Chapter 11

Structure from motion and SLAM

R. Szeliski, *Computer Vision*, Texts in Computer Science,
https://doi.org/10.1007/978-3-030-34372-9_11

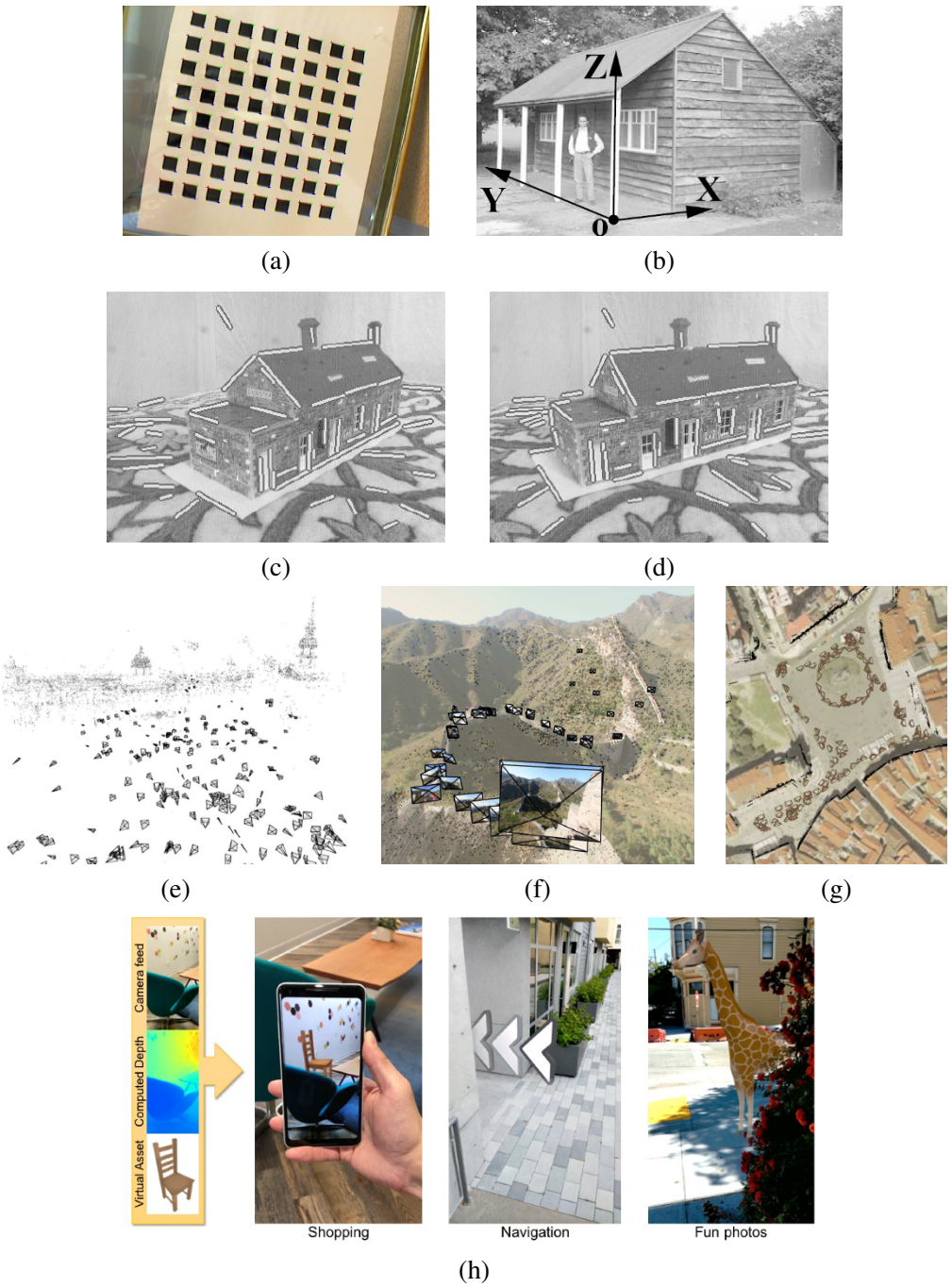


Figure 11.1 Structure from motion examples: (a) a two-dimensional calibration target (Zhang 2000) © 2000 IEEE; (b) single view metrology (Criminisi, Reid, and Zisserman 2000) © 2000 Springer. (c–d) line matching (Schmid and Zisserman 1997) © 1997 IEEE; (e–g) 3D reconstructions of Trafalgar Square, Great Wall of China, and Prague Old Town Square (Snavely, Seitz, and Szeliski 2006) © 2006 ACM; (h) smartphone augmented reality showing real-time depth occlusion effects (Valentin, Kowdle *et al.* 2018) © 2018 ACM.

The reconstruction of 3D models from images has been one of the central topics in computer vision since its inception (Figure 1.7). In fact, it was then believed that the construction of 3D models was a prerequisite for scene understanding and recognition (Marr 1982), although work in the last few decades has proven otherwise. However, 3D modeling has also proven to be immensely useful in applications such as virtual tourism (Section 11.4.6), autonomous navigation (Section 11.5.1), and augmented reality (Section 11.5.2).

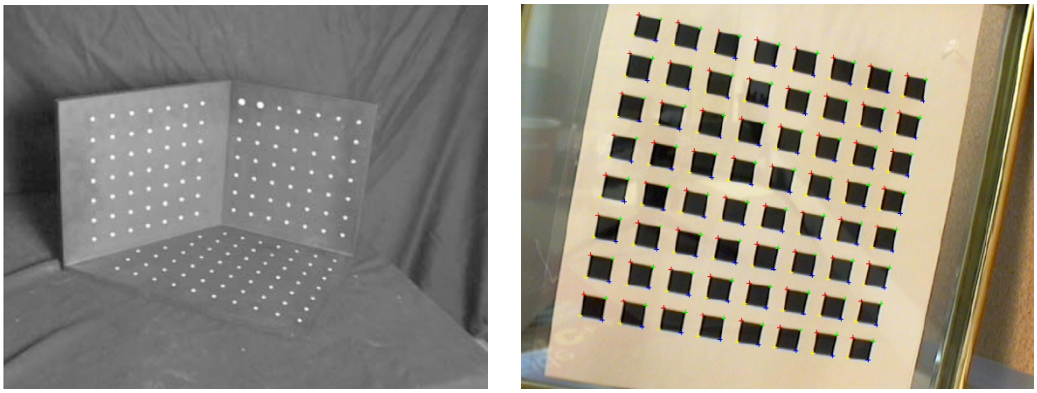
In the last three chapters, we focused on techniques for establishing correspondences between 2D images and using these in a variety of applications such as image stitching, video enhancement, and computational photography. In this chapter, we turn to the topic of using such correspondences to build sparse 3D models of a scene and to re-localize cameras with respect to such models. While this process often involves simultaneously estimating both 3D geometry (structure) and camera pose (motion), it is commonly known (for historical reasons) as *structure from motion* (Ullman 1979).

The topics of projective geometry and structure from motion are extremely rich and some excellent textbooks and surveys have been written on them (Faugeras and Luong 2001; Hartley and Zisserman 2004; Moons, Van Gool, and Vergauwen 2010; Ma, Soatto *et al.* 2012). This chapter skips over a lot of the richer material available in these books, such as the trifocal tensor and algebraic techniques for full self-calibration, and concentrates instead on the basics that we have found useful in large-scale, image-based reconstruction problems (Snavely, Seitz, and Szeliski 2006).

We begin this chapter in Section 11.1 with a review of commonly used techniques for calibrating the camera *intrinsics*, e.g., the focal length and radial distortion parameters we introduced in Sections 2.1.4–2.1.5. Next, we discuss how to estimate the *extrinsic pose* of a camera from 3D to 2D point correspondences (Section 11.2) as well as how to *triangulate* a set of 2D correspondences to estimate a point's 3D location. We then look at the two-frame structure from motion problem (Section 11.3), which involves the determination of the *epipolar geometry* between two cameras and which can also be used to recover certain information about the camera intrinsics using self-calibration (Section 11.3.4). Section 11.4.1 looks at *factorization* approaches to simultaneously estimating structure and motion from large numbers of point tracks using orthographic approximations to the projection model. We then develop a more general and useful approach to structure from motion, namely the simultaneous *bundle adjustment* of all the camera and 3D structure parameters (Section 11.4.2). We also look at special cases that arise when there are higher-level structures, such as lines and planes, in the scene (Section 11.4.8). In the last part of this chapter (Section 11.5), we look at real-time systems for simultaneous localization and mapping (SLAM), which reconstruct a 3D world model while moving through an environment, and can be applied to both visual navigation and augmented reality.

11.1 Geometric intrinsic calibration

As we discuss in the next section (Equations (11.14–11.15)), the computation of the internal (intrinsic) camera calibration parameters can occur simultaneously with the estimation of the (extrinsic) pose of the camera with respect to a known calibration target. This, indeed, is the "classic" approach to camera calibration used in both the photogrammetry (Slama 1980) and the computer vision (Tsai 1987) communities. In this section, we look at simpler alternative formulations that may not involve the full solution of a non-linear regression problem, the use of alternative calibration targets, and the estimation of the non-linear part of camera optics such as radial distortion. In some applications, you can use the EXIF tags associated with a JPEG image to obtain a rough estimate of a camera's focal length and hence to initialize iterative estimation algorithms; but this technique should be used with caution as the results are often inaccurate.



(a) (b)

Figure 11.2 Calibration patterns: (a) a three-dimensional target (Quan and Lan 1999) © 1999 IEEE; (b) a two-dimensional target (Zhang 2000) © 2000 IEEE. Note that radial distortion needs to be removed from such images before the feature points can be used for calibration.

Calibration patterns

The use of a calibration pattern or set of markers is one of the more reliable ways to estimate a camera's intrinsic parameters. In photogrammetry, it is common to set up a camera in a large field looking at distant calibration targets whose exact location has been precomputed using surveying equipment (Slama 1980; Atkinson 1996; Kraus 1997). In this case, the translational component of the pose becomes irrelevant and only the camera rotation and intrinsic parameters need to be recovered.

If a smaller calibration rig needs to be used, e.g., for indoor robotics applications or for mobile robots that carry their own calibration target, it is best if the calibration object can span as much of the workspace as possible (Figure 11.2a), as planar targets often fail to accurately predict the components of the pose that lie far away from the plane. A good way to determine if the calibration has been successfully performed is to estimate the covariance in the parameters (Section 8.1.4) and then project 3D points from various points in the workspace into the image in order to estimate their 2D positional uncertainty.

If no calibration pattern is available, it is also possible to perform calibration simultaneously with structure and pose recovery (Sections 11.1.3 and 11.4.2), which is known as *self-calibration* (Faugeras, Luong, and Maybank 1992; Pollefeys, Koch, and Van Gool 1999; Hartley and Zisserman 2004; Moons, Van Gool, and Vergauwen 2010). However, such an approach requires a large amount of imagery to be accurate.

Planar calibration patterns

When a finite workspace is being used and accurate machining and motion control platforms are available, a good way to perform calibration is to move a planar calibration target through the workspace volume and use the known 3D point locations for calibration. This approach is sometimes called the *N-planes* calibration approach (Gremban, Thorpe, and Kanade 1988; Champleboux, Lavallée *et al.* 1992b; Grossberg and Nayar 2001) and has the advantage that each camera pixel can be mapped to a unique 3D ray in space, which takes care of both linear effects modeled by the calibration matrix $\mathbf{K}$ and non-linear effects such as radial distortion (Section 11.1.4).

A less cumbersome but also less accurate calibration can be obtained by waving a planar calibra-

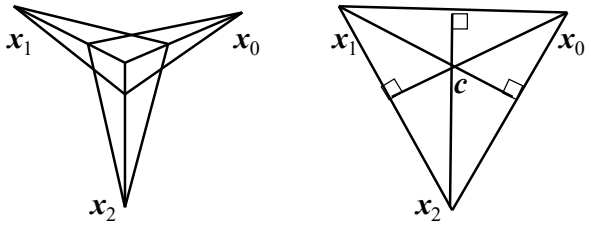


Figure 11.3 Calibration from vanishing points: (a) any pair of finite vanishing points $(\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j)$ can be used to estimate the focal length; (b) the orthocenter of the vanishing point triangle gives the image center of the image $\mathbf{c}$.

tion pattern in front of a camera (Figure 11.2b). In this case, the pattern's pose has to be recovered in conjunction with the intrinsics. In this technique, each input image is used to compute a separate homography (8.19–8.23) $\tilde{\mathbf{H}}$ mapping the plane's calibration points $(X_i, Y_i, 1)$ into image coordinates (x_i, y_i),

$$\mathbf{x}_i = \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \sim \mathbf{K} \begin{bmatrix} \mathbf{r}_0 & \mathbf{r}_1 & \mathbf{t} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ 1 \end{bmatrix} \sim \tilde{\mathbf{H}} \mathbf{p}_i, \tag{11.1}$$

where the $\mathbf{r}_i$ are the first two columns of $\mathbf{R}$ and $\sim$ indicates equality up to scale. From these, Zhang (2000) shows how to form linear constraints on the nine entries in the $\mathbf{B} = \mathbf{K}^{-T}\mathbf{K}^{-1}$ matrix, from which the calibration matrix $\mathbf{K}$ can be recovered using a matrix square root and inversion. The matrix $\mathbf{B}$ is known as the *image of the absolute conic* (IAC) in projective geometry and is commonly used for camera calibration (Hartley and Zisserman 2004, Section 8.5). If only the focal length is being recovered, the even simpler approach of using vanishing points described below can be used instead.

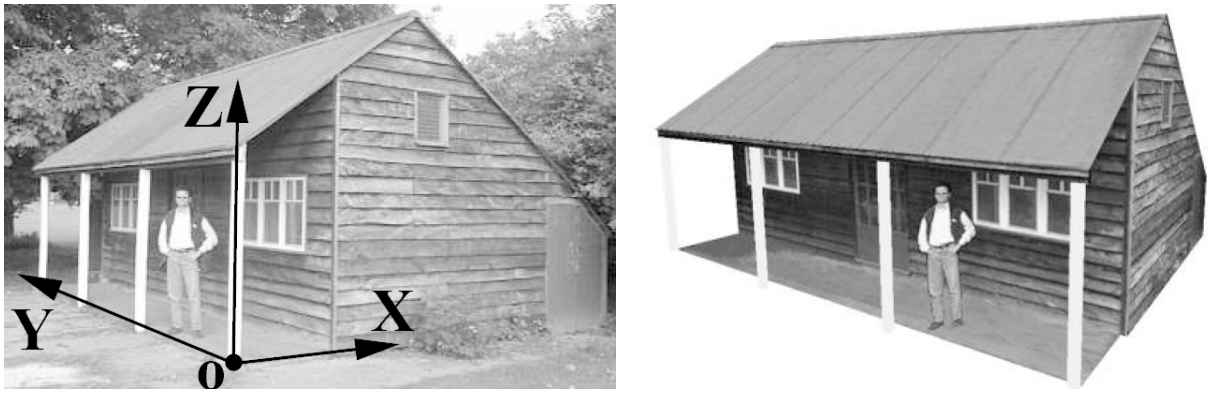
11.1.1 Vanishing points

A common case for calibration that occurs often in practice is when the camera is looking at a manufactured or architectural scene with long extended rectangular patterns such as boxes or building walls. In this case, we can intersect the 2D lines corresponding to 3D parallel lines to compute their *vanishing points*, as described in Section 7.4.3, and use these to determine the intrinsic and extrinsic calibration parameters (Caprile and Torre 1990; Becker and Bove 1995; Liebowitz and Zisserman 1998; Cipolla, Drummond, and Robertson 1999; Antone and Teller 2002; Criminisi, Reid, and Zisserman 2000; Hartley and Zisserman 2004; Pflugfelder 2008).

Let us assume that we have detected two or more orthogonal vanishing points, all of which are *finite*, i.e., they are not obtained from lines that appear to be parallel in the image plane (Figure 11.3a). Let us also assume a simplified form for the calibration matrix $\mathbf{K}$ where only the focal length is unknown (2.59). It is often safe for rough 3D modeling to assume that the optical center is at the center of the image, that the aspect ratio is 1, and that there is no skew. In this case, the projection equation for the vanishing points can be written as

$$\hat{\mathbf{x}}_i = \begin{bmatrix} x_i - c_x \\ y_i - c_y \\ f \end{bmatrix} \sim \mathbf{R}\mathbf{p}_i = \mathbf{r}_i, \tag{11.2}$$

where $\mathbf{p}_i$ corresponds to one of the cardinal directions $(1, 0, 0)$, $(0, 1, 0)$, or $(0, 0, 1)$, and $\mathbf{r}_i$ is the ith column of the rotation matrix $\mathbf{R}$.



(a) (b)

Figure 11.4 Single view metrology (Criminisi, Reid, and Zisserman 2000) © 2000 Springer: (a) input image showing the three coordinate axes computed from the two horizontal vanishing points (which can be determined from the sidings on the shed); (b) a new view of the 3D reconstruction.

From the orthogonality between columns of the rotation matrix, we have

$$\mathbf{r}_i \cdot \mathbf{r}_j \sim (x_i - c_x)(x_j - c_x) + (y_i - c_y)(y_j - c_y) + f^2 = 0, \quad i \neq j \tag{11.3}$$

from which we can obtain an estimate for f^2. Note that the accuracy of this estimate increases as the vanishing points move closer to the center of the image. In other words, it is best to tilt the calibration pattern a decent amount around the 45° axis, as in Figure 11.3a. Once the focal length f has been determined, the individual columns of $\mathbf{R}$ can be estimated by normalizing the left-hand side of (11.2) and taking cross products. Alternatively, the orthogonal Procrustes algorithm (8.32) can be used.

If all three vanishing points are visible and finite in the same image, it is also possible to estimate the image center as the orthocenter of the triangle formed by the three vanishing points (Caprile and Torre 1990; Hartley and Zisserman 2004, Section 8.6) (Figure 11.3b). In practice, however, it is more accurate to re-estimate any unknown intrinsic calibration parameters using non-linear least squares (11.14).

11.1.2 *Application*: Single view metrology

A fun application of vanishing point estimation and camera calibration is the *single view metrology* system developed by Criminisi, Reid, and Zisserman (2000). Their system allows people to interactively measure heights and other dimensions as well as to build piecewise-planar 3D models, as shown in Figure 11.4.

The first step in their system is to identify two orthogonal vanishing points on the ground plane and the vanishing point for the vertical direction, which can be done by drawing some parallel sets of lines in the image. Alternatively, automated techniques such as those discussed in Section 7.4.3 or by Schaffalitzky and Zisserman (2000) could be used. The user then marks a few dimensions in the image, such as the height of a reference object, and the system can automatically compute the height of another object. Walls and other planar impostors (geometry) can also be sketched and reconstructed.

