Laplacian can be approximated by a Difference-of-Gaussians [80]. Research has then focused on affine invariant region detection to handle more perspective changes. [11, 110, 91] used an affine invariant point detector based on the Harris detector, where the affine transformation that makes equal the two eigen values of the auto correlation matrix is evaluated to rectify the patch appearance. [134] achieves such invariance by fitting an ellipse to the local texture. [87] proposes a fast algorithm to extract Maximally Stable Extremal Regions demonstrated in a live demo. [92] gives a good summary and comparisons of the existing affine invariant regions detectors. The reader interested in experimenting with such methods can find some code or executable available for research purposes on the web pages of David Lowe and Krystian Mikolajczyk. Our own method [74] is also easy to reimplement, and the reader is encouraged to try it. Remember that all these approaches mostly depend on the object texture, and would work better on plain, textured, Lambertian target.

5.2 Wide Baseline Matching

Once a feature point has been extracted, the most popular approach to matching it is first to characterize it in terms of its image neighborhood and then to compare this characterization to those present in the database. Such characterization, or *local descriptor*, should be not only invariant to viewpoint and illumination changes but also highly distinctive. We briefly review some of the most representative below.

5.2.1 Local Descriptors

Many such descriptors have been proposed over the years. For example, [111] computes rotation invariant descriptors as functions of relatively high order image derivatives to achieve orientation invariance; [134] fits an ellipse to the texture around local intensity extrema and uses the Generalized Color Moments [93] as a descriptor. [82] introduces a



(a) (b) (c)

Fig. 5.1 Using SIFT for tracking-by-detection. (a) Detected SIFT features [81]. (b,c) They have been used to track the pose of the camera and add the virtual teapot [120]. (Courtesy of D.G. Lowe and I. Gordon).

descriptor called SIFT based on multiple orientation histograms, which tolerates significant local deformations. This last descriptor has been shown in [90] to be one of the most efficient. As illustrated by Figure 5.1, it has been successfully applied to 3D tracking in [113, 120] and we now describe it in more detail.

The remarkable invariance of the SIFT descriptor is achieved by a succession of carefully designed techniques. First the location and scale of the keypoints are determined precisely by interpolating the pyramid of Difference-of-Gaussians used for the detection. To achieve image rotation invariance, an orientation is also assigned to the keypoint. It is taken to be the one corresponding to a peak in the histogram of the gradient orientations within a region around the keypoint. This method is quite stable under viewpoint changes, and achieves an accuracy of a few degrees. The image neighborhood of the feature point is then corrected according to the estimated scale and orientation, and a local descriptor is computed on the resulting image region to achieve invariance to the remaining variations, such as illumination or out-of-plane variation. The point neighborhood is divided into several, typically 4×4, subregions and the contents of each subregion is summarized by an height-bin histogram of gradient orientations. The keypoint descriptor becomes a vector with 128 dimensions, built by concatenating the different histograms. Finally, this vector is normalized to unit length to reduce the effects of illumination changes.

5.2.2 Statistical Classification

The SIFT descriptor has been empirically shown to be both very distinctive and computationally cheaper than those based on filter banks. To shift even more of the computational burden from matching to training, which can be performed beforehand, we have proposed in our own work an alternative approach based on machine learning techniques [74]. We treat wide baseline matching of keypoints as a classification problem, in which each class corresponds to the set of all possible views of such a point. Given one or more images of a target object, the system synthesizes a large number of views, or image patches, of individual keypoints to automatically build the training set. If the object can be assumed to be locally planar, this is done by simply warping image patches around the points under affine deformations, otherwise, given the 3D model, standard Computer Graphics texture-mapping techniques can be used. This second approach relaxes the planarity assumptions.

The classification itself is performed using randomized trees [1]. Each non-terminal node of a tree contains a test of the type: "Is this pixel brighter than this one?" that splits the image space. Each leaf contains an estimate based on training data of the conditional distribution over the classes given that a patch reaches that leaf. A new image is classified by simply dropping it down the tree. Since only pixel intensities comparisons are involved, this procedure is very fast and robust to illumination changes. Thanks to the efficiency of randomized trees, it yields reliable classification results.

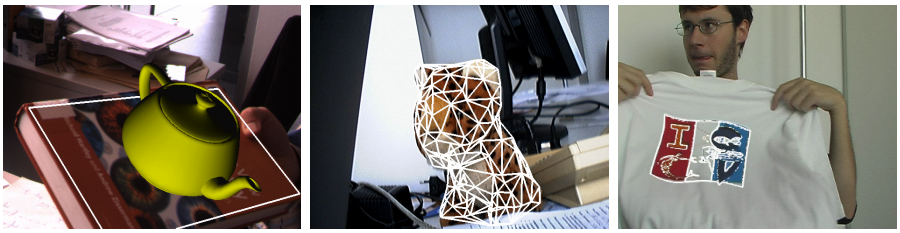


Fig. 5.2 Detection and computation in real-time of the 3D pose of a planar object, a full 3D object, and a deformable object. (From [74] and [102].)

As depicted by Figure 5.2, this method has been successfully used to detect and compute the 3D pose of planar, non-planar, and even deformable objects [74, 102].

5.3 From Wide Baseline Matching to 3D Tracking

As mentioned before, wide baseline matching techniques can be used to perform 3D tracking. To illustrate this, we briefly describe here the SIFT-based implementation reported in [120].

First, during a learning stage, a database of scene feature points is built by extracting SIFT keypoints in some reference images. Because the keypoints are detected in scale-space, the scene does not necessarily have to be well-textured. Their 3D positions are recovered using a structure-from-motion algorithm. Two-view correspondences are first established based on the SIFT descriptors, and chained to construct multi-view correspondences while avoiding prohibitive complexity. Then the 3D positions are recovered by a global optimization over all camera parameters and these point coordinates, which is initialized as suggested in [127].

At run-time, SIFT features are extracted from the current frame, matched against the database, resulting in a set of 2D / 3D correspondences. The camera pose can be recovered using RANSAC and a P3P algorithm, as described in Subsection 2.3.3.

The best candidate match for a SIFT feature extracted from the current frame is assumed to be its nearest neighbor, in the sense of the Euclidean distance of the descriptor vectors, in the point database. The size of the database and the high dimensionality of these vectors would make the exhaustive search intractable, especially for real-time applications. To allow for fast search, the database is organized as a k-d tree. The search is performed so that bins are explored in the order of their closest distance from the query description vector, and stopped after a given number of data points has been considered, as described in [13]. In practice, this approach returns the actual nearest neighbor with high probability.

Because the performance of RANSAC degrades rapidly when the percentage of outliers in the set of 2D / 3D correspondences increases,

the query point is matched only if it is close enough to its nearest-neighbor. This greatly reduces the number of false matches that may result from cluttered background. A good idea at this point is that matches based on feature point recognition should be refined by a local image-patch based search for improved matching accuracy before being used for tracking.

As discussed Subsection 4.4.4, recovering the camera positions in each frame independently and from noisy data typically results in jitter. To stabilize the pose, a regularization term that smoothes camera motion across consecutive frames is introduced. Its weight is iteratively estimated to eliminate as much jitter as possible without introducing drift when the motion is fast. The full method runs at four frames per second on a 1.8 GHz ThinkPad.

5.4 The End of Recursive Tracking?

Since real-time tracking-by-detection has become a practical possibility, one must wonder if the conventional recursive tracking methods that have been presented in the previous subsections of this survey are obsolescent.

We do not believe this to be the case. As illustrated by the case of the SIFT-based tracking system [120] discussed above, treating each frame independently has its problems. Imposing temporal continuity constraints across frames can help increase the robustness and quality of the results. Furthermore, wide baseline matching tends to be both less accurate and more computationally intensive than the short baseline variety.

As shown in Subsection 4.4.4, combining both kinds of approaches can yield the best of both worlds: Robustness from tracking-by-detection, and accuracy from recursive tracking. In our opinion, this is where the future of tracking lies. The challenge will be to become able, perhaps by taking advantage of recursive techniques that do not require prior training, to learn object descriptions online so that a tracker can operate in a complex environment with minimal *a priori* knowledge.

6

Conclusion

To conclude this survey, we first attempt to offer some practical advice to the reader who may be wondering what method to use in a given situation. We then discuss what we see as the future of the 3D tracking research.

6.1 Choosing the Appropriate Approach to 3D Tracking

Such a choice crucially depends on the target application and the environment in which it is expected to work. Table 6.1 summarizes the possibilities that we discuss in more detail below.

Even after more than twenty years of research, practical vision-based 3–D tracking systems still rely on fiducials because this remains the only approach that is sufficiently fast, robust, and accurate. Therefore, if it is practical to introduce them in the environment the system inhabits, this solution surely must be retained. If expense is not a major issue, commercial products such as the ones proposed by the Advanced Real-time Tracking or Geodetic services, Inc., Advanced Real-time Tracking GmbH, Metronor, ViconPeak, AICON 3D Systems GmbH companies provide the accuracy, reliability, and speed

Section	**Selected references**	**Rely on**	**Suitable when**	**Manual Initia-lisation**	**Accuracy**	**Failure modes**	**3D Model Required**
3.1	Commercial products	Retro-reflective fiducials, infrared cameras	Use of fiducials acceptable; Expense not a primary concern	No	Highly accurate	Very rare	Can be reconstructed by the system
3.2	[3]	Planar fiducials	Use of fiducials acceptable	No	Accurate	Hidden fiducials	No
4.1	[35]	Edges	Strong edges, simple background	Yes	Can jitter	Very fast motion Cluttered background	Yes
4.3	[66]	Template Matching	Small planar objects	Yes Yes	Highly accurate	Occlusion Very fast motion	No
4.4.3	[119]	Interest Points	Planar textured objects	Yes	Can drift	Very fast motion	No
4.4.4	[137]	Interest Points	3D textured objects	Yes	Avoid drift by using keyframes, reduce jitter	Very fast motion	Yes
4.5.1	[100, 28]	Interest Points	3D textured scenes	No	Limited drift	Very fast motion	No

Table 6.1 Selected 3D tracking methods that currently run at frame-rate.

required by industrial or medical applications. Their drawback is that they require retro-reflective spheres and infrared cameras, which makes their deployment relatively expensive and cumbersome. ARToolkit [3] is a freely available alternative that uses planar fiducials that may be printed on pieces of paper. While less accurate, it remains robust and allows for fast development of low-cost applications. As a result, it has become popular in the Augmented Reality Community.