In the formulation originally developed by Criminisi, Reid, and Zisserman (2000), the system produces an *affine* reconstruction, i.e., one that is only known up to a set of independent scaling

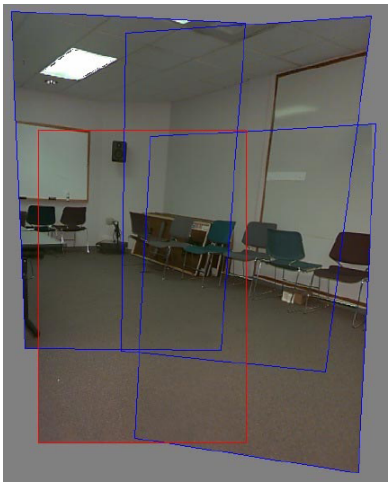


Figure 11.5 Four images taken with a hand-held camera registered using a 3D rotation motion model, which can be used to estimate the focal length of the camera (Szeliski and Shum 1997) © 2000 ACM.

factors along each axis. A potentially more useful system can be constructed by assuming that the camera is calibrated up to an unknown focal length, which can be recovered from orthogonal (finite) vanishing directions, as we have just described in Section 11.1.1. Once this is done, the user can indicate an origin on the ground plane and another point a known distance away. From this, points on the ground plane can be directly projected into 3D, and points above the ground plane, when paired with their ground plane projections, can also be recovered. A fully metric reconstruction of the scene then becomes possible.

Exercise 11.4 has you implement such a system and then use it to model some simple 3D scenes. Section 13.6.1 describes other, potentially multi-view, approaches to architectural reconstruction, including an interactive piecewise-planar modeling system that uses vanishing points to establish 3D line directions and plane normals (Sinha, Steedly *et al.* 2008).

11.1.3 Rotational motion

When no calibration targets or known structures are available but you can rotate the camera around its front nodal point (or, equivalently, work in a large open environment where all objects are distant), the camera can be calibrated from a set of overlapping images by assuming that it is undergoing pure rotational motion, as shown in Figure 11.5 (Stein 1995; Hartley 1997b; Hartley, Hayman *et al.* 2000; de Agapito, Hayman, and Reid 2001; Kang and Weiss 1999; Shum and Szeliski 2000; Frahm and Koch 2003). When a full 360° motion is used to perform this calibration, a very accurate estimate of the focal length f can be obtained, as the accuracy in this estimate is proportional to the total number of pixels in the resulting cylindrical panorama (Section 8.2.6) (Stein 1995; Shum and Szeliski 2000).

To use this technique, we first compute the homographies $\tilde{\mathbf{H}}_{ij}$ between all overlapping pairs of images, as explained in Equations (8.19–8.23). Then, we use the observation, first made in Equation (2.72) and explored in more detail in Equation (8.38), that each homography is related to the inter-camera rotation $\mathbf{R}_{ij}$ through the (unknown) calibration matrices $\mathbf{K}_i$ and $\mathbf{K}_j$,

$$\tilde{\mathbf{H}}_{ij} = \mathbf{K}_i \mathbf{R}_i \mathbf{R}_j^{-1} \mathbf{K}_j^{-1} = \mathbf{K}_i \mathbf{R}_{ij} \mathbf{K}_j^{-1}. \tag{11.4}$$

The simplest way to obtain the calibration is to use the simplified form of the calibration matrix (2.59), where we assume that the pixels are square and the image center lies at the geometric center of the 2D pixel array, i.e., $\mathbf{K}_k = \mathrm{diag}(f_k, f_k, 1)$. We subtract half the width and height from the original pixel coordinates to that the pixel $(x, y) = (0, 0)$ lies at the center of the image. We can then rewrite Equation (11.4) as

$$\mathbf{R}_{10} \sim \mathbf{K}_1^{-1}\tilde{\mathbf{H}}_{10}\mathbf{K}_0 \sim \begin{bmatrix} h_{00} & h_{01} & f_0^{-1}h_{02} \\ h_{10} & h_{11} & f_0^{-1}h_{12} \\ f_1 h_{20} & f_1 h_{21} & f_0^{-1} f_1 h_{22} \end{bmatrix}, \tag{11.5}$$

where h_{ij} are the elements of $\tilde{\mathbf{H}}_{10}$.

Using the orthonormality properties of the rotation matrix $\mathbf{R}_{10}$ and the fact that the right-hand side of (11.5) is known only up to a scale, we obtain

$$h_{00}^2 + h_{01}^2 + f_0^{-2}h_{02}^2 = h_{10}^2 + h_{11}^2 + f_0^{-2}h_{12}^2 \tag{11.6}$$

and

$$h_{00}h_{10} + h_{01}h_{11} + f_0^{-2}h_{02}h_{12} = 0. \tag{11.7}$$

From this, we can compute estimates for f_0 of

$$f_0^2 = \frac{h_{12}^2 - h_{02}^2}{h_{00}^2 + h_{01}^2 - h_{10}^2 - h_{11}^2} \quad \text{if} \quad \mathrm{h_{00}^2 + h_{01}^2 \neq h_{10}^2 + h_{11}^2} \tag{11.8}$$

or

$$f_0^2 = -\frac{h_{02}h_{12}}{h_{00}h_{10} + h_{01}h_{11}} \quad \text{if} \quad \mathrm{h_{00}h_{10} \neq -h_{01}h_{11}}. \tag{11.9}$$

If neither of these conditions holds, we can also take the dot products between the first (or second) row and the third one. Similar results can be obtained for f_1 as well, by analyzing the columns of $\tilde{\mathbf{H}}_{10}$. If the focal length is the same for both images, we can take the geometric mean of f_0 and f_1 as the estimated focal length $f = \sqrt{f_1 f_0}$. When multiple estimates of f are available, e.g., from different homographies, the median value can be used as the final estimate. A more general (upper-triangular) estimate of $\mathbf{K}$ can be obtained in the case of a fixed-parameter camera $\mathbf{K}_i = \mathbf{K}$ using the technique of Hartley (1997b). Extensions to the cases of temporally varying calibration parameters and non-stationary cameras are discussed by Hartley, Hayman *et al.* (2000) and de Agapito, Hayman, and Reid (2001).

The quality of the intrinsic camera parameters can be greatly increased by constructing a full 360° panorama, as mis-estimating the focal length will result in a gap (or excessive overlap) when the first image in the sequence is stitched to itself (Figure 8.6). The resulting misalignment can be used to improve the estimate of the focal length and to re-adjust the rotation estimates, as described in Section 8.2.4. Rotating the camera by 90° around its optical axis and re-shooting the panorama is a good way to check for aspect ratio and skew pixel problems, as is generating a full hemi-spherical panorama when there is sufficient texture.

Ultimately, however, the most accurate estimate of the calibration parameters (including radial distortion) can be obtained using a full simultaneous non-linear minimization of the intrinsic and extrinsic (rotation) parameters, as described in Section 11.2.2.

11.1.4 Radial distortion

When images are taken with wide-angle lenses, it is often necessary to model *lens distortions* such as *radial distortion*. As discussed in Section 2.1.5, the radial distortion model says that coordinates in

the observed images are displaced towards (*barrel* distortion) or away (*pincushion* distortion) from the image center by an amount proportional to their radial distance (Figure 2.13a–b). The simplest radial distortion models use low-order polynomials (c.f. Equation (2.78)),

$$\begin{aligned} \hat{x} &= x(1 + \kappa_1 r^2 + \kappa_2 r^4) \\ \hat{y} &= y(1 + \kappa_1 r^2 + \kappa_2 r^4), \end{aligned} \tag{11.10}$$

where $(x, y) = (0, 0)$ at the radial distortion center (2.77), $r^2 = x^2 + y^2$, and κ_1 and κ_2 are called the *radial distortion parameters* (Brown 1971; Slama 1980).[1]

A variety of techniques can be used to estimate the radial distortion parameters for a given lens, if the digital camera has not already done this in its capture software. One of the simplest and most useful is to take an image of a scene with a lot of straight lines, especially lines aligned with and near the edges of the image. The radial distortion parameters can then be adjusted until all of the lines in the image are straight, which is commonly called the *plumb-line method* (Brown 1971; Kang 2001; El-Melegy and Farag 2003). Exercise 11.5 gives some more details on how to implement such a technique.

Another approach is to use several overlapping images and to combine the estimation of the radial distortion parameters with the image alignment process, i.e., by extending the pipeline used for stitching in Section 8.3.1. Sawhney and Kumar (1999) use a hierarchy of motion models (translation, affine, projective) in a coarse-to-fine strategy coupled with a quadratic radial distortion correction term. They use direct (intensity-based) minimization to compute the alignment. Stein (1997) uses a feature-based approach combined with a general 3D motion model (and quadratic radial distortion), which requires more matches than a parallax-free rotational panorama but is potentially more general. More recent approaches sometimes simultaneously compute both the unknown intrinsic parameters and the radial distortion coefficients, which may include higher-order terms or more complex rational or non-parametric forms (Claus and Fitzgibbon 2005; Sturm 2005; Thirthala and Pollefeys 2005; Barreto and Daniilidis 2005; Hartley and Kang 2005; Steele and Jaynes 2006; Tardif, Sturm *et al.* 2009).

When a known calibration target is being used (Figure 11.2), the radial distortion estimation can be folded into the estimation of the other intrinsic and extrinsic parameters (Zhang 2000; Hartley and Kang 2007; Tardif, Sturm *et al.* 2009). This can be viewed as adding another stage to the general non-linear minimization pipeline shown in Figure 11.7 between the intrinsic parameter multiplication box $\mathbf{f}_{\mathrm{C}}$ and the perspective division box $\mathbf{f}_{\mathrm{P}}$. (See Exercise 11.6 on more details for the case of a planar calibration target.)

Of course, as discussed in Section 2.1.5, more general models of lens distortion, such as fisheye and non-central projection, may sometimes be required. While the parameterization of such lenses may be more complicated (Section 2.1.5), the general approach of either using calibration rigs with known 3D positions or self-calibration through the use of multiple overlapping images of a scene can both be used (Hartley and Kang 2007; Tardif, Sturm, and Roy 2007). The same techniques used to calibrate for radial distortion can also be used to reduce the amount of chromatic aberration by separately calibrating each color channel and then warping the channels to put them back into alignment (Exercise 11.7).

[1] Sometimes the relationship between x and $\hat{x}$ is expressed the other way around, i.e., using primed (final) coordinates on the right-hand side, $x = \hat{x}(1 + \kappa_1 \hat{r}^2 + \kappa_2 \hat{r}^4)$. This is convenient if we map image pixels into (warped) rays and then undistort the rays to obtain 3D rays in space, i.e., if we are using inverse warping.

11.2 Pose estimation

A particular instance of feature-based alignment, which occurs very often, is estimating an object's 3D pose from a set of 2D point projections. This *pose estimation* problem is also known as *extrinsic* calibration, as opposed to the *intrinsic* calibration of internal camera parameters such as focal length, which we discuss in Section 11.1. The problem of recovering pose from three correspondences, which is the minimal amount of information necessary, is known as the *perspective-3-point-problem* (P3P),[2] with extensions to larger numbers of points collectively known as PnP (Haralick, Lee *et al.* 1994; Quan and Lan 1999; Gao, Hou *et al.* 2003; Moreno-Noguer, Lepetit, and Fua 2007; Persson and Nordberg 2018).

In this section, we look at some of the techniques that have been developed to solve such problems, starting with the *direct linear transform* (DLT), which recovers a 3×4 camera matrix, followed by other "linear" algorithms, and then looking at statistically optimal iterative algorithms.

11.2.1 Linear algorithms

The simplest way to recover the pose of the camera is to form a set of rational linear equations analogous to those used for 2D motion estimation (8.19) from the camera matrix form of perspective projection (2.55–2.56),

$$x_i = \frac{p_{00}X_i + p_{01}Y_i + p_{02}Z_i + p_{03}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}} \tag{11.11}$$

$$y_i = \frac{p_{10}X_i + p_{11}Y_i + p_{12}Z_i + p_{13}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}}, \tag{11.12}$$

where (x_i, y_i) are the measured 2D feature locations and (X_i, Y_i, Z_i) are the known 3D feature locations (Figure 11.6). As with (8.21), this system of equations can be solved in a linear fashion for the unknowns in the camera matrix $\mathbf{P}$ by multiplying the denominator on both sides of the equation.Because $\mathbf{P}$ is unknown up to a scale, we can either fix one of the entries, e.g., $p_{23} = 1$, or find the smallest singular vector of the set of linear equations. The resulting algorithm is called the *direct linear transform* (DLT) and is commonly attributed to Sutherland (1974). (For a more in-depth discussion, see Hartley and Zisserman (2004).) To compute the 12 (or 11) unknowns in $\mathbf{P}$, at least six correspondences between 3D and 2D locations must be known.

As with the case of estimating homographies (8.21–8.23), more accurate results for the entries in $\mathbf{P}$ can be obtained by directly minimizing the set of Equations (11.11–11.12) using non-linear least squares with a small number of iterations. Note that instead of taking the ratios of the X/Z and Y/Z values as in (11.11–11.12), it is also possible to take a cross product of the 3-vector $(x_i, y_i, 1)$ image measurement and the 3-D ray (X, Y, Z) and set the three elements of this cross-product to 0. The resulting three equations, when interpreted as a set of least squares constraints, in effect compute the squared sine of the angle between the two rays.

Once the entries in $\mathbf{P}$ have been recovered, it is possible to recover both the intrinsic calibration matrix $\mathbf{K}$ and the rigid transformation $(\mathbf{R}, \mathbf{t})$ by observing from Equation (2.56) that

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]. \tag{11.13}$$

Because $\mathbf{K}$ is upper-triangular (see the discussion in Section 2.1.4), both $\mathbf{K}$ and $\mathbf{R}$ can be obtained from the front 3×3 sub-matrix of $\mathbf{P}$ using RQ factorization (Golub and Van Loan 1996).[3]

[2] The "3-point" algorithms actually require a 4th point to resolve a 4-way ambiguity.

[3] Note the unfortunate clash of terminologies: In matrix algebra textbooks, $\mathbf{R}$ represents an upper-triangular matrix; in computer vision, $\mathbf{R}$ is an orthogonal rotation.

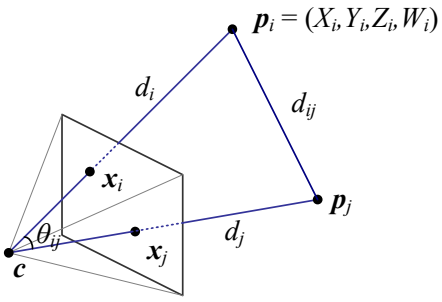


Figure 11.6 Pose estimation by the direct linear transform and by measuring visual angles and distances between pairs of points.

In most applications, however, we have some prior knowledge about the intrinsic calibration matrix $\mathbf{K}$, e.g., that the pixels are square, the skew is very small, and the image center is near the geometric center of the image (2.57–2.59). Such constraints can be incorporated into a non-linear minimization of the parameters in $\mathbf{K}$ and $(\mathbf{R}, \mathbf{t})$, as described in Section 11.2.2.

In the case where the camera is already calibrated, i.e., the matrix $\mathbf{K}$ is known (Section 11.1), we can perform pose estimation using as few as three points (Fischler and Bolles 1981; Haralick, Lee *et al.* 1994; Quan and Lan 1999). The basic observation that these *linear PnP* (*perspective n-point*) algorithms employ is that the visual angle between any pair of 2D points $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{x}}_j$ must be the same as the angle between their corresponding 3D points $\mathbf{p}_i$ and $\mathbf{p}_j$ (Figure 11.6).

A full derivation of this approach can be found in the first edition of this book (Szeliski 2010, Section 6.2.1) and also in (Quan and Lan 1999), where the authors provide accuracy results for this and other techniques, which use fewer points but require more complicated algebraic manipulations. The paper by Moreno-Noguer, Lepetit, and Fua (2007) reviews other alternatives and also gives a lower complexity algorithm that typically produces more accurate results. An even more recent paper by Terzakis and Lourakis (2020) reviews papers published in the last decade.

Unfortunately, because minimal PnP solutions can be quite noise sensitive and also suffer from *bas-relief ambiguities* (e.g., depth reversals) (Section 11.4.5), it is prudent to optimize the initial estimates from PnP using the iterative technique described in Section 11.2.2. An alternative pose estimation algorithm involves starting with a scaled orthographic projection model and then iteratively refining this initial estimate using a more accurate perspective projection model (DeMenthon and Davis 1995). The attraction of this model, as stated in the paper's title, is that it can be implemented "in 25 lines of [Mathematica] code".

CNN-based pose estimation

As with other areas on computer vision, deep neural networks have also been applied to pose estimation. Some representative papers include Xiang, Schmidt *et al.* (2018), Oberweger, Rad, and Lepetit (2018), Hu, Hugonot *et al.* (2019), Peng, Liu *et al.* (2019), and (Hu, Fua *et al.* 2020) for object pose estimation, and papers such as Kendall and Cipolla (2017) and Kim, Dunn, and Frahm (2017) discussed in Section 11.2.3 on location recognition. There is also a very active community around estimating pose from RGB-D images, with the most recent papers (Hagelskjær and Buch 2020; Labbé, Carpentier *et al.* 2020) evaluated on the BOP (Benchmark for 6DOF Object Pose) (Hodaň, Michel *et al.* 2018).[4]

[4] https://bop.felk.cvut.cz/challenges/bop-challenge-2020, https://cmp.felk.cvut.cz/sixd/workshop_2020

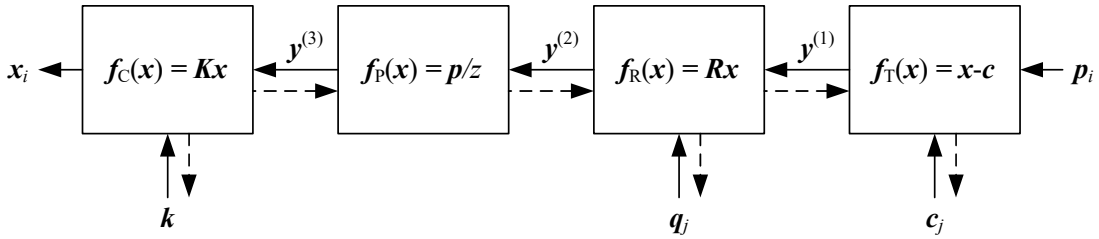


Figure 11.7 A set of chained transforms for projecting a 3D point $\mathbf{p}_i$ to a 2D measurement $\mathbf{x}_i$ through a series of transformations $\mathbf{f}^{(k)}$, each of which is controlled by its own set of parameters. The dashed lines indicate the flow of information as partial derivatives are computed during a backward pass.

11.2.2 Iterative non-linear algorithms

The most accurate and flexible way to estimate pose is to directly minimize the squared (or robust) reprojection error for the 2D points as a function of the unknown pose parameters in $(\mathbf{R}, \mathbf{t})$ and optionally $\mathbf{K}$ using non-linear least squares (Tsai 1987; Bogart 1991; Gleicher and Witkin 1992). We can write the projection equations as

$$\mathbf{x}_i = \mathbf{f}(\mathbf{p}_i; \mathbf{R}, \mathbf{t}, \mathbf{K}) \tag{11.14}$$

and iteratively minimize the robustified linearized reprojection errors

$$E_{\text{NLP}} = \sum_i \rho\left(\frac{\partial \mathbf{f}}{\partial \mathbf{R}}\Delta\mathbf{R} + \frac{\partial \mathbf{f}}{\partial \mathbf{t}}\Delta\mathbf{t} + \frac{\partial \mathbf{f}}{\partial \mathbf{K}}\Delta\mathbf{K} - \mathbf{r}_i\right), \tag{11.15}$$

where $\mathbf{r}_i = \tilde{\mathbf{x}}_i - \hat{\mathbf{x}}_i$ is the current residual vector (2D error in predicted position) and the partial derivatives are with respect to the unknown pose parameters (rotation, translation, and optionally calibration). The *robust loss function* ρ, which we first introduced in (4.15) in Section 4.1.3, is used to reduce the influence of outlier correspondences. Note that if full 2D covariance estimates are available for the 2D feature locations, the above squared norm can be weighted by the inverse point covariance matrix, as in Equation (8.11).