However, this state of affairs may be about to change as computers have just now become fast enough to reliably handle natural features in real-time, thereby making it possible to completely do away with fiducials. This is especially true when dealing with objects that are polygonal, textured, or both.

Polygonal Objects For objects that have strong contours and are silhouetted against relatively simple backgrounds, the RAPiD-like methods of Subsection 4.1 are a good place to start. They give good results while being fast and relatively simple to implement. By relying on fast optimization techniques and on a fast and reliable top-down feature extraction, it is possible to process images at more then 50 Hz on a modern PC [69]. They are also naturally robust to light and scales changes, and specular effects. They run very fast even on older, slower computers. As a result, they have actually been used for visual servoing in industrial environments where reliable edges can be found.

Note, however, that these trackers are prone to catastrophic failures resulting in complete loss of track. This is particularly true when the background becomes cluttered or when an aspect change occurs, making it easy to confuse a true object edge with another one. This can be minimized by careful implementation but not completely eliminated. Note that such failures are more frequent when tracking simple objects, such as a rectangular box, than more complex ones. This makes sense because the latter provide a richer sampling of the set of possible orientations. Therefore, if some of the data is corrupted by noise, it is usually easier for a robust estimator to ignore it and use the rest.

Textured Objects If the target object is textured, such image cue can replace, or complement, the contour information.

If the target object is planar and occlusions are unlikely to occur, template-based methods such as those discussed in Subsection 4.3.3 have been reported [66] to perform very accurately in the presence of an agile motion. This method was reported to take less than 10 ms on a by now old computer (an O2 Silicon Graphics workstation with a 150 MHz R5000 processor). It is unfortunately very sensitive to occlusions and hard to extend to fully 3–Dimensional objects.

By contrast, the interest point-based approaches discussed in Subsections 4.4.4 and 4.4.3 do not suffer from these limitations, at least when the scene is sufficiently textured. However, they entail both a much more involved implementation and a larger computational burden. They entail both a much more involved implementation and a larger computational burden. They run at frame-rate on modern PCs but still require the full CPU to do so. We therefore expect to have to wait for a few more years to have them use only a fraction of the available CPU, as contour-based methods already do. We nevertheless believe the wait to be worthwhile because these methods can be made to be very robust to partial occlusions as well as to lighting, aspect, and background changes. Among the many existing methods, the choice can be made on the basis of the available models for the target object. In the absence of such a model, one can rely on a fully bottom-up approach [28, 100] or on one that treats the scene as a set of planar patches [119], but with no explicit 3–D model management. [100] reports a 13 frame-per-second on a 1 GHz computer. [28] reaches 30 frame-per-second rate on a Pentium M 1.6 GHz laptop, by considering fewer but stronger interest points.

If a full 3–D model is available, or even required as is usually the case for AR applications, it can be effectively used to eliminate jitter and ensure accurate reprojection into the images [137]. Furthermore, if a few keyframes – that is, views of the target object for which the pose known – can be created during a training phase, they can be used in conjunction with the 3–D model to complement the information provided by matching against previous frames. This results in an algorithm that does not drift and can recover from tracking failures [137]. It runs at 20 frame-per-second on a 2.8 GHz.

6.2 Implementation Issues

In their basic principles, most of the approaches discussed here are fairly easy to understand. However, their performance is often critically connected to the quality of the implementation. Those implementation details are often buried towards the end of the papers, if mentioned at all. To give the reader a flavor of what to look for, we give here a short and non-exhaustive list of such details.

For all these methods, while a rough calibration of the internal parameters camera can suffice for reasonable results, an accurate, carefully performed calibration can yield to surprising improvements. A wide-angle camera generally helps by providing more constraints. As shown Subsection 2.1.5, the distortions can easily be corrected. When possible, one can also use new omnidirectional cameras [46, 126] that provide even more image constraints and improve both accuracy and robustness.

6.2.1 Contour Based Methods

As discussed in [85], it seems preferable to sample the projected contours rather than the 3–D model contours. The use of first order Gaussian kernels also helps a lot to find the corresponding image contour, even with a mask that is kept as small as 3×3 or 5×5 to preserve accuracy and the computation speed. When looking along a scan-line for an image edge in cases where several may be found, the criterion used to pick the best has to balance gradient-strength against proximity to the previous position in a way that prevents fast divergence of the tracker. One such way is to endow the algorithm with the ability to consider several hypotheses simultaneously [136].

6.2.2 Interest Point Based Methods

As discussed in [54], in an effective DLT-style approach to estimating a homography between images, the point coordinates should be normalized to improve the accuracy. The Harris detector [53, 116] is a valid choice for point detection. However, to reduce the risk of producing

ill-conditioned configuration, the points should be well spread on the visible surface of the object. An effective way to do this is to first divide the input images into subregions, and then to apply an adaptive threshold per subregion. This ensures that some points will be detected in low textured regions while reducing the number of points detected in highly textured ones. Once detected, the points could be tracked using the Kanade-Lucas-Tomasi tracker [129, 116], for which an implementation can be found in the OpenCv library [60] and on the website of Stan Birchfield [14]. However, in our experience, it is better to detect and match points in successive images, as described in Subsections 4.4.1 and 4.4.2. It seems to be less prone to drift, and allows for aspect changes.

6.3 The Future of 3D Tracking

Even though 3D tracking algorithms are on the verge of becoming practical without requiring fiducials, the reader must be aware that the recursive nature of most of these algorithms makes them inherently fragile: They must be initialized manually and cannot recover if the process fails for any reason. In practice, even the best methods suffer such failures all too often, for example because the motion is too fast, a complete occlusion occurs, or simply because the target object moves momentarily out of the field of view.

This can be addressed by combining image data with dynamics data provided by inertial sensors or gyroscopes [123, 41]. The sensors allow a prediction of the camera position or relative motion that can then be refined using vision techniques similar to the ones described in this survey. Such combination is possible for applications, such as Augmented Reality, that require tracking of the camera with respect to a static scene, assuming one is willing to instrument the camera. However, instrumenting the camera is not always an option. For example, it would be of no use to track moving cars with a static camera.

A more generic and desirable approach is therefore to develop purely image-based methods that can detect the target object and compute its 3D pose from a single image. If they are fast enough, they can then be used to initialize and re-initialize the system as often as needed,

even if they cannot provide the same accuracy as traditional recursive approaches that use temporal continuity constraints to refine their estimates. Techniques able to do just this are just beginning to come online. And, since they are the last missing part of the puzzle, we expect that we will not have to wait for another twenty years for purely vision-based commercial systems to become a reality.

References

[1] Y. Amit and D. Geman, "Shape quantization and recognition with randomized trees," *Neural Computation*, vol. 9, no. 7, pp. 1545–1588, 1997.

[2] M. Armstrong and A. Zisserman, "Robust object tracking," in *Proceedings of Asian Conference on Computer Vision*, pp. 58–62, 1995.

[3] ARToolKit http://www.hitl.washington.edu/artoolkit/.

[4] K. Arun, T. Huang, and S. Blostein, "Least-squares fitting of two 3-D points sets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, no. 5, pp. 698–700, 1987.

[5] A. Azarbayejani and A. P. Pentland, "Recursive estimation of motion, structure and focal length," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 6, pp. 562–575, 1995.

[6] R. Azuma, "A survey of augmented reality," in *Computer Graphics, SIGGRAPH Proceedings*, pp. 1–8, August 1995.

[7] S. Baker and I. Matthews, "Lucas-Kanade 20 years on: A unifying framework," *International Journal of Computer Vision*, pp. 221–255, March 2004.

[8] S. Balcisoy, M. Kallmann, R. Torre, P. Fua, and D. Thalmann, "Interaction techniques with virtual humans in mixed environment," in *International Symposium on Mixed Reality* (Yokohama, Japan), March 2001.

[9] Y. Bar-Shalom and T. Fortmann, *Tracking and Data Association*. Academic Press, 1988.

[10] S. Basu, I. Essa, and A. Pentland, "Motion regularization for model-based head tracking," in *International Conference on Pattern Recognition* (Vienna, Austria), 1996.

[11] A. Baumberg, "Reliable feature matching across widely separated views," in *Conference on Computer Vision and Pattern Recognition*, pp. 774–781, 2000.

[12] P. A. Beardsley, A. Zisserman, and D. W. Murray, "Sequential update of projective and affine structure from motion," *International Journal of Computer Vision*, vol. 23, no. 3, pp. 235–259, 1997.

[13] J. Beis and D. Lowe, "Shape indexing using approximate nearest-neighbour search in high-dimensional spaces," in *Conference on Computer Vision and Pattern Recognition* (Puerto Rico), pp. 1000–1006, 1997.

[14] S. Birchfield, "KLT: An Implementation of the Kanade-Lucas-Tomasi Feature Tracker," http://www.ces.clemson.edu/~stb/klt/.

[15] M. J. Black and Y. Yacoob, "Recognizing facial expressions in image sequences using local parameterized models of image motion," *International Journal of Computer Vision*, pp. 23–48, 1997.

[16] A. Blake and M. Isard, *Active Contours*. Springer-Verlag, 1998.

[17] D. Brown, "Close range camera calibration," *Photogrammetric Engineering*, vol. 37, no. 8, pp. 855–866, 1971.

[18] R. Brown and P. Hwang, *Introduction to Random Signals and Applied Kalman Filtering, Second Edition*. John Wiley & Sons, Inc., 1992.

[19] M. Cascia, S. Sclaroff, and V. Athitsos, "Fast, reliable head tracking under varying illumination: An approach based on registration of texture-mapped 3D models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, April 2000.

[20] T.-J. Cham and J. Rehg, "A multiple hypothesis approach to figure tracking," in *Conference on Computer Vision and Pattern Recognition* (Ft. Collins, CO), June 1999.

[21] K. Chia, A. Cheok, and S. Prince, "Online 6 DOF augmented reality registration from natural features," in *International Symposium on Mixed and Augmented Reality*, 2002.

[22] A. Chiuso, P. Favaro, H. Jin, and S. Soatto, "Structure from motion causally integrated over time," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 4, pp. 523–535, 2002.

[23] Y. Cho, W. Lee, and U. Neumann, "A multi-ring color fiducial system and intensity-invariant detection method for scalable fiducial-tracking augmented reality," in *International Workshop on Augmented Reality*, 1998.

[24] D. Claus and A. Fitzgibbon, "Reliable fiducial detection in natural scenes," in *European Conference on Computer Vision*, pp. 469–480, Springer-Verlag, May 2004.