An easier to understand (and implement) version of the above non-linear regression problem can be constructed by re-writing the projection equations as a concatenation of simpler steps, each of which transforms a 4D homogeneous coordinate $\mathbf{p}_i$ by a simple transformation such as translation, rotation, or perspective division (Figure 11.7). The resulting projection equations can be written as

$$\mathbf{y}^{(1)} = \mathbf{f}_{\text{T}}(\mathbf{p}_i; \mathbf{c}_j) = \mathbf{p}_i - \mathbf{c}_j, \tag{11.16}$$

$$\mathbf{y}^{(2)} = \mathbf{f}_{\text{R}}(\mathbf{y}^{(1)}; \mathbf{q}_j) = \mathbf{R}(\mathbf{q}_j)\,\mathbf{y}^{(1)}, \tag{11.17}$$

$$\mathbf{y}^{(3)} = \mathbf{f}_{\text{P}}(\mathbf{y}^{(2)}) = \frac{\mathbf{y}^{(2)}}{z^{(2)}}, \tag{11.18}$$

$$\mathbf{x}_i = \mathbf{f}_{\text{C}}(\mathbf{y}^{(3)}; \mathbf{k}) = \mathbf{K}(\mathbf{k})\,\mathbf{y}^{(3)}. \tag{11.19}$$

Note that in these equations, we have indexed the camera centers $\mathbf{c}_j$ and camera rotation quaternions $\mathbf{q}_j$ by an index j, in case more than one pose of the calibration object is being used (see also Section 11.4.2.) We are also using the camera center $\mathbf{c}_j$ instead of the world translation $\mathbf{t}_j$, as this is a more natural parameter to estimate.

The advantage of this chained set of transformations is that each one has a simple partial derivative with respect both to its parameters and to its input. Thus, once the predicted value of $\tilde{\mathbf{x}}_i$ has

been computed based on the 3D point location $\mathbf{p}_i$ and the current values of the pose parameters $(\mathbf{c}_j, \mathbf{q}_j, \mathbf{k})$, we can obtain all of the required partial derivatives using the chain rule

$$\frac{\partial \mathbf{r}_i}{\partial \mathbf{p}^{(k)}} = \frac{\partial \mathbf{r}_i}{\partial \mathbf{y}^{(k)}} \frac{\partial \mathbf{y}^{(k)}}{\partial \mathbf{p}^{(k)}}, \tag{11.20}$$

where $\mathbf{p}^{(k)}$ indicates one of the parameter vectors that is being optimized. (This same "trick" is used in neural networks as part of *backpropagation*, which we presented in Figure 5.31.)

The one special case in this formulation that can be considerably simplified is the computation of the rotation update. Instead of directly computing the derivatives of the 3×3 rotation matrix $\mathbf{R}(\mathbf{q})$ as a function of the unit quaternion entries, you can prepend the incremental rotation matrix $\Delta\mathbf{R}(\boldsymbol{\omega})$ given in Equation (2.35) to the current rotation matrix and compute the partial derivative of the transform with respect to these parameters, which results in a simple cross product of the backward chaining partial derivative and the outgoing 3D vector, as explained in Equation (2.36).

Target-based augmented reality

A widely used application of pose estimation is *augmented reality*, where virtual 3D images or annotations are superimposed on top of a live video feed, either through the use of see-through glasses (a head-mounted display) or on a regular computer or mobile device screen (Azuma, Baillot *et al.* 2001; Haller, Billinghurst, and Thomas 2007; Billinghurst, Clark, and Lee 2015). In some applications, a special pattern printed on cards or in a book is tracked to perform the augmentation (Kato, Billinghurst *et al.* 2000; Billinghurst, Kato, and Poupyrev 2001). For a desktop application, a grid of dots printed on a mouse pad can be tracked by a camera embedded in an augmented mouse to give the user control of a full six degrees of freedom over their position and orientation in a 3D space (Hinckley, Sinclair *et al.* 1999). Today, tracking known targets such as movie posters is used in some phone-based augmented reality systems such as Facebook's Spark AR.[5]

Sometimes, the scene itself provides a convenient object to track, such as the rectangle defining a desktop used in *through-the-lens camera control* (Gleicher and Witkin 1992). In outdoor locations, such as film sets, it is more common to place special markers such as brightly colored balls in the scene to make it easier to find and track them (Bogart 1991). In older applications, surveying techniques were used to determine the locations of these balls before filming. Today, it is more common to apply structure-from-motion directly to the film footage itself (Section 11.5.2).

Exercise 8.4 has you implement a tracking and pose estimation system for augmented-reality applications.

11.2.3 *Application*: Location recognition

One of the most exciting applications of pose estimation is in the area of location recognition, which can be used both in desktop applications (*"Where did I take this holiday snap?"*) and in mobile smartphone applications. The latter case includes not only finding out your current location based on a cell-phone image, but also providing you with navigation directions or annotating your images with useful information, such as building names and restaurant reviews (i.e., a pocketable form of *augmented reality*). This problem is also often called *visual (or image-based) localization* (Se, Lowe, and Little 2002; Zhang and Kosecka 2006; Janai, Güney *et al.* 2020, Section 13.3) or *visual place recognition* (Lowry, Sünderhauf *et al.* 2015).

[5] https://sparkar.facebook.com/ar-studio

Figure 11.8 Feature-based location recognition (Schindler, Brown, and Szeliski 2007) © 2007 IEEE: (a) three typical series of overlapping street photos; (b) handheld camera shots and (c) their corresponding database photos.

Some approaches to location recognition assume that the photos consist of architectural scenes for which vanishing directions can be used to pre-rectify the images for easier matching (Robertson and Cipolla 2004). Other approaches use general affine covariant interest points to perform *wide baseline matching* (Schaffalitzky and Zisserman 2002), with the winning entry on the ICCV 2005 Computer Vision Contest (Szeliski 2005) using this approach (Zhang and Kosecka 2006). The Photo Tourism system of Snavely, Seitz, and Szeliski (2006) (Section 14.1.2) was the first to apply these kinds of ideas to large-scale image matching and (implicit) location recognition from internet photo collections taken under a wide variety of viewing conditions.

The main difficulty in location recognition is in dealing with the extremely large community (user-generated) photo collections on websites such as Flickr (Philbin, Chum *et al.* 2007; Chum, Philbin *et al.* 2007; Philbin, Chum *et al.* 2008; Irschara, Zach *et al.* 2009; Turcot and Lowe 2009; Sattler, Leibe, and Kobbelt 2011, 2017) or commercially captured databases (Schindler, Brown, and Szeliski 2007; Klingner, Martin, and Roseborough 2013; Torii, Arandjelović *et al.* 2018). The prevalence of commonly appearing elements such as foliage, signs, and common architectural elements further complicates the task (Schindler, Brown, and Szeliski 2007; Jegou, Douze, and Schmid 2009; Chum and Matas 2010b; Knopp, Sivic, and Pajdla 2010; Torii, Sivic *et al.* 2013; Sattler, Havlena *et al.* 2016). Figure 7.26 shows some results on location recognition from community photo collections, while Figure 11.8 shows sample results from denser commercially acquired datasets. In the latter case, the overlap between adjacent database images can be used to verify and prune potential matches using "temporal" filtering, i.e., requiring the query image to match nearby overlapping database images before accepting the match. Similar ideas have been used to improve location recognition from panoramic video sequences (Levin and Szeliski 2004; Samano, Zhou, and Calway 2020) and to combine local SLAM reconstructions from image sequences with matching against a precomputed map for higher reliability (Stenborg, Sattler, and Hammarstrand 2020). Recognizing indoor locations inside buildings and shopping malls poses its own set of challenges, including textureless areas and repeated elements (Levin and Szeliski 2004; Wang, Fidler, and Urtasun 2015; Sun, Xie *et al.* 2017; Taira, Okutomi *et al.* 2018; Taira, Rocco *et al.* 2019; Lee, Ryu *et al.* 2021). The matching of ground-level to aerial images has also been studied (Kaminsky, Snavely *et al.* 2009; Shan, Wu *et al.* 2014).

Some of the initial research on location recognition was organized around the Oxford 5k and Paris 6k datasets (Philbin, Chum *et al.* 2007, 2008; Radenović, Iscen *et al.* 2018), as well as the Vienna (Irschara, Zach *et al.* 2009) and Photo Tourism (Li, Snavely, and Huttenlocher 2010) datasets, and later around the 7 scenes indoor RGB-D dataset (Shotton, Glocker *et al.* 2013) and Cambridge Landmarks (Kendall, Grimes, and Cipolla 2015). The NetVLAD paper (Arandjelovic, Gronat *et al.* 2016) was tested on Google Street View Time Machine data. Currently, the most widely used visual localization datasets are collected at the Long-Term Visual Localization Benchmark[6] and include such datasets as Aachen Day-Night (Sattler, Maddern *et al.* 2018) and InLoc (Taira, Okutomi *et al.* 2018). And while most localization systems work from collections of ground-level images, it is also possible to re-localize based on textured digital elevation (terrain) models for outdoor (non-city) applications (Baatz, Saurer *et al.* 2012; Brejcha, Lukáč *et al.* 2020).

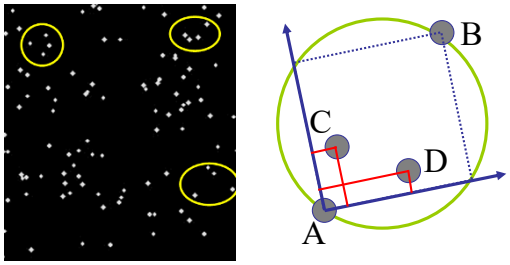
Some of the most recent approaches to localization use deep networks to generate feature descriptors (Arandjelovic, Gronat *et al.* 2016; Kim, Dunn, and Frahm 2017; Torii, Arandjelović *et al.* 2018; Radenović, Tolias, and Chum 2019; Yang, Kien Nguyen *et al.* 2019; Sarlin, Unagar *et al.* 2021), perform large-scale instance retrieval (Radenović, Tolias, and Chum 2019; Cao, Araujo, and Sim 2020; Ng, Balntas *et al.* 2020; Tolias, Jenicek, and Chum 2020; Pion, Humenberger *et al.* 2020 and Section 6.2.3), map images to 3D scene coordinates (Brachmann and Rother 2018), or perform end-to-end scene coordinate regression (Shotton, Glocker *et al.* 2013), absolute pose regression (APR) (Kendall, Grimes, and Cipolla 2015; Kendall and Cipolla 2017), or relative pose regression (RPR) (Melekhov, Ylioinas *et al.* 2017; Balntas, Li, and Prisacariu 2018). Recent evaluations of these techniques have shown that classical approaches based on feature matching followed by geometric pose optimization typically outperform pose regression approaches in terms of accuracy and generalization (Sattler, Zhou *et al.* 2019; Zhou, Sattler *et al.* 2019; Ding, Wang *et al.* 2019; Lee, Ryu *et al.* 2021; Sarlin, Unagar *et al.* 2021).

The Long-Term Visual Localization benchmark has a leaderboard listing the best-performing localization systems. In the CVPR 2020 workshop and challenge, some of the winning entries were based on recent detectors, descriptors, and matchers such as SuperGlue (Sarlin, DeTone *et al.* 2020), ASLFeat (Luo, Zhou *et al.* 2020), and R2D2 (Revaud, Weinzaepfel *et al.* 2019). Other systems that did well include HF-Net (Sarlin, Cadena *et al.* 2019), ONavi (Fan, Zhou *et al.* 2020), and D2-Net (Dusmanu, Rocco *et al.* 2019). An even more recent trend is to use DNNs or transformers to establish dense coarse-to-fine matches (Jiang, Trulls *et al.* 2021; Sun, Shen *et al.* 2021).

Another variant on location recognition is the automatic discovery of *landmarks*, i.e., frequently photographed objects and locations. Simon, Snavely, and Seitz (2007) show how these kinds of objects can be discovered simply by analyzing the matching graph constructed as part of the 3D modeling process in Photo Tourism. More recent work has extended this approach to larger datasets using efficient clustering techniques (Philbin and Zisserman 2008; Li, Wu *et al.* 2008; Chum, Philbin, and Zisserman 2008; Chum and Matas 2010a; Arandjelović and Zisserman 2012), combining metadata such as GPS and textual tags with visual search (Quack, Leibe, and Van Gool 2008; Crandall, Backstrom *et al.* 2009; Li, Snavely *et al.* 2012), and using multiple descriptors to obtain real-time performance in micro aerial vehicle navigation (Lim, Sinha *et al.* 2012). It is now even possible to automatically associate object tags with images based on their co-occurrence in multiple loosely tagged images (Simon and Seitz 2008; Gammeter, Bossard *et al.* 2009).

The concept of organizing the world's photo collections by location has even been recently extended to organizing all of the universe's (astronomical) photos in an application called *astrometry*.[7] The technique used to match any two star fields is to take quadruplets of nearby stars (a pair of

[6] https://www.visuallocalization.net
[7] https://astrometry.net



(a) (b)

Figure 11.9 Locating star fields using astrometry, https://astrometry.net. (a) Input star field and some selected star quads. (b) The 2D coordinates of stars C and D are encoded relative to the unit square defined by A and B.

stars and another pair inside their diameter) to form a 30-bit *geometric hash* by encoding the relative positions of the second pair of points using the inscribed square as the reference frame, as shown in Figure 11.9. Traditional information retrieval techniques (k-d trees built for different parts of a sky atlas) are then used to find matching quads as potential star field location hypotheses, which can then be verified using a similarity transform.

11.2.4 Triangulation

The problem of determining a point's 3D position from a set of corresponding image locations and known camera positions is known as *triangulation*. This problem is the converse of the pose estimation problem we studied in Section 11.2.

One of the simplest ways to solve this problem is to find the 3D point $\mathbf{p}$ that lies closest to all of the 3D rays corresponding to the 2D matching feature locations $\{\mathbf{x}_j\}$ observed by cameras $\{\mathbf{P}_j = \mathbf{K}_j[\mathbf{R}_j|\mathbf{t}_j]\}$, where $\mathbf{t}_j = -\mathbf{R}_j\mathbf{c}_j$ and $\mathbf{c}_j$ is the jth camera center (2.55–2.56). As you can see in Figure 11.10, these rays originate at $\mathbf{c}_j$ in a direction $\hat{\mathbf{v}}_j = \mathcal{N}(\mathbf{R}_j^{-1}\mathbf{K}_j^{-1}\mathbf{x}_j)$, where $\mathcal{N}(\mathbf{v})$ normalizes a vector $\mathbf{v}$ to unit length. The nearest point to $\mathbf{p}$ on this ray, which we denote as $\mathbf{q}_j = \mathbf{c}_j + d_j\hat{\mathbf{v}}_j$, minimizes the distance

$$\|\mathbf{q}_j - \mathbf{p}\|^2 = \|\mathbf{c}_j + d_j\hat{\mathbf{v}}_j - \mathbf{p}\|^2, \tag{11.21}$$

which has a minimum at $d_j = \hat{\mathbf{v}}_j \cdot (\mathbf{p} - \mathbf{c}_j)$. Hence,

$$\mathbf{q}_j = \mathbf{c}_j + (\hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)(\mathbf{p} - \mathbf{c}_j) = \mathbf{c}_j + (\mathbf{p} - \mathbf{c}_j)_{\|}, \tag{11.22}$$

in the notation of Equation (2.29), and the squared distance between $\mathbf{p}$ and $\mathbf{q}_j$ is

$$r_j^2 = \|(\mathbf{I} - \hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)(\mathbf{p} - \mathbf{c}_j)\|^2 = \|(\mathbf{p} - \mathbf{c}_j)_{\perp}\|^2. \tag{11.23}$$

The optimal value for $\mathbf{p}$, which lies closest to all of the rays, can be computed as a regular least squares problem by summing over all the r_j^2 and finding the optimal value of $\mathbf{p}$,

$$\mathbf{p} = \left[\sum_j (\mathbf{I} - \hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)\right]^{-1}\left[\sum_j (\mathbf{I} - \hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T)\mathbf{c}_j\right]. \tag{11.24}$$

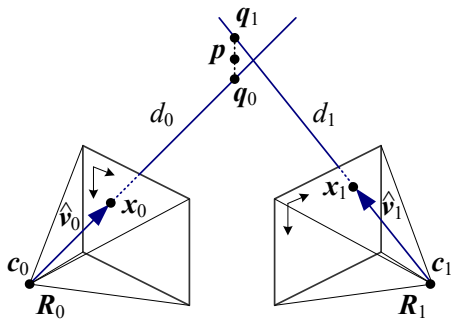


Figure 11.10 3D point triangulation by finding the point $\mathbf{p}$ that lies nearest to all of the optical rays $\mathbf{c}_j + d_j\hat{\mathbf{v}}_j$.

An alternative formulation, which is more statistically optimal and which can produce significantly better estimates if some of the cameras are closer to the 3D point than others, is to minimize the residual in the measurement equations

$$x_j = \frac{p_{00}^{(j)}X + p_{01}^{(j)}Y + p_{02}^{(j)}Z + p_{03}^{(j)}W}{p_{20}^{(j)}X + p_{21}^{(j)}Y + p_{22}^{(j)}Z + p_{23}^{(j)}W} \tag{11.25}$$

$$y_j = \frac{p_{10}^{(j)}X + p_{11}^{(j)}Y + p_{12}^{(j)}Z + p_{13}^{(j)}W}{p_{20}^{(j)}X + p_{21}^{(j)}Y + p_{22}^{(j)}Z + p_{23}^{(j)}W}, \tag{11.26}$$

where (x_j, y_j) are the measured 2D feature locations and $\{p_{00}^{(j)} \dots p_{23}^{(j)}\}$ are the known entries in camera matrix $\mathbf{P}_j$ (Sutherland 1974).

As with Equations (8.21, 11.11, and 11.12), this set of non-linear equations can be converted into a linear least squares problem by multiplying both sides of the denominator, again resulting in the direct linear transform (DLT) formulation. Note that if we use homogeneous coordinates $\mathbf{p} = (X, Y, Z, W)$, the resulting set of equations is homogeneous and is best solved as a singular value decomposition (SVD) or eigenvalue problem (looking for the smallest singular vector or eigenvector). If we set $W = 1$, we can use regular linear least squares, but the resulting system may be singular or poorly conditioned, i.e., if all of the viewing rays are parallel, as occurs for points far away from the camera.

For this reason, it is generally preferable to parameterize 3D points using homogeneous coordinates, especially if we know that there are likely to be points at greatly varying distances from the cameras. Of course, minimizing the set of observations (11.25–11.26) using non-linear least squares, as described in (8.14 and 8.23), is preferable to using linear least squares, regardless of the representation chosen.