[25] L. Cohen and I. Cohen, "Finite-element methods for active contour models and balloons for 2-D and 3-D images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, pp. 1131–1147, November 1993.

[26] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, pp. 564–575, 2003.

[27] A. Comport, E. Marchand, and F. Chaumette, "A real-time tracker for markerless augmented reality," in *International Symposium on Mixed and Augmented Reality* (Tokyo, Japan), September 2003.

[28] A. Davison, "Real-time simultaneous localisation and mapping with a single camera," in *Proceedings of International Conference on Computer Vision*, pp. 1403–1410, October 2003.

[29] D. DeCarlo and D. Metaxas, "Optical flow constraints on deformable models with applications to face tracking," *International Journal of Computer Vision*, vol. 38, pp. 99–127, 2000.

[30] F. Dellaert and R. Collins, "Fast image-based tracking by selective pixel integration," in *ICCV Workshop of Frame-Rate Vision*, pp. 1–22, 1999.

[31] D. DeMenthon and L. S. Davis, "Model-based object pose in 25 lines of code," *International Journal of Computer Vision*, vol. 15, pp. 123–141, 1995.

[32] R. Deriche and O. Faugeras, "Tracking line segments," *Image and Vision Computing*, vol. 8, pp. 271–270, 1990.

[33] R. Deriche and G. Giraudon, "A Computational approach for corner and vertex detection," *International Journal of Computer Vision*, vol. 10, no. 2, pp. 101–124, 1993.

[34] A. Doucet, N. de Freitas, and N. Gordon, eds., *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, 2001.

[35] T. Drummond and R. Cipolla, "Real-time visual tracking of complex structures," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, pp. 932–946, July 2002.

[36] R. Evans, "Filtering of pose estimates genrated by the RAPiD tracker in applications," in *British Machine Vision Conference* (Oxford), pp. 79–84, 1990.

[37] O. Faugeras, *Three-Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, 1993.

[38] M. Fischler and R. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[39] A. Fitzgibbon and A. Zisserman, "Automatic camera recovery for closed or open image sequences," in *European Conference on Computer Vision* (Freiburg, Germany), pp. 311–326, June 1998.

[40] W. Förstner, "A feature-based correspondence algorithm for image matching," *International Archives of Photogrammetry and Remote Sensing*, vol. 26, no. 3, pp. 150–166, 1986.

[41] E. Foxlin and L. Naimark, "Miniaturization, claibration and accuracy evaluation of a hybrid self-tracker," in *International Symposium on Mixed and Augmented Reality* (Tokyo, Japan), 2003.

[42] J. Fryer and D. Goodin, "In-flight aerial camera calibration from photography of linear features," *Photogrammetric Engineering and Remote Sensing*, vol. 55, no. 12, pp. 1751–1754, 1989.

[43] D. Gavrila, "The visual analysis of human movement: A survey," *Computer Vision and Image Understanding*, vol. 73, January 1999.

[44] Y. Genc, S. Riedel, F. Souvannavong, and N. Navab, "Marker-less tracking for augmented reality: A learning-based approach," in *International Symposium on Mixed and Augmented Reality*, 2002.

[45] D. Gennery, "Visual tracking of known three-dimensional objects," *International Journal of Computer Vision*, vol. 7, no. 1, pp. 243–270, 1992.
[46] C. Geyer and K. Daniilidis, "Omnidirectional video," *The Visual Computer*, vol. 19, pp. 405–416, October 2003.
[47] F. Grassia, "Practical parameterization of rotations using the exponential map," *Journal of Graphics Tools*, vol. 3, no. 3, pp. 29–48, 1998.
[48] A. Gruen and T. Huang, eds., *Calibration and Orientation of Cameras in Computer Vision*. Vol. 34 of *Springer Series in Information Sciences*, Springer-Verlag, 2001.
[49] M. Haag and H.-M. Nagel, "Combination of edge element and optical flow estimates for 3-D model-based vehicle tracking intraffic image sequences," *International Journal of Computer Vision*, vol. 35, no. 3, pp. 295–319, 1999.
[50] G. Hager and P. Belhumeur, "Efficient region tracking with parametric models of geometry and illumination," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 10, pp. 1025–1039, 1998.
[51] R. Haralick, D. Lee, K. Ottenburg, and M. Nolle, "Analysis and solutions of the three point perspective pose estimation problem," in *Conference on Computer Vision and Pattern Recognition*, pp. 592–598, June 1991.
[52] C. Harris, *Tracking with Rigid Objects*. MIT Press, 1992.
[53] C. Harris and M. Stephens, "A combined corner and edge detector," in *Fourth Alvey Vision Conference*, Manchester, 1988.
[54] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.
[55] J. Heikkila and O. Silven, "A four-step camera calibration procedure with implicit image correction," in *Conference on Computer Vision and Pattern Recognition*, pp. 1106–1112, 1997.
[56] W. A. Hoff, K. Nguyen, and T. Lyon, "Computer vision-based registration techniques for augmented reality," in *Proceedings of Intelligent Robots and Control Systems XV, Intelligent Control Systems and Advanced Manufacturing*, pp. 538–548, November 1996.
[57] B. Horn, "Closed-form solution of absolute orientation using unit quaternions," *Journal of the Optical Society of America*, vol. 4, pp. 629–642, April 1987.
[58] B. Horn, H. Hilden, and S. Negahdaripour, "Closed-form solution of absolute orientation using orthonormal matrices," *Journal of the Optical Society of America*, vol. 5, no. 7, pp. 1127–1135, 1987.
[59] P. Huber, *Robust Statistics*. New York: Wiley, 1981.
[60] Intel, "Open Source Computer Vision Library," http://www.intel.com/research/mrl/research/opencv/.
[61] M. Isard and A. Blake, "A smoothing filter for Condensation," in *European Conference on Computer Vision*, pp. 767–781, 1998.
[62] A. Jepson, D. J. Fleet, and T. El-Maraghi, "Robust on-line appearance models for vision tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 10, pp. 1296–1311, 2003.
[63] H. Jin, P. Favaro, and S. Soatto, "A semi-direct approach to structure from motion," *The Visual Computer*, vol. 19, pp. 1–18, 2003.