For the case of two observations, it turns out that the location of the point $\mathbf{p}$ that exactly minimizes the true reprojection error (11.25–11.26) can be computed using the solution of degree six polynomial equations (Hartley and Sturm 1997). Another problem to watch out for with triangulation is the issue of *cheirality*, i.e., ensuring that the reconstructed points lie in front of all the cameras (Hartley 1998). While this cannot always be guaranteed, a useful heuristic is to take the points that lie behind the cameras because their rays are diverging (imagine Figure 11.10 where the rays were pointing *away* from each other) and to place them on the plane at infinity by setting their W values to 0.

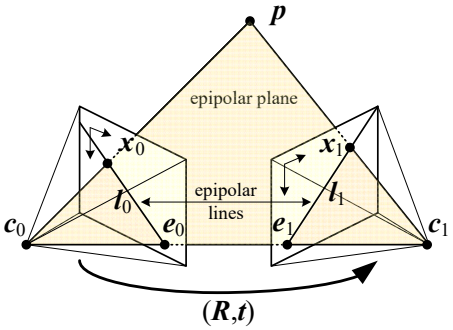


Figure 11.11 Epipolar geometry: The vectors $\mathbf{t} = \mathbf{c}_1 - \mathbf{c}_0$, $\mathbf{p} - \mathbf{c}_0$ and $\mathbf{p} - \mathbf{c}_1$ are co-planar and define the basic epipolar constraint expressed in terms of the pixel measurements $\mathbf{x}_0$ and $\mathbf{x}_1$.

11.3 Two-frame structure from motion

So far in our study of 3D reconstruction, we have always assumed that either the 3D point positions or the 3D camera poses are known in advance. In this section, we take our first look at *structure from motion*, which is the simultaneous recovery of 3D structure and pose from image correspondences. In particular, we examine techniques that operate on just two frames with point correspondences. We divide this section into the study of classic "n-point" algorithms, special (degenerate) cases, projective (uncalibrated) reconstruction, and self-calibration for cameras whose intrinsic calibrations are unknown.

11.3.1 Eight, seven, and five-point algorithms

Consider Figure 11.11, which shows a 3D point $\mathbf{p}$ being viewed from two cameras whose relative position can be encoded by a rotation $\mathbf{R}$ and a translation $\mathbf{t}$. As we do not know anything about the camera positions, without loss of generality, we can set the first camera at the origin $\mathbf{c}_0 = \mathbf{0}$ and at a canonical orientation $\mathbf{R}_0 = \mathbf{I}$.

The 3D point $\mathbf{p}_0 = d_0\hat{\mathbf{x}}_0$ observed in the first image at location $\hat{\mathbf{x}}_0$ and at a z distance of d_0 is mapped into the second image by the transformation

$$d_1\hat{\mathbf{x}}_1 = \mathbf{p}_1 = \mathbf{R}\mathbf{p}_0 + \mathbf{t} = \mathbf{R}(d_0\hat{\mathbf{x}}_0) + \mathbf{t}, \tag{11.27}$$

where $\hat{\mathbf{x}}_j = \mathbf{K}_j^{-1}\mathbf{x}_j$ are the (local) ray direction vectors. Taking the cross product of the two (interchanged) sides with $\mathbf{t}$ in order to annihilate it on the right-hand side yields[8]

$$d_1[\mathbf{t}]_\times\hat{\mathbf{x}}_1 = d_0[\mathbf{t}]_\times\mathbf{R}\hat{\mathbf{x}}_0. \tag{11.28}$$

Taking the dot product of both sides with $\hat{\mathbf{x}}_1$ yields

$$d_0\hat{\mathbf{x}}_1^T([\mathbf{t}]_\times\mathbf{R})\hat{\mathbf{x}}_0 = d_1\hat{\mathbf{x}}_1^T[\mathbf{t}]_\times\hat{\mathbf{x}}_1 = 0, \tag{11.29}$$

because the right-hand side is a triple product with two identical entries. (Another way to say this is that the cross product matrix $[\mathbf{t}]_\times$ is skew symmetric and returns 0 when pre- and post-multiplied by the same vector.)

[8] The cross-product operator $[\,]_\times$ was introduced in (2.32).

We therefore arrive at the basic *epipolar constraint*

$$\hat{\mathbf{x}}_1^T \mathbf{E} \hat{\mathbf{x}}_0 = 0, \tag{11.30}$$

where

$$\mathbf{E} = [\mathbf{t}]_\times \mathbf{R} \tag{11.31}$$

is called the *essential matrix* (Longuet-Higgins 1981).

An alternative way to derive the epipolar constraint is to notice that, for the cameras to be oriented so that the rays $\hat{\mathbf{x}}_0$ and $\hat{\mathbf{x}}_1$ intersect in 3D at point $\mathbf{p}$, the vectors connecting the two camera centers $\mathbf{c}_1 - \mathbf{c}_0 = -\mathbf{R}_1^{-1}\mathbf{t}$ and the rays corresponding to pixels $\mathbf{x}_0$ and $\mathbf{x}_1$, namely $\mathbf{R}_j^{-1}\hat{\mathbf{x}}_j$, must be co-planar. This requires that the triple product

$$(\hat{\mathbf{x}}_0, \mathbf{R}^{-1}\hat{\mathbf{x}}_1, -\mathbf{R}^{-1}\mathbf{t}) = (\mathbf{R}\hat{\mathbf{x}}_0, \hat{\mathbf{x}}_1, -\mathbf{t}) = \hat{\mathbf{x}}_1 \cdot (\mathbf{t} \times \mathbf{R}\hat{\mathbf{x}}_0) = \hat{\mathbf{x}}_1^T([\mathbf{t}]_\times \mathbf{R})\hat{\mathbf{x}}_0 = 0. \tag{11.32}$$

Notice that the essential matrix $\mathbf{E}$ maps a point $\hat{\mathbf{x}}_0$ in image 0 into a line $\mathbf{l}_1 = \mathbf{E}\hat{\mathbf{x}}_0 = [\mathbf{t}]_\times \mathbf{R}\hat{\mathbf{x}}_0$ in image 1, because $\hat{\mathbf{x}}_1^T \mathbf{l}_1 = 0$ (Figure 11.11). All such lines must pass through the second *epipole* $\mathbf{e}_1$, which is therefore defined as the left singular vector of $\mathbf{E}$ with a 0 singular value, or, equivalently, the projection of the vector $\mathbf{t}$ into image 1. The dual (transpose) of these relationships gives us the epipolar line in the first image as $\mathbf{l}_0 = \mathbf{E}^T\hat{\mathbf{x}}_1$ and $\mathbf{e}_0$ as the zero-value right singular vector of $\mathbf{E}$.

Eight-point algorithm. Given this fundamental relationship (11.30), how can we use it to recover the camera motion encoded in the essential matrix $\mathbf{E}$? If we have N corresponding measurements $\{(\mathbf{x}_{i0}, \mathbf{x}_{i1})\}$, we can form N homogeneous equations in the nine elements of $\mathbf{E} = \{e_{00} \dots e_{22}\}$,

$$\begin{aligned} x_{i0}x_{i1}e_{00} \quad &+y_{i0}x_{i1}e_{01} \quad &+x_{i1}e_{02} \quad &+ \\ x_{i0}y_{i1}e_{00} \quad &+y_{i0}y_{i1}e_{11} \quad &+y_{i1}e_{12} \quad &+ \\ x_{i0}e_{20} \quad &+y_{i0}e_{21} \quad &+e_{22} \quad &= 0 \end{aligned} \tag{11.33}$$

where $\mathbf{x}_{ij} = (x_{ij}, y_{ij}, 1)$. This can be written more compactly as

$$[\mathbf{x}_{i1}\, \mathbf{x}_{i0}^T] \otimes \mathbf{E} = \mathbf{Z}_i \otimes \mathbf{E} = \mathbf{z}_i \cdot \mathbf{f} = 0, \tag{11.34}$$

where $\otimes$ indicates an element-wise multiplication and summation of matrix elements, and $\mathbf{z}_i$ and $\mathbf{f}$ are the vectorized forms of the $\mathbf{Z}_i = \hat{\mathbf{x}}_{i1}\hat{\mathbf{x}}_{i0}^T$ and $\mathbf{E}$ matrices.[9] Given $N \geq 8$ such equations, we can compute an estimate (up to scale) for the entries in $\mathbf{E}$ using an SVD.

In the presence of noisy measurements, how close is this estimate to being statistically optimal? If you look at the entries in (11.33), you can see that some entries are the products of image measurements such as $x_{i0}y_{i1}$ and others are direct image measurements (or even the identity). If the measurements have comparable noise, the terms that are products of measurements have their noise amplified by the other element in the product, which can lead to very poor scaling, e.g., an inordinately large influence of points with large coordinates (far away from the image center).

To counteract this trend, Hartley (1997a) suggests that the point coordinates should be translated and scaled so that their centroid lies at the origin and their variance is unity, i.e.,

$$\tilde{x}_i = s(x_i - \mu_x) \tag{11.35}$$

$$\tilde{y}_i = s(y_i - \mu_y) \tag{11.36}$$

[9] We use $\mathbf{f}$ instead of $\mathbf{e}$ to denote the vectorized form of $\mathbf{E}$ to avoid confusion with the epipoles $\mathbf{e}_j$.

such that $\sum_i \tilde{x}_i = \sum_i \tilde{y}_i = 0$ and $\sum_i \tilde{x}_i^2 + \sum_i \tilde{y}_i^2 = 2n$, where n is the number of points.[10]

Once the essential matrix $\tilde{\mathbf{E}}$ has been computed from the transformed coordinates $\{(\tilde{\mathbf{x}}_{i0}, \tilde{\mathbf{x}}_{i1})\}$, where $\tilde{\mathbf{x}}_{ij} = \mathbf{T}_j \hat{\mathbf{x}}_{ij}$ and $\mathbf{T}_j$ is the 3×3 matrix that implements the shift and scale operations in (11.35–11.36), the original essential matrix $\mathbf{E}$ can be recovered as

$$\mathbf{E} = \mathbf{T}_1^T \tilde{\mathbf{E}} \mathbf{T}_0. \tag{11.37}$$

In his paper, Hartley (1997a) compares the improvement due to his re-normalization strategy to alternative distance measures proposed by others such as Zhang (1998a,b) and concludes that his simple re-normalization in most cases is as effective as (or better than) alternative techniques. Torr and Fitzgibbon (2004) recommend a variant on this algorithm where the norm of the upper 2×2 sub-matrix of $\mathbf{E}$ is set to 1 and show that it has even better stability with respect to 2D coordinate transformations.

7-point algorithm. Because $\mathbf{E}$ is rank-deficient, it turns out that we actually only need seven correspondences of the form of Equation (11.34) instead of eight to estimate this matrix (Hartley 1994a; Torr and Murray 1997; Hartley and Zisserman 2004). The advantage of using fewer correspondences inside a RANSAC robust fitting stage is that fewer random samples need to be generated. From this set of seven homogeneous equations (which we can stack into a 7×9 matrix for SVD analysis), we can find two independent vectors, say $\mathbf{f}_0$ and $\mathbf{f}_1$ such that $\mathbf{z}_i \cdot \mathbf{f}_j = 0$. These two vectors can be converted back into 3×3 matrices $\mathbf{E}_0$ and $\mathbf{E}_1$, which span the solution space for

$$\mathbf{E} = \alpha \mathbf{E}_0 + (1 - \alpha) \mathbf{E}_1. \tag{11.38}$$

To find the correct value of α, we observe that $\mathbf{E}$ has a zero determinant, as it is rank deficient, and hence

$$|\alpha \mathbf{E}_0 + (1 - \alpha) \mathbf{E}_1| = 0. \tag{11.39}$$

This gives us a cubic equation in α, which has either one or three solutions (roots). Substituting these values into (11.38) to obtain $\mathbf{E}$, we can test this essential matrix against other unused feature correspondences to select the correct one.

The normalized "eight-point algorithm" (Hartley 1997a) and seven-point algorithm described above are not the only way to estimate the camera motion from correspondences. Additional variants include a five-point algorithm that requires finding the roots of a 10th degree polynomial (Nistér 2004) as well as variants that handle special (restricted) motions or scene structures, as discussed later on in this section. Because such algorithms use fewer points to compute their estimates, they are less sensitive to outliers when used as part of a random sampling (RANSAC) strategy.[11]

Recovering $\mathbf{t}$ and $\mathbf{R}$. Once an estimate for the essential matrix $\mathbf{E}$ has been recovered, the direction of the translation vector $\mathbf{t}$ can be estimated. Note that the absolute distance between the two cameras can never be recovered from pure image measurements alone, regardless of how many cameras or points are used. Knowledge about absolute camera and point positions or distances, often called *ground control points* in photogrammetry, is always required to establish the final scale, position, and orientation.

[10] More precisely, Hartley (1997a) suggests scaling the points "so that the average distance from the origin is equal to $\sqrt{2}$" but the heuristic of unit variance is faster to compute (does not require per-point square roots) and should yield comparable improvements.

[11] You can find an experimental comparison of a number of RANSAC variants at https://opencv.org/evaluating-opencvs-new-ransacs/.

To estimate this direction $\hat{\mathbf{t}}$, observe that under ideal noise-free conditions, the essential matrix $\mathbf{E}$ is singular, i.e., $\hat{\mathbf{t}}^T\mathbf{E} = 0$. This singularity shows up as a singular value of 0 when an SVD of $\mathbf{E}$ is performed,

$$\mathbf{E} = [\hat{\mathbf{t}}]_\times \mathbf{R} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \begin{bmatrix}\mathbf{u}_0 & \mathbf{u}_1 & \hat{\mathbf{t}}\end{bmatrix} \begin{bmatrix}1 & & \\ & 1 & \\ & & 0\end{bmatrix} \begin{bmatrix}\mathbf{v}_0^T \\ \mathbf{v}_1^T \\ \mathbf{v}_2^T\end{bmatrix}. \tag{11.40}$$

When $\mathbf{E}$ is computed from noisy measurements, the singular vector associated with the smallest singular value gives us $\hat{\mathbf{t}}$. (The other two singular values should be similar but are not, in general, equal to 1 because $\mathbf{E}$ is only computed up to an unknown scale.)

Once $\hat{\mathbf{t}}$ has been recovered, how can we estimate the corresponding rotation matrix $\mathbf{R}$? Recall that the cross-product operator $[\hat{\mathbf{t}}]_\times$ (2.32) projects a vector onto a set of orthogonal basis vectors that include $\hat{\mathbf{t}}$, zeros out the $\hat{\mathbf{t}}$ component, and rotates the other two by 90°,

$$[\hat{\mathbf{t}}]_\times = \mathbf{S}\mathbf{Z}\mathbf{R}_{90^\circ}\mathbf{S}^T = \begin{bmatrix}\mathbf{s}_0 & \mathbf{s}_1 & \hat{\mathbf{t}}\end{bmatrix} \begin{bmatrix}1 & & \\ & 1 & \\ & & 0\end{bmatrix} \begin{bmatrix}0 & -1 & \\ 1 & 0 & \\ & & 1\end{bmatrix} \begin{bmatrix}\mathbf{s}_0^T \\ \mathbf{s}_1^T \\ \hat{\mathbf{t}}^T\end{bmatrix}, \tag{11.41}$$

where $\hat{\mathbf{t}} = \mathbf{s}_0 \times \mathbf{s}_1$. From Equations (11.40 and 11.41), we get

$$\mathbf{E} = [\hat{\mathbf{t}}]_\times \mathbf{R} = \mathbf{S}\mathbf{Z}\mathbf{R}_{90^\circ}\mathbf{S}^T\mathbf{R} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T, \tag{11.42}$$

from which we can conclude that $\mathbf{S} = \mathbf{U}$. Recall that for a noise-free essential matrix, ($\mathbf{\Sigma} = \mathbf{Z}$), and hence

$$\mathbf{R}_{90^\circ}\mathbf{U}^T\mathbf{R} = \mathbf{V}^T \tag{11.43}$$

and

$$\mathbf{R} = \mathbf{U}\mathbf{R}_{90^\circ}^T\mathbf{V}^T. \tag{11.44}$$

Unfortunately, we only know both $\mathbf{E}$ and $\hat{\mathbf{t}}$ up to a sign. Furthermore, the matrices $\mathbf{U}$ and $\mathbf{V}$ are not guaranteed to be rotations (you can flip both their signs and still get a valid SVD). For this reason, we have to generate all four possible rotation matrices

$$\mathbf{R} = \pm\mathbf{U}\mathbf{R}_{\pm 90^\circ}^T\mathbf{V}^T \tag{11.45}$$

and keep the two whose determinant $|\mathbf{R}| = 1$. To disambiguate between the remaining pair of potential rotations, which form a *twisted pair* (Hartley and Zisserman 2004, p. 259), we need to pair them with both possible signs of the translation direction $\pm\hat{\mathbf{t}}$ and select the combination for which the largest number of points is seen in front of both cameras.[12]

The property that points must lie in front of the camera, i.e., at a positive distance along the viewing rays emanating from the camera, is known as *cheirality* (Hartley 1998). In addition to determining the signs of the rotation and translation, as described above, the cheirality (sign of the distances) of the points in a reconstruction can be used inside a RANSAC procedure (along with the reprojection errors) to distinguish between likely and unlikely configurations.[13] cheirality can also be used to transform projective reconstructions (Sections 11.3.3 and 11.3.4) into *quasi-affine* reconstructions (Hartley 1998).

[12] In the noise-free case, a single point suffices. It is safer, however, to test all or a sufficient subset of points, downweighting the ones that lie close to the plane at infinity, for which it is easy to get depth reversals.

[13] Note that as points get further away from a camera, i.e., closer toward the plane at infinity, errors in cheirality become more likely.

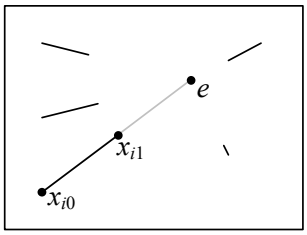


Figure 11.12 Pure translational camera motion results in visual motion where all the points move towards (or away from) a common *focus of expansion* (FOE) $\mathbf{e}$. They therefore satisfy the triple product condition $(\mathbf{x}_0, \mathbf{x}_1, \mathbf{e}) = \mathbf{e} \cdot (\mathbf{x}_0 \times \mathbf{x}_1) = 0$.

11.3.2 Special motions and structures

In certain situations, specially tailored algorithms can take advantage of known (or guessed) camera arrangements or 3D structures.

Pure translation (known rotation). In the case where we know the rotation, we can pre-rotate the points in the second image to match the viewing direction of the first. The resulting set of 3D points all move towards (or away from) the *focus of expansion* (FOE), as shown in Figure 11.12.[14] The resulting essential matrix $\mathbf{E}$ is (in the noise-free case) skew symmetric and so can be estimated more directly by setting $e_{ij} = -e_{ji}$ and $e_{ii} = 0$ in (11.33). Two points with non-zero parallax now suffice to estimate the FOE.

A more direct derivation of the FOE estimate can be obtained by minimizing the triple product

$$\sum_i (\mathbf{x}_{i0}, \mathbf{x}_{i1}, \mathbf{e})^2 = \sum_i ((\mathbf{x}_{i0} \times \mathbf{x}_{i1}) \cdot \mathbf{e})^2, \tag{11.46}$$

which is equivalent to finding the null space for the set of equations

$$(y_{i0} - y_{i1})e_0 + (x_{i1} - x_{i0})e_1 + (x_{i0}y_{i1} - y_{i0}x_{i1})e_2 = 0. \tag{11.47}$$

Note that, as in the eight-point algorithm, it is advisable to normalize the 2D points to have unit variance before computing this estimate.

In situations where a large number of points at infinity are available, e.g., when shooting outdoor scenes or when the camera motion is small compared to distant objects, this suggests an alternative RANSAC strategy for estimating the camera motion. First, pick a pair of points to estimate a rotation, hoping that both of the points lie at infinity (very far from the camera). Then, compute the FOE and check whether the residual error is small (indicating agreement with this rotation hypothesis) and whether the motions towards or away from the epipole (FOE) are all in the same direction (ignoring very small motions, which may be noise-contaminated).

Pure rotation. The case of pure rotation results in a degenerate estimate of the essential matrix $\mathbf{E}$ and of the translation direction $\hat{\mathbf{t}}$. Consider first the case of the rotation matrix being known. The estimates for the FOE will be degenerate, because $\mathbf{x}_{i0} \approx \mathbf{x}_{i1}$, and hence (11.47), is degenerate. A similar argument shows that the equations for the essential matrix (11.33) are also rank-deficient.

This suggests that it might be prudent before computing a full essential matrix to first compute a rotation estimate $\mathbf{R}$ using (8.32), potentially with just a small number of points, and then compute the residuals after rotating the points before proceeding with a full $\mathbf{E}$ computation.

[14] Fans of *Star Trek* and *Star Wars* will recognize this as the "jump to hyperdrive" visual effect.

Dominant planar structure. When a dominant plane is present in the scene, DEGENSAC, which tests whether too many correspondences are co-planar, can be used to recover the fundamental matrix more reliably than the seven-point algorithm (Chum, Werner, and Matas 2005).

As you can tell from the previous special cases, there exist many different specialized cases of two-frame structure-from-motion as well as many alternative appropriate techniques. The OpenGV library developed by Kneip and Furgale (2014) contains open-source implementations of many of these algorithms.[15]

11.3.3 Projective (uncalibrated) reconstruction

In many cases, such as when trying to build a 3D model from internet or legacy photos taken by unknown cameras without any EXIF tags, we do not know ahead of time the intrinsic calibration parameters associated with the input images. In such situations, we can still estimate a two-frame reconstruction, although the true metric structure may not be available, e.g., orthogonal lines or planes in the world may not end up being reconstructed as orthogonal.

Consider the derivations we used to estimate the essential matrix $\mathbf{E}$ (11.30–11.32). In the uncalibrated case, we do not know the calibration matrices $\mathbf{K}_j$, so we cannot use the normalized ray directions $\hat{\mathbf{x}}_j = \mathbf{K}_j^{-1}\mathbf{x}_j$. Instead, we have access only to the image coordinates $\mathbf{x}_j$, and so the essential matrix equation (11.30) becomes

$$\hat{\mathbf{x}}_1^T\mathbf{E}\hat{\mathbf{x}}_1 = \mathbf{x}_1^T\mathbf{K}_1^{-T}\mathbf{E}\mathbf{K}_0^{-1}\mathbf{x}_0 = \mathbf{x}_1^T\mathbf{F}\mathbf{x}_0 = 0, \tag{11.48}$$

where

$$\mathbf{F} = \mathbf{K}_1^{-T}\mathbf{E}\mathbf{K}_0^{-1} \tag{11.49}$$

is called the *fundamental matrix* (Faugeras 1992; Hartley, Gupta, and Chang 1992; Hartley and Zisserman 2004).

Like the essential matrix, the fundamental matrix is (in principle) rank two,

$$\mathbf{F} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \begin{bmatrix}\mathbf{u}_0 & \mathbf{u}_1 & \mathbf{e}_1\end{bmatrix}\begin{bmatrix}\sigma_0 & & \\ & \sigma_1 & \\ & & 0\end{bmatrix}\begin{bmatrix}\mathbf{v}_0^T \\ \mathbf{v}_1^T \\ \mathbf{e}_0^T\end{bmatrix}. \tag{11.50}$$

Its smallest left singular vector indicates the epipole $\mathbf{e}_1$ in the image 1 and its smallest right singular vector is $\mathbf{e}_0$ (Figure 11.11). The fundamental matrix can be factored into a skew-symmetric cross product matrix $[\mathbf{e}]_\times$ and a homography $\tilde{\mathbf{H}}$,

$$\mathbf{F} = [\mathbf{e}]_\times\tilde{\mathbf{H}}. \tag{11.51}$$

The homography $\tilde{\mathbf{H}}$, which in principle from (11.49) should equal

$$\tilde{\mathbf{H}} = \mathbf{K}_1^{-T}\mathbf{R}\mathbf{K}_0^{-1}, \tag{11.52}$$

cannot be uniquely recovered from $\mathbf{F}$, as any homography of the form $\tilde{\mathbf{H}}' = \tilde{\mathbf{H}} + \mathbf{e}\mathbf{v}^T$ results in the same $\mathbf{F}$ matrix. (Note that $[\mathbf{e}]_\times$ annihilates any multiple of $\mathbf{e}$.)

Any one of these valid homographies $\tilde{\mathbf{H}}$ maps some plane in the scene from one image to the other. It is not possible to tell in advance which one it is without either selecting four or more co-planar correspondences to compute $\tilde{\mathbf{H}}$ as part of the $\mathbf{F}$ estimation process (in a manner analogous to guessing a rotation for $\mathbf{E}$) or mapping all points in one image through $\tilde{\mathbf{H}}$ and seeing which ones

[15] https://laurentkneip.github.io/opengv

line up with their corresponding locations in the other. The resulting representation is often referred to as *plane plus parallax* (Kumar, Anandan, and Hanna 1994; Sawhney 1994) and is described in more detail in Section 2.1.4.

To create a *projective* reconstruction of the scene, we can pick any valid homography $\tilde{\mathbf{H}}$ that satisfies Equation (11.49). For example, following a technique analogous to Equations (11.40–11.44), we get

$$\mathbf{F} = [\mathbf{e}]_\times \tilde{\mathbf{H}} = \mathbf{S}\mathbf{Z}\mathbf{R}_{90^\circ}\mathbf{S}^T\tilde{\mathbf{H}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{11.53}$$

and hence

$$\tilde{\mathbf{H}} = \mathbf{U}\mathbf{R}_{90^\circ}^T\hat{\mathbf{\Sigma}}\mathbf{V}^T, \tag{11.54}$$

where $\hat{\mathbf{\Sigma}}$ is the singular value matrix with the smallest value replaced by a reasonable alternative (say, the middle value).[16] We can then form a pair of camera matrices

$$\mathbf{P}_0 = [\mathbf{I}|\mathbf{0}] \qquad \text{and} \qquad \mathbf{P}_0 = [\tilde{\mathbf{H}}|\mathbf{e}], \tag{11.55}$$

from which a projective reconstruction of the scene can be computed using triangulation (Section 11.2.4).

While the projective reconstruction may not be useful on its own, it can often be *upgraded* to an affine or metric reconstruction, as described below. Even without this step, however, the fundamental matrix $\mathbf{F}$ can be very useful in finding additional correspondences, as they must all lie on corresponding epipolar lines, i.e., any feature $\mathbf{x}_0$ in image 0 must have its correspondence lying on the associated epipolar line $\mathbf{l}_1 = \mathbf{F}\mathbf{x}_0$ in image 1, assuming that the point motions are due to a rigid transformation.

11.3.4 Self-calibration

The results of structure from motion computation are much more useful if a *metric* reconstruction is obtained, i.e., one in which parallel lines are parallel, orthogonal walls are at right angles, and the reconstructed model is a scaled version of reality. Over the years, a large number of *self-calibration* (or *auto-calibration*) techniques have been developed for converting a projective reconstruction into a metric one, which is equivalent to recovering the unknown calibration matrices $\mathbf{K}_j$ associated with each image (Hartley and Zisserman 2004; Moons, Van Gool, and Vergauwen 2010).

In situations where additional information is known about the scene, different methods may be employed. For example, if there are parallel lines in the scene, three or more vanishing points, which are the images of points at infinity, can be used to establish the homography for the plane at infinity, from which focal lengths and rotations can be recovered. If two or more finite *orthogonal* vanishing points have been observed, the single-image calibration method based on vanishing points (Section 11.1.1) can be used instead.

In the absence of such external information, it is not possible to recover a fully parameterized independent calibration matrix $\mathbf{K}_j$ for each image from correspondences alone. To see this, consider the set of all camera matrices $\mathbf{P}_j = \mathbf{K}_j[\mathbf{R}_j|\mathbf{t}_j]$ projecting world coordinates $\mathbf{p}_i = (X_i, Y_i, Z_i, W_i)$ into screen coordinates $\mathbf{x}_{ij} \sim \mathbf{P}_j\mathbf{p}_i$. Now consider transforming the 3D scene $\{\mathbf{p}_i\}$ through an arbitrary 4×4 projective transformation $\tilde{\mathbf{H}}$, yielding a new model consisting of points $\mathbf{p}'_i = \tilde{\mathbf{H}}\mathbf{p}_i$. Post-multiplying each $\mathbf{P}_j$ matrix by $\tilde{\mathbf{H}}^{-1}$ still produces the same screen coordinates and a new set calibration matrices can be computed by applying RQ decomposition to the new camera matrix $\mathbf{P}'_j = \mathbf{P}_j\tilde{\mathbf{H}}^{-1}$.

[16] Hartley and Zisserman (2004, p. 256) recommend using $\tilde{\mathbf{H}} = [\mathbf{e}]_\times\mathbf{F}$ (Luong and Viéville 1996), which places the camera on the plane at infinity.

For this reason, all self-calibration methods assume some restricted form of the calibration matrix, either by setting or equating some of their elements or by assuming that they do not vary over time. While most of the techniques discussed by Hartley and Zisserman (2004); Moons, Van Gool, and Vergauwen (2010) require three or more frames, in this section we present a simple technique that can recover the focal lengths (f_0, f_1) of both images from the fundamental matrix $\mathbf{F}$ in a two-frame reconstruction (Hartley and Zisserman 2004, p. 472).

To accomplish this, we assume that the camera has zero skew, a known aspect ratio (usually set to 1), and a known image center, as in Equation (2.59). How reasonable is this assumption in practice? The answer, as with many questions, is "it depends".

If absolute metric accuracy is required, as in photogrammetry applications, it is imperative to pre-calibrate the cameras using one of the techniques from Section 11.1 and to use ground control points to pin down the reconstruction. If instead, we simply wish to reconstruct the world for visualization or image-based rendering applications, as in the Photo Tourism system of Snavely, Seitz, and Szeliski (2006), this assumption is quite reasonable in practice.

Most cameras today have square pixels and an image center near the middle of the image, and are much more likely to deviate from a simple camera model due to radial distortion (Section 11.1.4), which should be compensated for whenever possible. The biggest problems occur when images have been cropped off-center, in which case the image center will no longer be in the middle, or when perspective pictures have been taken of a different picture, in which case a general camera matrix becomes necessary.[17]

Given these caveats, the two-frame focal length estimation algorithm based on the Kruppa equations developed by Hartley and Zisserman (2004, p. 456) proceeds as follows. Take the left and right singular vectors $\{\mathbf{u}_0, \mathbf{u}_1, \mathbf{v}_0, \mathbf{v}_1\}$ of the fundamental matrix $\mathbf{F}$ (11.50) and their associated singular values $\{\sigma_0, \sigma_1\}$ and form the following set of equations:

$$\frac{\mathbf{u}_1^T \mathbf{D}_0 \mathbf{u}_1}{\sigma_0^2 \mathbf{v}_0^T \mathbf{D}_1 \mathbf{v}_0} = -\frac{\mathbf{u}_0^T \mathbf{D}_0 \mathbf{u}_1}{\sigma_0 \sigma_1 \mathbf{v}_0^T \mathbf{D}_1 \mathbf{v}_1} = \frac{\mathbf{u}_0^T \mathbf{D}_0 \mathbf{u}_0}{\sigma_1^2 \mathbf{v}_1^T \mathbf{D}_1 \mathbf{v}_1}, \tag{11.56}$$

where the two matrices

$$\mathbf{D}_j = \mathbf{K}_j \mathbf{K}_j^T = \mathrm{diag}(f_j^2, f_j^2, 1) = \begin{bmatrix} f_j^2 & & \\ & f_j^2 & \\ & & 1 \end{bmatrix} \tag{11.57}$$

encode the unknown focal lengths. For simplicity, let us rewrite each of the numerators and denominators in (11.56) as

$$e_{ij0}(f_0^2) = \mathbf{u}_i^T \mathbf{D}_0 \mathbf{u}_j = a_{ij} + b_{ij} f_0^2, \tag{11.58}$$

$$e_{ij1}(f_1^2) = \sigma_i \sigma_j \mathbf{v}_i^T \mathbf{D}_1 \mathbf{v}_j = c_{ij} + d_{ij} f_1^2. \tag{11.59}$$

Notice that each of these is affine (linear plus constant) in either f_0^2 or f_1^2. Hence, we can cross-multiply these equations to obtain quadratic equations in f_j^2, which can readily be solved. (See also the work by Bougnoux (1998) and Kanatani and Matsunaga (2000) for some alternative formulations.)

An alternative solution technique is to observe that we have a set of three equations related by an unknown scalar λ, i.e.,

$$e_{ij0}(f_0^2) = \lambda e_{ij1}(f_1^2) \tag{11.60}$$

[17] In Photo Tourism, our system registered photographs of an information sign outside Notre Dame with real pictures of the cathedral.

(Richard Hartley, personal communication, July 2009). These can readily be solved to yield $(f_0^2, \lambda f_1^2, \lambda)$ and hence (f_0, f_1).

How well does this approach work in practice? There are certain degenerate configurations, such as when there is no rotation or when the optical axes intersect, when it does not work at all. (In such a situation, you can vary the focal lengths of the cameras and obtain a deeper or shallower reconstruction, which is an example of a *bas-relief ambiguity* (Section 11.4.5).) Hartley and Zisserman (2004) recommend using techniques based on three or more frames. However, if you find two images for which the estimates of $(f_0^2, \lambda f_1^2, \lambda)$ are well conditioned, they can be used to initialize a more complete bundle adjustment of all the parameters (Section 11.4.2). An alternative, which is often used in systems such as Photo Tourism, is to use camera EXIF tags or generic default values to initialize focal length estimates and refine them as part of bundle adjustment.

11.3.5 *Application*: View morphing

An interesting application of basic two-frame structure from motion is *view morphing* (also known as *view interpolation*, see Section 14.1), which can be used to generate a smooth 3D animation from one view of a 3D scene to another (Chen and Williams 1993; Seitz and Dyer 1996).

To create such a transition, you must first smoothly interpolate the camera matrices, i.e., the camera positions, orientations, and focal lengths. While simple linear interpolation can be used (representing rotations as quaternions (Section 2.1.3)), a more pleasing effect is obtained by *easing in* and *easing out* the camera parameters, e.g., using a raised cosine, as well as moving the camera along a more circular trajectory (Snavely, Seitz, and Szeliski 2006).

To generate in-between frames, either a full set of 3D correspondences needs to be established (Section 12.3) or 3D models (proxies) must be created for each reference view. Section 14.1 describes several widely used approaches to this problem. One of the simplest is to just triangulate the set of matched feature points in each image, e.g., using Delaunay triangulation. As the 3D points are re-projected into their intermediate views, pixels can be mapped from their original source images to their new views using affine or projective mapping (Szeliski and Shum 1997). The final image is then composited using a linear blend of the two reference images, as with usual morphing (Section 3.6.3).

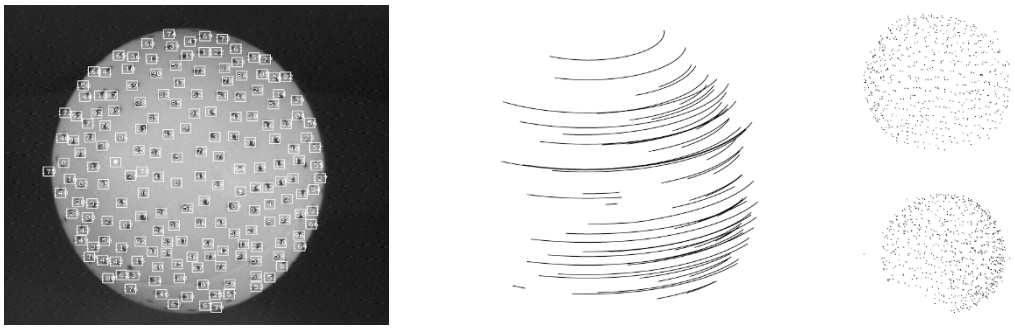
11.4 Multi-frame structure from motion

While two-frame techniques are useful for reconstructing sparse geometry from stereo image pairs and for initializing larger-scale 3D reconstructions, most applications can benefit from the much larger number of images that are usually available in photo collections and videos of scenes.

In this section, we briefly review an older technique called *factorization*, which can provide useful solutions for short video sequences, and then turn to the more commonly used *bundle adjustment* approach, which uses non-linear least squares to obtain optimal solutions under general camera configurations.

11.4.1 Factorization

When processing video sequences, we often get extended *feature tracks* (Section 7.1.5) from which it is possible to recover the structure and motion using a process called *factorization*. Consider the tracks generated by a rotating ping pong ball, which has been marked with dots to make its shape



(a) (b) (c)

Figure 11.13 3D reconstruction of a rotating ping pong ball using factorization (Tomasi and Kanade 1992) © 1992 Springer: (a) sample image with tracked features overlaid; (b) subsampled feature motion stream; (c) two views of the reconstructed 3D model.

and motion more discernable (Figure 11.13). We can readily see from the shape of the tracks that the moving object must be a sphere, but how can we infer this mathematically?

It turns out that, under orthography or related models we discuss below, the shape and motion can be recovered simultaneously using a singular value decomposition (Tomasi and Kanade 1992). The details of how to do this are presented in the paper by Tomasi and Kanade (1992) and also in the first edition of this book (Szeliski 2010, Section 7.3).

Once the rotation matrices and 3D point locations have been recovered, there still exists a bas-relief ambiguity, i.e., we can never be sure if the object is rotating left to right or if its depth reversed version is moving the other way. (This can be seen in the classic rotating Necker Cube visual illusion.) Additional cues, such as the appearance and disappearance of points, or perspective effects, both of which are discussed below, can be used to remove this ambiguity.

For motion models other than pure orthography, e.g., for scaled orthography or para-perspective, the approach above must be extended in the appropriate manner. Such techniques are relatively straightforward to derive from first principles; more details can be found in papers that extend the basic factorization approach to these more flexible models (Poelman and Kanade 1997). Additional extensions of the original factorization algorithm include multi-body rigid motion (Costeira and Kanade 1995), sequential updates to the factorization (Morita and Kanade 1997), the addition of lines and planes (Morris and Kanade 1998), and re-scaling the measurements to incorporate individual location uncertainties (Anandan and Irani 2002).