[64] F. Jurie, "Tracking objects with a recognition algorithm," *Pattern Recognition Letters*, vol. 3–4, no. 19, pp. 331–340, 1998.

[65] F. Jurie and M. Dhome, "A simple and efficient template matching algorithm," in *International Conference on Computer Vision* (Vancouver, Canada), July 2001.

[66] F. Jurie and M. Dhome, "Hyperplane approximation for template matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, pp. 996–100, July 2002.

[67] H. Kato and M. Billinghurst, "Marker racking and HMD Calibration for a video-based augmented reality conferencing system," in *IEEE and ACM International Workshop on Augmented Reality*, October 1999.

[68] H. Kato, M. Billinghurst, I. Poupyrev, K. Imamoto, and K. Tachibana, "Virtual object manipulation on a table-top AR environment," in *International Symposium on Augmented Reality*, pp. 111–119, 2000.

[69] G. Klein and T. Drummond, "Robust visual tracking for non-instrumented augmented reality," in *International Symposium on Mixed and Augmented Reality*, pp. 36–45, October 2003.

[70] D. Koller, K. Daniilidis, and H.-H. Nagel, "Model-based object tracking in monocular image sequences of road traffic scenes," *International Journal of Computer Vision*, vol. 10, pp. 257–281, July 1993.

[71] D. Koller, G. Klinker, E. Rose, D. Breen, R. Whitaker, and M. Tuceryan, "Real-time vision-based camera tracking for augmented reality applications," in *ACM Symposium on Virtual Reality Software and Technology* (Lausanne, Switzerland), pp. 87–94, September 1997.

[72] H. Kollnig and H.-H. Nagel, "3D pose estimation by directly matching polyhedral models to gray value gradients," *International Journal of Computer Vision*, vol. 23, pp. 283–302, July 1997.

[73] A. Kosaka and G. Nakazawa, "Vision-based motion tracking of rigid objects using prediction of uncertainties," in *International Conference on Robotics and Automation* (Nagoya, Japan), pp. 2637–2644, 1995.

[74] V. Lepetit, P. Lagger, and P. Fua, "Randomized trees for real-time keypoint recognition," in *Conference on Computer Vision and Pattern Recognition* (San Diego, CA), June 2005.

[75] H. Li, P. Roivainen, and R. Forchheimer, "3-D motion estimation in model-based facial image coding," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, pp. 545–555, June 1993.

[76] J. Li, R. Laganiere, and G. Roth, "Online estimation of trifocal tensors for augmenting live video," in *International Symposium on Mixed and Augmented Reality*, pp. 182–190, 2004.

[77] T. Lindeberg, "Scale-space theory: A basic tool for analysing structures at different scales," *Journal of Applied Statistics*, vol. 21, no. 2, pp. 224–270, 1994.

[78] D. G. Lowe, "Fitting parameterized three-dimensional models to images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, pp. 441–450, June 1991.

[79] D. G. Lowe, "Robust model-based motion tracking through the integration of search and estimation," *International Journal of Computer Vision*, vol. 8, no. 2, pp. 113–122, 1992.

[80] D. Lowe, "Object recognition from local scale-invariant features," in *International Conference on Computer Vision*, pp. 1150–1157, 1999.

[81] D. Lowe, "Local feature view clustering for 3D object recognition," in *Conference on Computer Vision and Pattern Recognition*, pp. 682–688, 2001.

[82] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 20, no. 2, pp. 91–110, 2004.

[83] B. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *International Joint Conference on Artificial Intelligence*, pp. 674–679, 1981.

[84] E. Marchand, P. Bouthemy, and F. Chaumette, "A 2D-3D model-based approach to real-time visual tracking," *Image and Vision Computing*, vol. 19, no. 13, pp. 941–955, 2001.

[85] E. Marchand, P. Bouthemy, F. Chaumette, and V. Moreau, "Robust real-time visual tracking using a 2D-3D model-based approach," in *International Conference on Computer Vision* (Corfu, Greece), pp. 262–268, September 1999.

[86] E. Marchand and F. Chaumette, "Virtual visual servoing: A framework for real-time augmented reality," in *Computer Graphics Forum, Eurographics* (Sarrebruck, Allemagne), pp. 289–298, September 2002.

[87] J. Matas, O. Chum, U. Martin, and T. Pajdla, "Robust wide baseline stereo from maximally stable extremal regions," in *British Machine Vision Conference* (London, UK), pp. 384–393, September 2002.