A disadvantage of factorization approaches is that they require a complete set of tracks, i.e., each point must be visible in each frame, for the factorization approach to work. Tomasi and Kanade (1992) deal with this problem by first applying factorization to smaller denser subsets and then using known camera (motion) or point (structure) estimates to *hallucinate* additional missing values, which allows them to incrementally incorporate more features and cameras. Huynh, Hartley, and Heyden (2003) extend this approach to view missing data as special cases of outliers. Buchanan and Fitzgibbon (2005) develop fast iterative algorithms for performing large matrix factorizations with missing data. The general topic of principal component analysis (PCA) with missing data also appears in other computer vision problems (Shum, Ikeuchi, and Reddy 1995; De la Torre and Black 2003; Gross, Matthews, and Baker 2006; Torresani, Hertzmann, and Bregler 2008; Vidal, Ma, and Sastry 2016).

Perspective and projective factorization

Another disadvantage of regular factorization is that it cannot deal with perspective cameras. One way to get around this problem is to perform an initial affine (e.g., orthographic) reconstruction and to then correct for the perspective effects in an iterative manner (Christy and Horaud 1996). This algorithm usually converges in three to five iterations, with the majority of the time spent in the SVD computation.

An alternative approach, which does not assume partially calibrated cameras (known image center, square pixels, and zero skew) is to perform a fully *projective* factorization (Sturm and Triggs 1996; Triggs 1996). In this case, the inclusion of the third row of the camera matrix in the measurement matrix is equivalent to multiplying each reconstructed measurement $\mathbf{x}_{ji} = \mathbf{M}_j\mathbf{p}_i$ by its inverse (projective) depth $\eta_{ji} = d_{ji}^{-1} = 1/(\mathbf{P}_{j2}\mathbf{p}_i)$ or, equivalently, multiplying each measured position by its projective depth d_{ji}. In the original paper by Sturm and Triggs (1996), the projective depths d_{ji} are obtained from two-frame reconstructions, while in later work (Triggs 1996; Oliensis and Hartley 2007), they are initialized to $d_{ji} = 1$ and updated after each iteration. Oliensis and Hartley (2007) present an update formula that is guaranteed to converge to a fixed point. None of these authors suggest actually estimating the third row of $\mathbf{P}_j$ as part of the projective depth computations. In any case, it is unclear when a fully projective reconstruction would be preferable to a partially calibrated one, especially if they are being used to initialize a full bundle adjustment of all the parameters.

One of the attractions of factorization methods is that they provide a "closed form" (sometimes called a "linear") method to initialize iterative techniques such as bundle adjustment. An alternative initialization technique is to estimate the homographies corresponding to some common plane seen by all the cameras (Rother and Carlsson 2002). In a calibrated camera setting, this can correspond to estimating consistent rotations for all of the cameras, for example, using matched vanishing points (Antone and Teller 2002). Once these have been recovered, the camera positions can then be obtained by solving a linear system (Antone and Teller 2002; Rother and Carlsson 2002; Rother 2003).

11.4.2 Bundle adjustment

As we have mentioned several times before, the most accurate way to recover structure and motion is to perform robust non-linear minimization of the measurement (re-projection) errors, which is commonly known in the photogrammetry (and now computer vision) communities as *bundle adjustment*.[18] Triggs, McLauchlan *et al.* (1999) provide an excellent overview of this topic, including its historical development, pointers to the photogrammetry literature (Slama 1980; Atkinson 1996; Kraus 1997), and subtle issues with gauge ambiguities. The topic is also treated in depth in textbooks and surveys on multi-view geometry (Faugeras and Luong 2001; Hartley and Zisserman 2004; Moons, Van Gool, and Vergauwen 2010).

We have already introduced the elements of bundle adjustment in our discussion on iterative pose estimation (Section 11.2.2), i.e., Equations (11.14–11.20) and Figure 11.7. The biggest difference between these formulas and full bundle adjustment is that our feature location measurements $\mathbf{x}_{ij}$ now depend not only on the point (track) index i but also on the camera pose index j,

$$\mathbf{x}_{ij} = \mathbf{f}(\mathbf{p}_i, \mathbf{R}_j, \mathbf{c}_j, \mathbf{K}_j), \tag{11.61}$$

[18] The term "bundle" refers to the bundles of rays connecting camera centers to 3D points and the term "adjustment" refers to the iterative minimization of re-projection error. Alternative terms for this in the vision community include *optimal motion estimation* (Weng, Ahuja, and Huang 1993) and *non-linear least squares* (Appendix A.3) (Taylor, Kriegman, and Anandan 1991; Szeliski and Kang 1994).

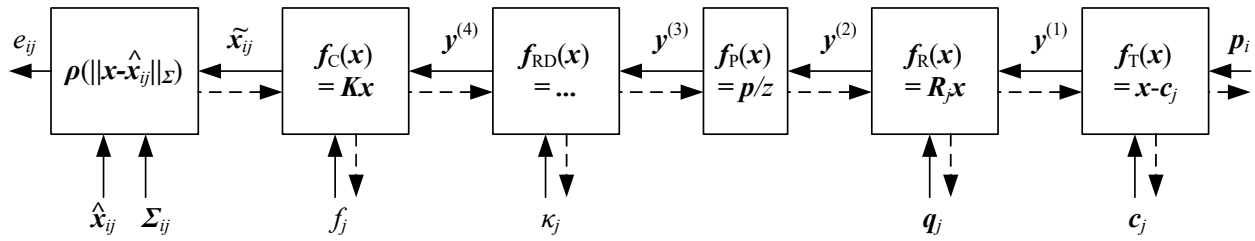


Figure 11.14 A set of chained transforms for projecting a 3D point $\mathbf{p}_i$ into a 2D measurement $\mathbf{x}_{ij}$ through a series of transformations $\mathbf{f}^{(k)}$, each of which is controlled by its own set of parameters. The dashed lines indicate the flow of information as partial derivatives are computed during a backward pass. The formula for the radial distortion function is $\mathbf{f}_{\mathrm{RD}}(\mathbf{x}) = (1 + \kappa_1 r^2 + \kappa_2 r^4)\mathbf{x}$.

and that the 3D point positions $\mathbf{p}_i$ are also being simultaneously updated. In addition, it is common to add a stage for radial distortion parameter estimation (2.78),

$$\mathbf{f}_{\mathrm{RD}}(\mathbf{x}) = (1 + \kappa_1 r^2 + \kappa_2 r^4)\mathbf{x}, \tag{11.62}$$

if the cameras being used have not been pre-calibrated, as shown in Figure 11.14.

While most of the boxes (transforms) in Figure 11.14 have previously been explained (11.19), the leftmost box has not. This box performs a robust comparison of the predicted and measured 2D locations $\hat{\mathbf{x}}_{ij}$ and $\tilde{\mathbf{x}}_{ij}$ after re-scaling by the measurement noise covariance $\mathbf{\Sigma}_{ij}$. In more detail, this operation can be written as

$$\mathbf{r}_{ij} = \tilde{\mathbf{x}}_{ij} - \hat{\mathbf{x}}_{ij}, \tag{11.63}$$

$$s_{ij}^2 = \mathbf{r}_{ij}^T \mathbf{\Sigma}_{ij}^{-1} \mathbf{r}_{ij}, \tag{11.64}$$

$$e_{ij} = \hat{\rho}(s_{ij}^2), \tag{11.65}$$

where $\hat{\rho}(r^2) = \rho(r)$. The corresponding Jacobians (partial derivatives) can be written as

$$\frac{\partial e_{ij}}{\partial s_{ij}^2} = \hat{\rho}'(s_{ij}^2), \tag{11.66}$$

$$\frac{\partial s_{ij}^2}{\partial \tilde{\mathbf{x}}_{ij}} = \mathbf{\Sigma}_{ij}^{-1} \mathbf{r}_{ij}. \tag{11.67}$$

The advantage of the chained representation introduced above is that it not only makes the computations of the partial derivatives and Jacobians simpler but it can also be adapted to any camera configuration. Consider for example a pair of cameras mounted on a robot that is moving around in the world, as shown in Figure 11.15a. By replacing the rightmost two transformations in Figure 11.14 with the transformations shown in Figure 11.15b, we can simultaneously recover the position of the robot at each time and the calibration of each camera with respect to the rig, in addition to the 3D structure of the world.

11.4.3 Exploiting sparsity

Large bundle adjustment problems, such as those involving reconstructing 3D scenes from thousands of internet photographs (Snavely, Seitz, and Szeliski 2008b; Agarwal, Furukawa *et al.* 2010, 2011; Snavely, Simon *et al.* 2010), can require solving non-linear least squares problems with millions of

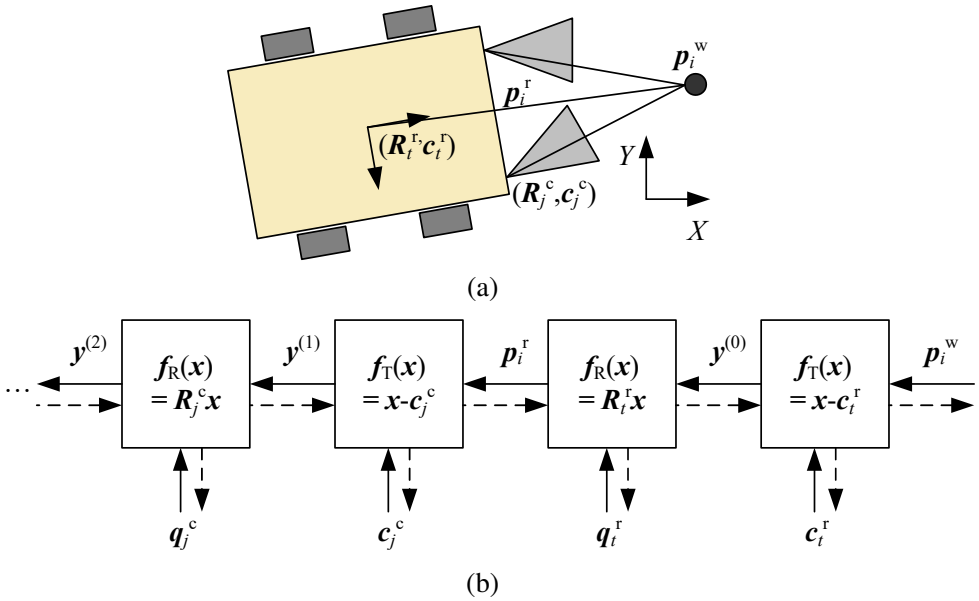


Figure 11.15 A camera rig and its associated transform chain. (a) As the mobile rig (robot) moves around in the world, its pose with respect to the world at time t is captured by $(\mathbf{R}_t^{\mathrm{r}}, \mathbf{c}_t^{\mathrm{r}})$. Each camera's pose with respect to the rig is captured by $(\mathbf{R}_j^{\mathrm{c}}, \mathbf{c}_j^{\mathrm{c}})$. (b) A 3D point with world coordinates $\mathbf{p}_i^{\mathrm{w}}$ is first transformed into rig coordinates $\mathbf{p}_i^{\mathrm{r}}$, and then through the rest of the camera-specific chain, as shown in Figure 11.14.

measurements (feature matches) and tens of thousands of unknown parameters (3D point positions and camera poses). Unless some care is taken, these kinds of problem can become intractable, because the (direct) solution of dense least squares problems is cubic in the number of unknowns.

Fortunately, structure from motion is a *bipartite* problem in structure and motion. Each feature point $\mathbf{x}_{ij}$ in a given image depends on one 3D point position $\mathbf{p}_i$ and one 3D camera pose $(\mathbf{R}_j, \mathbf{c}_j)$. This is illustrated in Figure 11.16a, where each circle (1–9) indicates a 3D point, each square (A–D) indicates a camera, and lines (edges) indicate which points are visible in which cameras (2D features). If the values for all the points are known or fixed, the equations for all the cameras become independent, and vice versa.

If we order the structure variables before the motion variables in the Hessian matrix $\mathbf{A}$ (and hence also the right-hand side vector $\mathbf{b}$), we obtain a structure for the Hessian shown in Figure 11.16c.[19] When such a system is solved using sparse Cholesky factorization (see Appendix A.4) (Björck 1996; Golub and Van Loan 1996), the *fill-in* occurs in the smaller motion Hessian A_{cc} (Szeliski and Kang 1994; Triggs, McLauchlan *et al.* 1999; Hartley and Zisserman 2004; Lourakis and Argyros 2009; Engels, Stewénius, and Nistér 2006). More recent papers (Byröd and Åström 2009; Jeong, Nistér *et al.* 2010; Agarwal, Snavely *et al.* 2010; Jeong, Nistér *et al.* 2012) explore the use of iterative (conjugate gradient) techniques for the solution of bundle adjustment problems. Other papers explore the use of parallel multicore algorithms (Wu, Agarwal *et al.* 2011).

In more detail, the *reduced* motion Hessian is computed using the *Schur complement*,

$$\mathbf{A}'_{\mathrm{CC}} = \mathbf{A}_{\mathrm{CC}} - \mathbf{A}_{\mathrm{PC}}^T \mathbf{A}_{\mathrm{PP}}^{-1} \mathbf{A}_{\mathrm{pc}}, \tag{11.68}$$

where $\mathbf{A}_{\mathrm{PP}}$ is the point (structure) Hessian (the top left block of Figure 11.16c), $\mathbf{A}_{\mathrm{PC}}$ is the point-

[19] This ordering is preferable when there are fewer cameras than 3D points, which is the usual case. The exception is when we are tracking a small number of points through many video frames, in which case this ordering should be reversed.

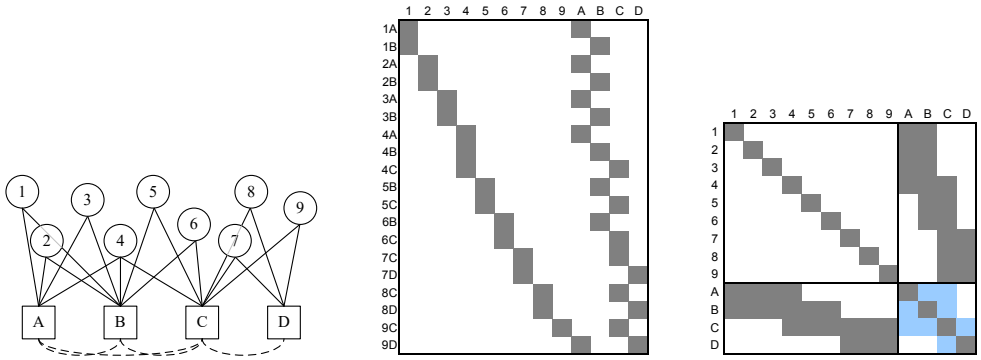


Figure 11.16 (a) Bipartite graph for a toy structure from motion problem and (b) its associated Jacobian $\mathbf{J}$ and (c) Hessian $\mathbf{A}$. Numbers indicate 3D points and letters indicate cameras. The dashed arcs and light blue squares indicate the fill-in that occurs when the structure (point) variables are eliminated.

camera Hessian (the top right block), and $\mathbf{A}_{\mathrm{CC}}$ and $\mathbf{A}'_{\mathrm{CC}}$ are the motion Hessians before and after the point variable elimination (the bottom right block of Figure 11.16c). Notice that $\mathbf{A}'_{\mathrm{CC}}$ has a non-zero entry between two cameras if they see any 3D point in common. This is indicated with dashed arcs in Figure 11.16a and light blue squares in Figure 11.16c.

Whenever there are global parameters present in the reconstruction algorithm, such as camera intrinsics that are common to all of the cameras, or camera rig calibration parameters such as those shown in Figure 11.15, they should be ordered last (placed along the right and bottom edges of $\mathbf{A}$) to reduce fill-in.

Engels, Stewénius, and Nistér (2006) provide a nice recipe for sparse bundle adjustment, including all the steps needed to initialize the iterations, as well as typical computation times for a system that uses a fixed number of backward-looking frames in a real-time setting. They also recommend using homogeneous coordinates for the structure parameters $\mathbf{p}_i$, which is a good idea, as it avoids numerical instabilities for points near infinity.

Bundle adjustment is now the standard method of choice for most structure-from-motion problems and is commonly applied to problems with hundreds of weakly calibrated images and tens of thousands of points. (Much larger problems are commonly solved in photogrammetry and aerial imagery, but these are usually carefully calibrated and make use of surveyed ground control points.) However, as the problems become larger, it becomes impractical to re-solve full bundle adjustment problems at each iteration.

One approach to dealing with this problem is to use an incremental algorithm, where new cameras are added over time. (This makes particular sense if the data is being acquired from a video camera or moving vehicle (Nistér, Naroditsky, and Bergen 2006; Pollefeys, Nistér *et al.* 2008).) A Kalman filter can be used to incrementally update estimates as new information is acquired. Unfortunately, such sequential updating is only statistically optimal for linear least squares problems.

For non-linear problems such as structure from motion, an extended Kalman filter, which linearizes measurement and update equations around the current estimate, needs to be used (Gelb 1974; Viéville and Faugeras 1990). To overcome this limitation, several passes can be made through the data (Azarbayejani and Pentland 1995). Because points disappear from view (and old cameras become irrelevant), a *variable state dimension filter* (VSDF) can be used to adjust the set of state variables over time, for example, by keeping only cameras and point tracks seen in the last k frames (McLauchlan 2000). A more flexible approach to using a fixed number of frames is to

propagate corrections backwards through points and cameras until the changes on parameters are below a threshold (Steedly and Essa 2001). Variants of these techniques, including methods that use a fixed window for bundle adjustment (Engels, Stewénius, and Nistér 2006) or select keyframes for doing full bundle adjustment (Klein and Murray 2008) are now commonly used in simultaneous localization and mapping (SLAM) and augmented-reality applications, as discussed in Section 11.5.

When maximum accuracy is required, it is still preferable to perform a full bundle adjustment over all the frames. To control the resulting computational complexity, one approach is to lock together subsets of frames into locally rigid configurations and to optimize the relative positions of these cluster (Steedly, Essa, and Dellaert 2003). A different approach is to select a smaller number of frames to form a *skeletal set* that still spans the whole dataset and produces reconstructions of comparable accuracy (Snavely, Seitz, and Szeliski 2008b). We describe this latter technique in more detail in Section 11.4.6, where we discuss applications of structure from motion to large image sets. Additional techniques for efficiently solving large structure from motion and SLAM systems can be found in the survey by Dellaert and Kaess (2017); Dellaert (2021).

While bundle adjustment and other robust non-linear least squares techniques are the methods of choice for most structure-from-motion problems, they suffer from initialization problems, i.e., they can get stuck in local energy minima if not started sufficiently close to the global optimum. Many systems try to mitigate this by being conservative in what reconstruction they perform early on and which cameras and points they add to the solution (Section 11.4.6). An alternative, however, is to re-formulate the problem using a norm that supports the computation of global optima.

Kahl and Hartley (2008) describe techniques for using L_∞ norms in geometric reconstruction problems. The advantage of such norms is that globally optimal solutions can be efficiently computed using second-order cone programming (SOCP). The disadvantage is that L_∞ norms are particularly sensitive to outliers and so must be combined with good outlier rejection techniques before they can be used.

A large number of high-quality open source bundle adjustment packages have been developed, including the Ceres Solver,[20] Multicore Bundle Adjustment (Wu, Agarwal *et al.* 2011),[21] the Sparse Levenberg-Marquardt based non-linear least squares optimizer and bundle adjuster,[22] and OpenSfM.[23] You can find more pointers to open-source software in Appendix Appendix C.2 and reviews of open-source and commercial photogrammetry software[24] as well as examples of their application[25] on the web.

11.4.4 *Application*: Match move

One of the neatest applications of structure from motion is to estimate the 3D motion of a video or film camera, along with the geometry of a 3D scene, in order to superimpose 3D graphics or computer-generated images (CGI) on the scene. In the visual effects industry, this is known as the *match move* problem (Roble 1999), as the motion of the synthetic 3D camera used to render the graphics must be *matched* to that of the real-world camera. For very small motions, or motions involving pure camera rotations, one or two tracked points can suffice to compute the necessary visual motion. For planar surfaces moving in 3D, four points are needed to compute the homography,

[20] http://ceres-solver.org
[21] https://grail.cs.washington.edu/projects/mcba
[22] https://github.com/chzach/SSBA
[23] https://www.opensfm.org
[24] https://peterfalkingham.com/2020/07/10/free-and-commercial-photogrammetry-software-review-2020
[25] https://beforesandafters.com/2020/07/06/tales-from-on-set-lidar-scanning-for-joker-and-john-wick-3, https://rd.nytimes.com/projects/reconstructing-journalistic-scenes-in-3d

which can then be used to insert planar overlays, e.g., to replace the contents of advertising billboards during sporting events.

The general version of this problem requires the estimation of the full 3D camera pose along with the focal length (zoom) of the lens and potentially its radial distortion parameters (Roble 1999). When the 3D structure of the scene is known ahead of time, pose estimation techniques such as *view correlation* (Bogart 1991) or *through-the-lens camera control* (Gleicher and Witkin 1992) can be used, as described in Section 11.4.4.

For more complex scenes, it is usually preferable to recover the 3D structure simultaneously with the camera motion using structure-from-motion techniques. The trick with using such techniques is that to prevent any visible jitter between the synthetic graphics and the actual scene, features must be tracked to very high accuracy and ample feature tracks must be available in the vicinity of the insertion location. Some of today's best known match move software packages, such as the *boujou* package from 2d3, which won an Emmy award in 2002, originated in structure-from-motion research in the computer vision community (Fitzgibbon and Zisserman 1998).

11.4.5 Uncertainty and ambiguities

Because structure from motion involves the estimation of so many highly coupled parameters, often with no known "ground truth" components, the estimates produced by structure from motion algorithms can often exhibit large amounts of uncertainty (Szeliski and Kang 1997; Wilson and Wehrwein 2020). An example of this is the classic *bas-relief ambiguity*, which makes it hard to simultaneously estimate the 3D depth of a scene and the amount of camera motion (Oliensis 2005).[26]

As mentioned before, a unique coordinate frame and scale for a reconstructed scene cannot be recovered from monocular visual measurements alone. (When a stereo rig is used, the scale can be recovered if we know the distance (baseline) between the cameras.) This seven-degree-of-freedom (coordinate frame and scale) *gauge ambiguity* makes it tricky to compute the covariance matrix associated with a 3D reconstruction (Triggs, McLauchlan *et al.* 1999; Kanatani and Morris 2001). A simple way to compute a covariance matrix that ignores the gauge freedom (indeterminacy) is to throw away the seven smallest eigenvalues of the information matrix (inverse covariance), whose values are equivalent to the problem Hessian $\mathbf{A}$ up to noise scaling (see Section 8.1.4 and Appendix B.6). After we do this, the resulting matrix can be inverted to obtain an estimate of the parameter covariance.

Szeliski and Kang (1997) use this approach to visualize the largest directions of variation in typical structure from motion problems. Not surprisingly, they find that, ignoring the gauge freedoms, the greatest uncertainties for problems such as observing an object from a small number of nearby viewpoints are in the depths of the 3D structure relative to the extent of the camera motion.[27]

It is also possible to estimate *local* or *marginal* uncertainties for individual parameters, which corresponds simply to taking block sub-matrices from the full covariance matrix. Under certain conditions, such as when the camera poses are relatively certain compared to 3D point locations, such uncertainty estimates can be meaningful. However, in many cases, individual uncertainty measures can mask the extent to which reconstruction errors are correlated, which is why looking at the first few modes of greatest joint variation can be helpful.

[26] Bas-relief refers to a kind of sculpture in which objects, often on ornamental friezes, are sculpted with less depth than they actually occupy. When lit from above by sunlight, they appear to have true 3D depth because of the ambiguity between relative depth and the angle of the illuminant (Section 13.1.1).

[27] A good way to minimize the amount of such ambiguities is to use wide field of view cameras (Antone and Teller 2002; Levin and Szeliski 2006).

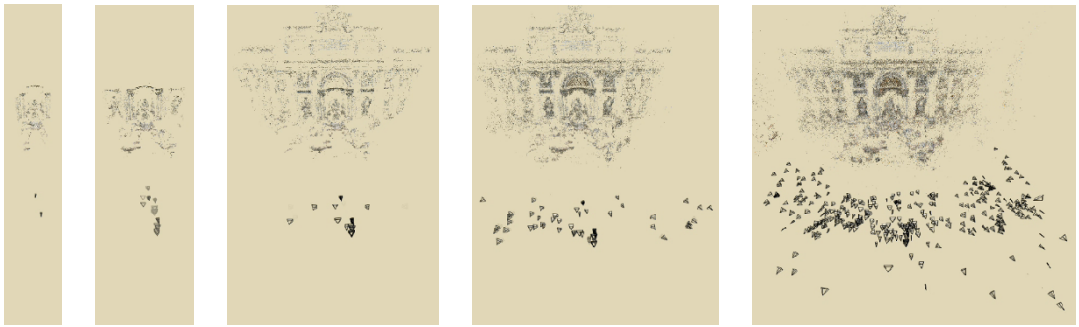


Figure 11.17 Incremental structure from motion (Snavely, Seitz, and Szeliski 2006) © 2006 ACM. Starting with an initial two-frame reconstruction of Trevi Fountain, batches of images are added using pose estimation, and their positions (along with the 3D model) are refined using bundle adjustment.

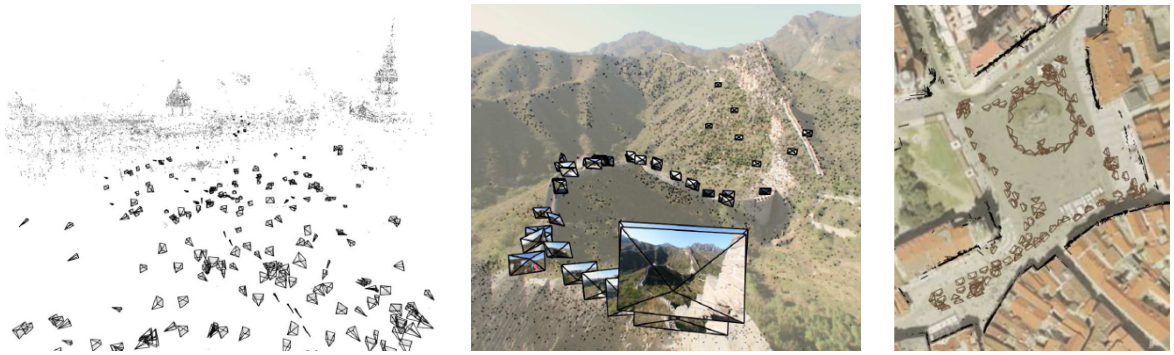
The other way in which gauge ambiguities affect structure from motion and, in particular, bundle adjustment is that they make the system Hessian matrix $\mathbf{A}$ rank-deficient and hence impossible to invert. A number of techniques have been proposed to mitigate this problem (Triggs, McLauchlan *et al.* 1999; Bartoli 2003). In practice, however, it appears that simply adding a small amount of the Hessian diagonal $\lambda \text{diag}(\mathbf{A})$ to the Hessian $\mathbf{A}$ itself, as is done in the Levenberg–Marquardt non-linear least squares algorithm (Appendix A.3), usually works well.

11.4.6 *Application*: Reconstruction from internet photos

The most widely used application of structure from motion is in the reconstruction of 3D objects and scenes from video sequences and collections of images (Pollefeys and Van Gool 2002). The last two decades have seen an explosion of techniques for performing this task automatically without the need for any manual correspondence or pre-surveyed ground control points. A lot of these techniques assume that the scene is taken with the same camera and hence the images all have the same intrinsics (Fitzgibbon and Zisserman 1998; Koch, Pollefeys, and Van Gool 2000; Schaffalitzky and Zisserman 2002; Tuytelaars and Van Gool 2004; Pollefeys, Nistér *et al.* 2008; Moons, Van Gool, and Vergauwen 2010). Many of these techniques take the results of the sparse feature matching and structure from motion computation and then compute dense 3D surface models using multi-view stereo techniques (Section 12.7) (Koch, Pollefeys, and Van Gool 2000; Pollefeys and Van Gool 2002; Pollefeys, Nistér *et al.* 2008; Moons, Van Gool, and Vergauwen 2010; Schönberger, Zheng *et al.* 2016).

An exciting innovation in this space has been the application of structure from motion and multi-view stereo techniques to thousands of images taken from the internet, where very little is known about the cameras taking the photographs (Snavely, Seitz, and Szeliski 2008a). Before the structure from motion computation can begin, it is first necessary to establish sparse correspondences between different pairs of images and to then link such correspondences into *feature tracks*, which associate individual 2D image features with global 3D points. Because the $O(N^2)$ comparison of all pairs of images can be very slow, a number of techniques have been developed in the recognition community to make this process faster (Section 7.1.4) (Nistér and Stewénius 2006; Philbin, Chum *et al.* 2008; Li, Wu *et al.* 2008; Chum, Philbin, and Zisserman 2008; Chum and Matas 2010a; Arandjelović and Zisserman 2012).

To begin the reconstruction process, it is important to select a good pair of images, where there are both a large number of consistent matches (to lower the likelihood of incorrect correspondences)



(a) (b) (c)

Figure 11.18 3D reconstructions produced by the incremental structure from motion algorithm developed by Snavely, Seitz, and Szeliski (2006) © 2006 ACM: (a) cameras and point cloud from Trafalgar Square; (b) cameras and points overlaid on an image from the Great Wall of China; (c) overhead view of a reconstruction of the Old Town Square in Prague registered to an aerial photograph.

and a significant amount of out-of-plane parallax,[28] to ensure that a stable reconstruction can be obtained (Snavely, Seitz, and Szeliski 2006). The EXIF tags associated with the photographs can be used to get good initial estimates for camera focal lengths, although this is not always strictly necessary, because these parameters are re-adjusted as part of the bundle adjustment process.

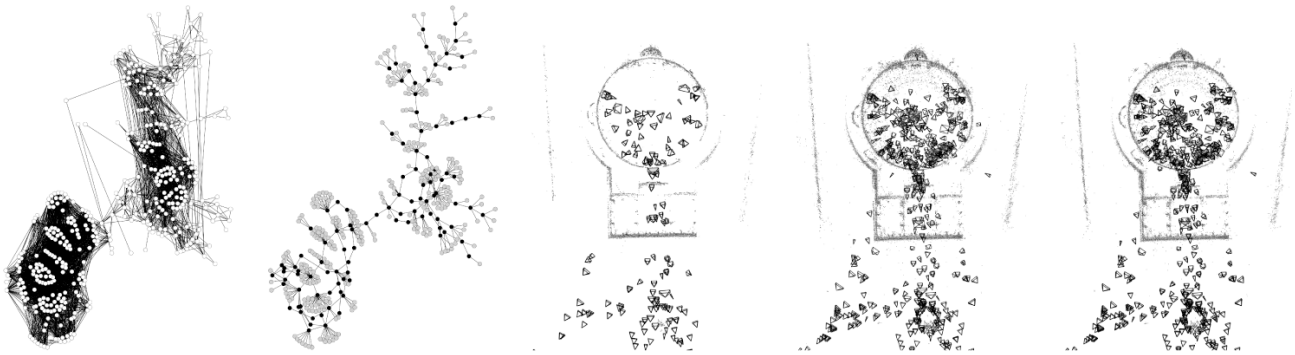
Once an initial pair has been reconstructed, the pose of cameras that see a sufficient number of the resulting 3D points can be estimated (Section 11.2) and the complete set of cameras and feature correspondences can be used to perform another round of bundle adjustment. Figure 11.17 shows the progression of the incremental bundle adjustment algorithm, where sets of cameras are added after each successive round of bundle adjustment, while Figure 11.18 shows some additional results. An alternative to this kind of *seed and grow* approach is to first reconstruct triplets of images and then hierarchically merge them into larger collections (Fitzgibbon and Zisserman 1998).

Unfortunately, as the incremental structure from motion algorithm continues to add more cameras and points, it can become extremely slow. The direct solution of a dense system of $O(N)$ equations for the camera pose updates can take $O(N^3)$ time; while structure from motion problems are rarely dense, scenes such as city squares have a high percentage of cameras that see points in common. Re-running the bundle adjustment algorithm after every few camera additions results in a quartic scaling of the run time with the number of images in the dataset. One approach to solving this problem is to select a smaller number of images for the original scene reconstruction and to fold in the remaining images at the very end.

Snavely, Seitz, and Szeliski (2008b) develop an algorithm for computing such a *skeletal set* of images, which is guaranteed to produce a reconstruction whose error is within a bounded factor of the optimal reconstruction accuracy. Their algorithm first evaluates all pairwise uncertainties (position covariances) between overlapping images and then chains them together to estimate a lower bound for the relative uncertainty of any distant pair. The skeletal set is constructed so that the maximal uncertainty between any pair grows by no more than a constant factor. Figure 11.19 shows an example of the skeletal set computed for 784 images of the Pantheon in Rome. As you can see, even though the skeletal set contains just a fraction of the original images, the shapes of the skeletal set and full bundle adjusted reconstructions are virtually indistinguishable.

Since the initial publication on large-scale internet photo reconstruction by Snavely, Seitz, and

[28] A simple way to compute this is to robustly fit a homography to the correspondences and measure reprojection errors.



(a) (b) (c) (d) (e)

Figure 11.19 Large-scale structure from motion using skeletal sets (Snavely, Seitz, and Szeliski 2008b) © 2008 IEEE: (a) original match graph for 784 images; (b) skeletal set containing 101 images; (c) top-down view of scene (Pantheon) reconstructed from the skeletal set; (d) reconstruction after adding in the remaining images using pose estimation; (e) final bundle adjusted reconstruction, which is almost identical.

Szeliski (2008a,b), there have been a large number of follow-on papers exploring even larger datasets and more efficient algorithms (Agarwal, Furukawa *et al.* 2010, 2011; Frahm, Fite-Georgel *et al.* 2010; Wu 2013; Heinly, Schönberger *et al.* 2015; Schönberger and Frahm 2016). Among these, the COLMAP open source structure from motion and multi-view stereo system is currently one of the most widely used, as it can reconstruct extremely large scenes, such as the one shown in Figure 11.20 (Schönberger and Frahm 2016).[29]

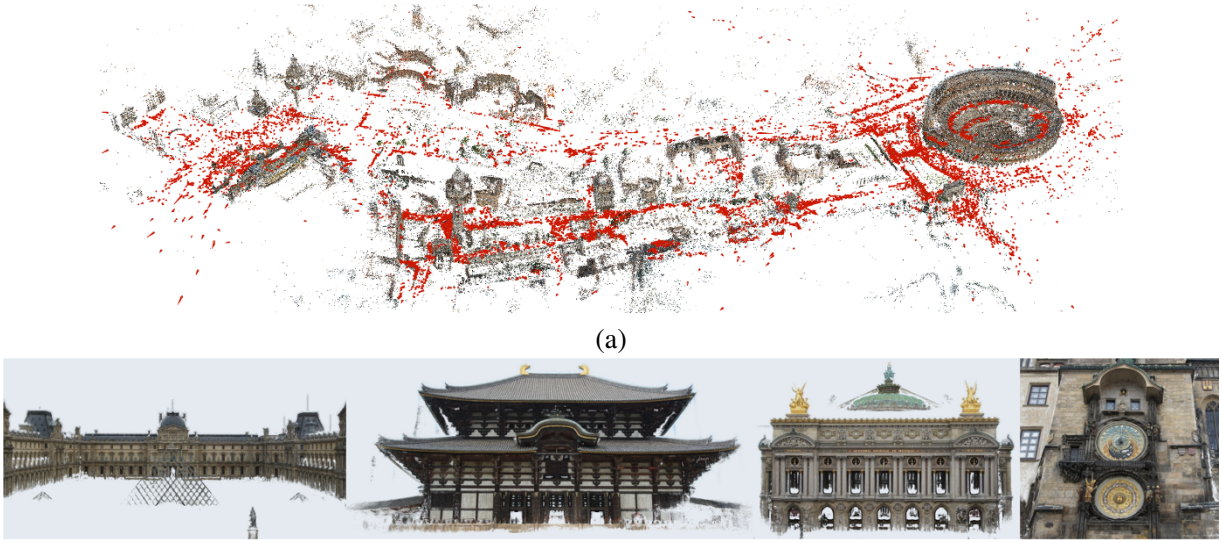
The ability to automatically reconstruct 3D models from large, unstructured image collections has also brought to light subtle problems with traditional structure from motion algorithms, including the need to deal with repetitive and duplicate structures (Wu, Frahm, and Pollefeys 2010; Roberts, Sinha *et al.* 2011; Wilson and Snavely 2013; Heinly, Dunn, and Frahm 2014) as well as dynamic visual objects such as people (Ji, Dunn, and Frahm 2014; Zheng, Wang *et al.* 2014). It has also opened up a wide variety of additional applications, including the ability to automatically find and label locations and regions of interest (Simon, Snavely, and Seitz 2007; Simon and Seitz 2008; Gammeter, Bossard *et al.* 2009) and to cluster large image collections so that they can be automatically labeled (Li, Wu *et al.* 2008; Quack, Leibe, and Van Gool 2008). Some additional applications related to image-based rendering are discussed in more detail in Section 14.1.2.

11.4.7 Global structure from motion

While incremental bundle adjustment algorithms are still the most commonly used approaches for large-scale reconstruction (Schönberger and Frahm 2016), they can be quite slow because of the need to successively solve increasing larger optimization problems. An alternative to iteratively growing the solution is to solve for all of the structure and motion unknowns in a single global step, once the feature correspondences have been established.

One approach to this is to set up a linear system of equations that relate all of the camera centers and 3D point, line, and plane equations to the known 2D feature or line positions (Kaucic, Hartley, and Dano 2001; Rother 2003). However, these approaches require a reference plane (e.g., building wall) to be visible and matched in all images, and are also sensitive to distant points, which must first be discarded. These approaches, while theoretically interesting, are not widely used.