[88] C. McGlove, E. Mikhail, and J. Bethel, eds., *Manual of Photogrametry*. American Society for Photogrammetry and Remote Sensing, Fifth ed., 2004.

[89] D. Metaxas and D. Terzopoulos, "Shape and nonrigid motion estimation through physics-based synthesis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 6, pp. 580–591, 1991.

[90] K. Mikolajczik and C. Schmid, "A performance evaluation of local descriptors," in *Conference on Computer Vision and Pattern Recognition*, pp. 257–263, June 2003.

[91] K. Mikolajczyk and C. Schmid, "An affine invariant interest point detector," in *European Conference on Computer Vision*, pp. 128–142, Springer, 2002. Copenhagen.

[92] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. V. Gool, "A comparison of affine region detectors," *Accepted to International Journal of Computer Vision*, 2005.

[93] F. Mindru, T. Moons, and L. VanGool, "Recognizing color patterns irrespective of viewpoint and illumination," in *Conference on Computer Vision and Pattern Recognition*, pp. 368–373, 1999.

[94] M. Moegring, C. Lessig, and O. Bimber, "Video see-through AR on consumer cell phones," in *International Symposium in Mixed and Augmented Reality*, pp. 252–253, 2004.

[95] N. Molton, A. Davison, and I. Reid, "Locally planar patch features for real-time structure from motion," in *British Machine Vision Conference*, September 2004.

[96] H. Moravec, *Robot Rover Visual Navigation*. Ann Arbor, Michigan: UMI Research Press, 1981.

[97] H. Moravec, "Towards automatic visual obstacle avoidance," in *International Joint Conference on Artificial Intelligence* (MIT, Cambridge, Mass.), p. 584, August 1977.

[98] L. Naimark and E. Foxlin, "Circular data matrix fiducial system and robust image processing for a wearable vision-interial self-tracker," in *International Symposium on Mixed and Augmented Reality* (Darmstadt, Germany), 2002.

[99] S. K. Nayar, S. A. Nene, and H. Murase, "Real-time 100 object recognition system," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 12, pp. 1186–1198, 1996.

[100] D. Nister, O. Naroditsky, and J. Bergen, "Visual odometry," in *Conference on Computer Vision and Pattern Recognition*, pp. 652–659, June 2004.

[101] D. Oberkampf, D. DeMenthon, and L. Davis, "Iterative pose estimation using coplanar feature points," *Computer Vision and Image Understanding*, vol. 63, pp. 495–511, 1996.

[102] J. Pilet, V. Lepetit, and P. Fua, "Real-time non-rigid surface detection," in *Conference on Computer Vision and Pattern Recognition* (San Diego, CA), June 2005.

[103] M. Pollefeys, R. Koch, and L. VanGool, "Self-calibration and metric reconstruction in spite of varying and unknown internal camera parameters," in *International Conference on Computer Vision*, 1998.

[104] L. Quan and Z. Lan, "Linear n-point camera pose determination," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, pp. 774–780, July 1999.

[105] A. Rahimi and T. Darrell, "Location estimation with a differential update network," in *Neural Information Processing Systems*, pp. 1049–1056, 2002.

[106] A. Rahimi, L.-P. Morency, and T. Darrell, "Reducing drift in parametric motion tracking," in *International Conference on Computer Vision*, pp. 315–322, 2001.

[107] S. Ravela, B. Draper, J. Lim, and R. Weiss, "Adaptive tracking and model registration across distinct aspects," in *International Conference on Intelligent Robots and Systems*, pp. 174–180, 1995.

[108] J. Rekimoto, "Matrix: A realtime object identification and registration method for augmented reality," in *Asia Pacific Computer Human Interaction*, 1998.

[109] A. Ruf, M. Tonko, R. Horaud, and H.-H. Nagel, "Visual tracking by adaptive kinematic prediction," in *Proceedings of International Conference on Intelligent Robots and Systems*, pp. 893–898, September 1997.

[110] F. Schaffalitzky and A. Zisserman, "Multi-view matching for unordered image sets, or "How do I organize my holiday snaps?"," in *Proceedings of European Conference on Computer Vision*, pp. 414–431, 2002.

[111] C. Schmid and R. Mohr, "Local grayvalue invariants for image retrieval," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, pp. 530–534, May 1997.

[112] S. Sclaroff and J. Isidoro, "Active blobs," in *International Conference on Computer Vision*, pp. 1146–1153, 1998.

[113] S. Se, D. G. Lowe, and J. Little, "Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks," *International Journal of Robotics Research*, vol. 22, no. 8, pp. 735–758, 2002.

[114] Y. Shan, Z. Liu, and Z. Zhang, "Model-based bundle adjustment with application to face modeling," in *International Conference on Computer Vision* (Vancouver, Canada), July 2001.

[115] J. Sherrah and S. Gong, "Fusion of 2D face alignment and 3D head pose estimation for robust and real-time performance," in *Proceedings of IEEE International Workshop on Recognition, Analysis and Tracking of Faces and Gestures in Real-Time Systems*, September 1999.

[116] J. Shi and C. Tomasi, "Good features to track," in *Conference on Computer Vision and Pattern Recognition* (Seattle), June 1994.

[117] G. Simon and M.-O. Berger, "A two-stage robust statistical method for temporal registration from features of various type," in *International Conference on Computer Vision* (Bombay, India), pp. 261–266, January 1998.

[118] G. Simon and M.-O. Berger, "Pose estimation from planar structures," *Computer Graphics and Applications*, vol. 22, pp. 46–53, November 2002.

[119] G. Simon, A. Fitzgibbon, and A. Zisserman, "Markerless tracking using planar structures in the scene," in *International Symposium on Mixed and Augmented Reality*, pp. 120–128, October 2000.

[120] I. Skrypnyk and D. G. Lowe, "Scene modelling, recognition and tracking with invariant image features," in *International Symposium on Mixed and Augmented Reality* (Arlington, VA), pp. 110–119, November 2004.

[121] C. Sminchisescu and B. Triggs, "Estimating articulated human motion with covariance scaled sampling," *International Journal of Robotics Research*, vol. 22, no. 6, pp. 371–393, 2003.

[122] S. M. Smith and J. M. Brady, "SUSAN – A new approach to low level image processing," Tech. Rep. TR95SMS1c, Oxford University, Chertsey, Surrey, UK, 1995.

[123] A. State, G. Hirota, D. Chen, W. Garett, and M. Livingston, "Superior augmented reality registration by integrating landmark tracking and magnetic tracking," *Computer Graphics, SIGGRAPH Proceedings,*, pp. 429–438, July 1996.

[124] P. Sturm and S. Maybank, "On plane-based camera calibration: A general algorithm, singularities, applications," in *Conference on Computer Vision and Pattern Recognition*, pp. 432–437, June 1999.

[125] I. Sutherland, "Sketchpad: A Man Machine Graphical Communications System," Technical Report 296, MIT Lincoln Laboratories, 1963.

[126] R. Swaminathan and S. K. Nayar, "A perspective on distortions," in *Conference on Computer Vision and Pattern Recognition*, June 2003.

[127] R. Szeliski and S. Kang, "Recovering 3-D shape and motion from image streams using non linear least squares," *Journal of Visual Communication and Image Representation*, vol. 5, no. 1, pp. 10–28, 1994.

[128] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust monte carlo localization for mobile robots," *Artificial Intelligence Journal*, vol. 128, no. 1–2, pp. 99–141, 2000.

[129] C. Tomasi and T. Kanade, "Detection and Tracking of Point Features," Technical Report CMU-CS-91-132, Carnegie Mellon University, April 1991.

[130] C. Tomasi and T. Kanade, "Shape and motion from image streams under orthography: A factorization method," *International Journal of Computer Vision*, vol. 9, no. 2, pp. 137–154, 1992.

[131] B. Tordoff, W. Mayol, T. de Campos, and D. Murray, "Head pose estimation for wearable robot control," in *British Machine Vision Conference*, pp. 807–816, 2002.

[132] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon, "Bundle adjustment – a modern synthesis," in *Vision Algorithms: Theory and Practice*, (B. Triggs, A. Zisserman, and R. Szeliski, eds.), pp. 298–372, Springer-Verlag, 2000.

[133] R. Tsai, "A versatile cameras calibration technique for high accuracy 3D machine vision mtrology using off-the-shelf TV cameras and lenses," *IEEE Journal of Robotics and Automation*, vol. 3, no. 4, pp. 323–344, 1987.

[134] T. Tuytelaars and L. VanGool, "Wide baseline stereo matching based on local, affinely invariant regions," in *British Machine Vision Conference*, pp. 412–422, 2000.

[135] M. Uenohara and T. Kanade, "Vision-based object registration for real-time image overlay," *Journal of Cognitive Neuroscience*, vol. 3, no. 1, pp. 71–86, 1991.

[136] L. Vacchetti, V. Lepetit, and P. Fua, "Combining edge and texture information for real-time accurate 3D camera tracking," in *International Symposium on Mixed and Augmented Reality* (Arlington, VA), November 2004.

[137] L. Vacchetti, V. Lepetit, and P. Fua, "Stable real-time 3D tracking using online and offline information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, pp. 1385–1391, October 2004.

[138] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Conference on Computer Vision and Pattern Recognition*, pp. 511–518, 2001.

[139] D. Wagner and I. Schmalsteig, "First steps towards handheld augmented reality," in *Proceedings of the 7th International Conference on Wearable Computers* (New York, USA), 2003.

[140] G. Welch and G. Bishop, "An introduction to Kalman Filter," Technical Report TR95-041, Department of Computer Science, University of North Carolina, 1995. http://www.cs.unc.edu/~welch/kalman/.

[141] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland, "Pfinder: Real-time tracking of the human body," in *Photonics East, SPIE*, 1995.

[142] A. Yuille, P. Hallinan, and D. Cohen, "Feature extraction from faces using deformable templates," *International Journal of Computer Vision*, vol. 8, no. 2, pp. 99–111, 1992.

[143] Z. Zhang, "A Flexible New Technique for Camera Calibration," http://research.microsoft.com/ zhang/Calib.

[144] Z. Zhang, "A flexible new technique for camera calibration.," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 1330–1334, 2000.

[145] Z. Zhang, R. Deriche, O. Faugeras, and Q. Luong, "A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry," *Artificial Intelligence*, vol. 78, pp. 87–119, 1995.