[29] https://colmap.github.io



(a)

(b)

Figure 11.20 Large-scale reconstructions created with the COLMAP structure from motion and multi-view stereo system: (a) sparse model of central Rome constructed from 21K photos (Schönberger and Frahm 2016) © 2016 IEEE; (b) dense models of several landmarks produced with the MVS pipeline (Schönberger, Zheng *et al.* 2016) © 2016 Springer.

A second approach, first proposed by Govindu (2001), starts by computing pairwise Euclidean structure and motion reconstructions using the techniques discussed in Section 11.3.[30] Pairwise rotation estimates are then used to compute a globally consistent orientation estimate for each camera, using a process known as *rotation averaging* (Govindu 2001; Martinec and Pajdla 2007; Chatterjee and Govindu 2013; Hartley, Trumpf *et al.* 2013; Dellaert, Rosen *et al.* 2020).[31] In a final step, the camera positions are determined by scaling each of the local camera translations, after they have been rotated into a global coordinate system (Govindu 2001, 2004; Martinec and Pajdla 2007; Sinha, Steedly, and Szeliski 2010). In the robotics (SLAM) community, this last step is called *pose graph optimization* (Carlone, Tron *et al.* 2015).

Figure 11.21 shows a more recent pipeline implementing this concept, which includes the initial feature point extraction, matching, and two-view reconstruction, followed by global rotation estimation, and then a final solve for the camera centers. The pipeline developed by Sinha, Steedly, and Szeliski (2010) also matches vanishing points, when these can be found, in order to eliminate rotational drift in the global orientation estimates.

While there are several alternative algorithms for estimating the global rotations, an even wider variety of algorithms exists for estimating the camera centers. After rotating all of the cameras by their global rotation estimate, we can compute globally oriented local translation direction in each reconstructed pair ij and denote this as $\hat{\mathbf{t}}_{ij}$. The fundamental relationship between the unknown camera centers $\{\mathbf{c}_i\}$ and the translation directions can be written as

$$\mathbf{c}_j - \mathbf{c}_i = s_{ij}\hat{\mathbf{t}}_{ij} \tag{11.69}$$

[30] While almost of all of these techniques assume known calibration (focal lengths) for each image, Sweeney, Kneip *et al.* (2015) estimate focal lengths from refined fundamental matrices.

[31] We have already introduced the concept of rotation averaging when we discussed global registration of panoramas in Section 8.3.1.

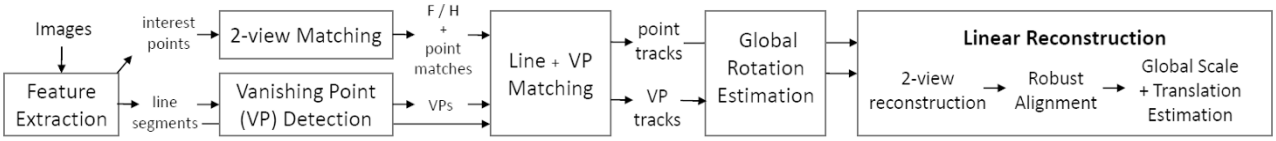


Figure 11.21 Global structure from motion pipeline from Sinha, Steedly, and Szeliski (2010) © 2010 Springer. Vanishing point and feature-based pairwise rotation estimates are used to first estimate a globally consistent set of orientations (rotations). The scales of all pairwise reconstructions along with the camera center positions are then estimated in a single linear least squares minimization.

or

$$\hat{\mathbf{t}}_{ij} \times (\mathbf{c}_j - \mathbf{c}_i) = 0 \tag{11.70}$$

(Govindu 2001). The first set of equations can be solved to obtain the camera centers $\{\mathbf{c}_i\}$ and the scale variables s_{ij}, while the second directly produces only the camera positions. In addition to being homogeneous (only known up to a scale), the camera centers also have a translational *gauge freedom*, i.e., they can all be translated (but this is always the case with structure from motion).

Because these equations minimize the algebraic alignment between local translation directions and global camera center differences, they do not correctly weight reconstructions with different baselines. Several alternatives have been proposed to remediate this (Govindu 2004; Sinha, Steedly, and Szeliski 2010; Jiang, Cui, and Tan 2013; Moulon, Monasse, and Marlet 2013; Wilson and Snavely 2014; Cui and Tan 2015; Özyeşil and Singer 2015; Holynski, Geraghty *et al.* 2020). Some of these techniques also cannot handle collinear cameras, as in the original formulation, as well as some more recent ones, we can shift cameras along a collinear segment and still satisfy the directional constraints.

For community photo collections taken over a large area such as a plaza, this is not a crucial problem (Wilson and Snavely 2014). However, for reconstructions from video or walks around or through a building, the collinear camera problem is a real issue. Sinha, Steedly, and Szeliski (2010) handle this by estimating the relative scales of pairwise reconstructions that share a common camera and then use these relative scales to constraint all of the global scales.

Two open-source structure from motion pipelines that include some of these global techniques are Theia[32] (Sweeney, Hollerer, and Turk 2015) and OpenMVG[33] (Moulon, Monasse *et al.* 2016). The papers have nice reviews of the related algorithms.

11.4.8 Constrained structure and motion

The most general algorithms for structure from motion make no prior assumptions about the objects or scenes that they are reconstructing. In many cases, however, the scene contains higher-level geometric primitives, such as lines and planes. These can provide information complementary to interest points and also serve as useful building blocks for 3D modeling and visualization. Furthermore, these primitives are often arranged in particular relationships, i.e., many lines and planes are either parallel or orthogonal to each other (Zhou, Furukawa, and Ma 2019; Zhou, Furukawa *et al.* 2020). This is particularly true of architectural scenes and models, which we study in more detail in Section 13.6.1.

Sometimes, instead of exploiting regularity in the scene structure, it is possible to take advantage of a constrained motion model. For example, if the object of interest is rotating on a turntable

[32] http://www.theia-sfm.org
[33] https://github.com/openMVG/openMVG

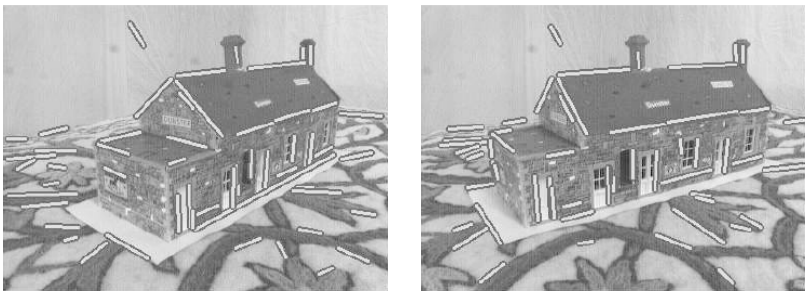


Figure 11.22 Two images of a toy house along with their matched 3D line segments (Schmid and Zisserman 1997) © 1997 Springer.

(Szeliski 1991b), i.e., around a fixed but unknown axis, specialized techniques can be used to recover this motion (Fitzgibbon, Cross, and Zisserman 1998). In other situations, the camera itself may be moving in a fixed arc around some center of rotation (Shum and He 1999). Specialized capture setups, such as mobile stereo camera rigs or moving vehicles equipped with multiple fixed cameras, can also take advantage of the knowledge that individual cameras are (mostly) fixed with respect to the capture rig, as shown in Figure 11.15.[34]

Line-based techniques

It is well known that pairwise epipolar geometry cannot be recovered from line matches alone, even if the cameras are calibrated. To see this, think of projecting the set of lines in each image into a set of 3D planes in space. You can move the two cameras around into any configuration you like and still obtain a valid reconstruction for 3D lines.

When lines are visible in three or more views, the trifocal tensor can be used to transfer lines from one pair of images to another (Hartley and Zisserman 2004). The trifocal tensor can also be computed on the basis of line matches alone.

Schmid and Zisserman (1997) describe a widely used technique for matching 2D lines based on the average of 15×15 pixel correlation scores evaluated at all pixels along their common line segment intersection.[35] In their system, the epipolar geometry is assumed to be known, e.g., computed from point matches. For wide baselines, all possible homographies corresponding to planes passing through the 3D line are used to warp pixels and the maximum correlation score is used. For triplets of images, the trifocal tensor is used to verify that the lines are in geometric correspondence before evaluating the correlations between line segments. Figure 11.22 shows the results of using their system.

Bartoli and Sturm (2003) describe a complete system for extending three view relations (trifocal tensors) computed from manual line correspondences to a full bundle adjustment of all the line and camera parameters. The key to their approach is to use the Plücker coordinates (2.12) to parameterize lines and to directly minimize reprojection errors. It is also possible to represent 3D line segments by their endpoints and to measure either the reprojection error perpendicular to the detected 2D line segments in each image or the 2D errors using an elongated uncertainty ellipse aligned with the line segment direction (Szeliski and Kang 1994).

[34] Because of mechanical compliance and jitter, it may be prudent to allow for a small amount of individual camera rotation around a nominal position.

[35] Because lines often occur at depth or orientation discontinuities, it may be preferable to compute correlation scores (or to match color histograms (Bay, Ferrari, and Van Gool 2005)) separately on each side of the line.

Instead of reconstructing 3D lines, Bay, Ferrari, and Van Gool (2005) use RANSAC to group lines into likely coplanar subsets. Four lines are chosen at random to compute a homography, which is then verified for these and other plausible line segment matches by evaluating color histogram-based correlation scores. The 2D intersection points of lines belonging to the same plane are then used as virtual measurements to estimate the epipolar geometry, which is more accurate than using the homographies directly.

An alternative to grouping lines into coplanar subsets is to group lines by parallelism. Whenever three or more 2D lines share a common vanishing point, there is a good likelihood that they are parallel in 3D. By finding multiple vanishing points in an image (Section 7.4.3) and establishing correspondences between such vanishing points in different images, the relative rotations between the various images (and often the camera intrinsics) can be directly estimated (Section 11.1.1). Finding an orthogonal set of vanishing points and using these to establish a global orientation is often called invoking the *Manhattan world assumption* (Coughlan and Yuille 1999). A generalized version where streets can meet at non-orthogonal angles was called the *Atlanta world* by Schindler and Dellaert (2004).

Shum, Han, and Szeliski (1998) describe a 3D modeling system that constructs calibrated panoramas from multiple images (Section 11.4.2) and then has the user draw vertical and horizontal lines in the image to demarcate the boundaries of planar regions. The lines are used to establish an absolute rotation for each panorama and are then used (along with the inferred vertices and planes) to build a 3D structure, which can be recovered up to scale from one or more images (Figure 13.20).

A fully automated approach to line-based structure from motion is presented by Werner and Zisserman (2002). In their system, they first find lines and group them by common vanishing points in each image (Section 7.4.3). The vanishing points are then used to calibrate the camera, i.e., to perform a "metric upgrade" (Section 11.1.1). Lines corresponding to common vanishing points are then matched using both appearance (Schmid and Zisserman 1997) and trifocal tensors. These lines are then used to infer planes and a block-structured model for the scene, as described in more detail in Section 13.6.1. More recent work using deep neural networks can also be used to construct 3D wireframe models from one or more images.

Plane-based techniques

In scenes that are rich in planar structures, e.g., in architecture, it is possible to directly estimate homographies between different planes, using either feature-based or intensity-based methods. In principle, this information can be used to simultaneously infer the camera poses and the plane equations, i.e., to compute plane-based structure from motion.

Luong and Faugeras (1996) show how a fundamental matrix can be directly computed from two or more homographies using algebraic manipulations and least squares. Unfortunately, this approach often performs poorly, because the algebraic errors do not correspond to meaningful reprojection errors (Szeliski and Torr 1998).

A better approach is to *hallucinate* virtual point correspondences within the areas from which each homography was computed and to feed them into a standard structure from motion algorithm (Szeliski and Torr 1998). An even better approach is to use full bundle adjustment with explicit plane equations, as well as additional constraints to force reconstructed co-planar features to lie exactly on their corresponding planes. (A principled way to do this is to establish a coordinate frame for each plane, e.g., at one of the feature points, and to use 2D in-plane parameterizations for the other points.) The system developed by Shum, Han, and Szeliski (1998) shows an example of such an approach, where the directions of lines and normals for planes in the scene are prespecified by the user. In more recent work, Micusik and Wildenauer (2017) use planes as additional constraints

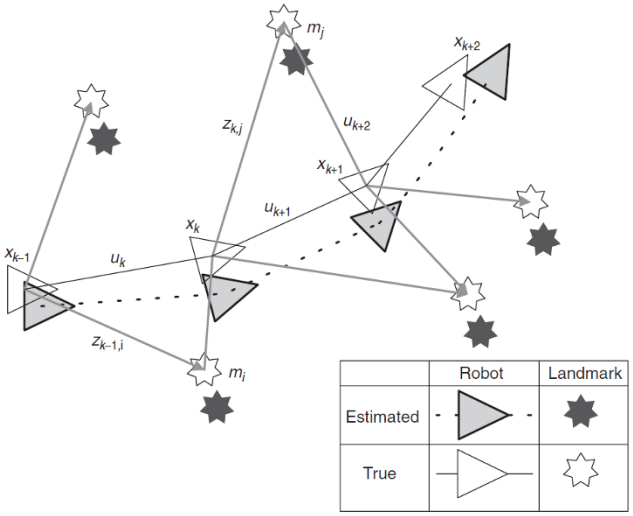


Figure 11.23 In simultaneous localization and mapping (SLAM), the system simultaneously estimates the positions of a robot and its nearby landmarks (Durrant-Whyte and Bailey 2006) © 2006 IEEE.

inside a bundle adjustment formulation. Other recent papers that use combinations of lines and/or planes to reduce drift in 3D reconstructions include (Zhou, Zou *et al.* 2015), Li, Yao *et al.* (2018), Yang and Scherer (2019), and Holynski, Geraghty *et al.* (2020).

11.5 Simultaneous localization and mapping (SLAM)

While the computer vision community has been studying structure from motion, i.e., the reconstruction of sparse 3D models from multiple images and videos, since the early 1980s (Longuet-Higgins 1981), the mobile robotics community has in parallel been studying the automatic construction of 3D maps from moving robots.[36] In robotics, the problem was formulated as the simultaneous estimation of 3D robot and *landmark* poses (Figure 11.23), and was known as *probabilistic mapping* (Thrun, Burgard, and Fox 2005) and *simultaneous localization and mapping* (SLAM) (Durrant-Whyte and Bailey 2006; Bailey and Durrant-Whyte 2006; Cadena, Carlone *et al.* 2016). In the computer vision community, the problem was originally called *visual odometry* (Levin and Szeliski 2004; Nistér, Naroditsky, and Bergen 2006; Maimone, Cheng, and Matthies 2007), although that term is now usually reserved for shorter-range motion estimation that does not involve building a global map with *loop closing* (Cadena, Carlone *et al.* 2016).

Early versions of such algorithms used range-sensing techniques, such as ultrasound, laser range finders, or stereo matching, to estimate local 3D geometry, which could then be fused into a 3D model. Newer techniques can perform the same task based purely on visual feature tracking from a monocular camera (Davison, Reid *et al.* 2007). Good introductory tutorials can be found in Durrant-Whyte and Bailey (2006) and Bailey and Durrant-Whyte (2006), while more comprehensive surveys of more recent techniques are presented in (Fuentes-Pacheco, Ruiz-Ascencio, and Rendón-Mancha 2015) and Cadena, Carlone *et al.* (2016).

SLAM differs from bundle adjustment in two fundamental aspects. First, it allows for a variety

[36] In the 1980s, the vision and robotics communities were essentially the same set of researchers working in these two sub-fields of artificial intelligence.

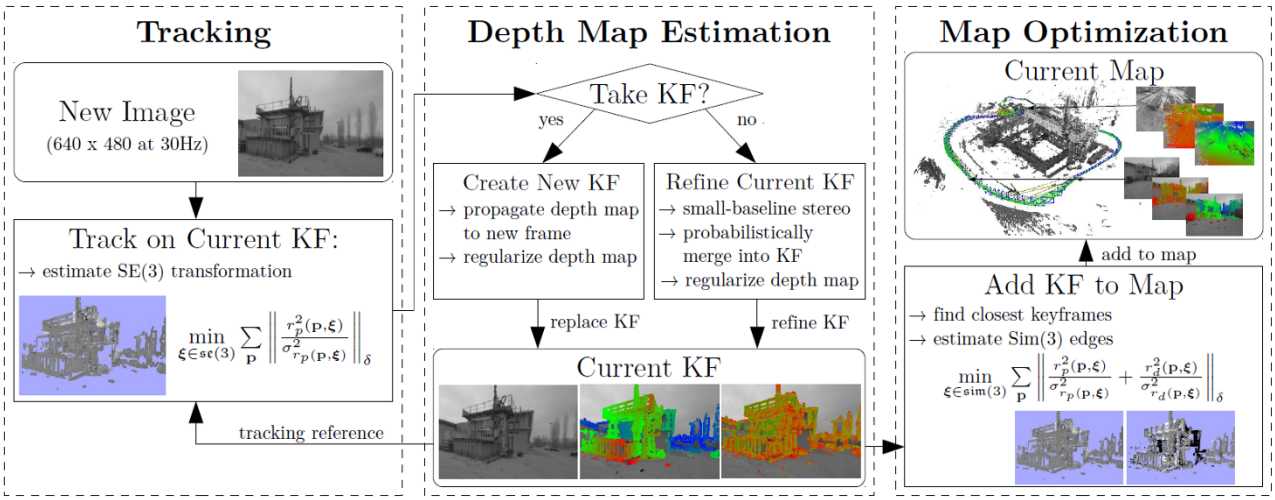


Figure 11.24 The architecture of the LSD-SLAM system (Engel, Schöps, and Cremers 2014) © 2014 Springer, showing the front end, which does the tracking, data association, and local 3D pose and structure (depth map) updating, and the back end, which does global map optimization.

of sensing devices, instead of just being restricted to tracked or matched feature points. Second, it solves the localization problem *online*, i.e., with no or very little lag in providing the current sensor pose. This makes it the method of choice for both time-critical robotics applications such as autonomous navigation (Section 11.5.1) and real-time augmented reality (Section 11.5.2).

Some of the important milestones in SLAM include:

- the application of SLAM to monocular cameras (MonoSLAM) (Davison, Reid *et al.* 2007);
- parallel tracking and mapping (PTAM) (Klein and Murray 2007), which split the front end (tracking) and back end (mapping) processes (Figure 11.24) onto two separate threads running at different rates (Figure 11.27) and then implemented the whole process on a camera phone (Klein and Murray 2009);
- adaptive relative bundle adjustment (Sibley, Mei *et al.* 2009, 2010), which maintains collections of local reconstructions anchored at different keyframes;
- incremental smoothing and mapping (iSAM) (Kaess, Ranganathan, and Dellaert 2008; Kaess, Johannsson *et al.* 2012) and other applications of factor graphs to handle the speed-accuracy-delay tradeoff (Dellaert and Kaess 2017; Dellaert 2021);
- dense tracking and mapping (DTAM) (Newcombe, Lovegrove, and Davison 2011), which estimates and updates a dense depth map for every frame;
- ORB-SLAM (Mur-Artal, Montiel, and Tardos 2015) and ORB-SLAM2 (Mur-Artal and Tardós 2017), which handle monocular, stereo, and RGB-D cameras as well as loop closures;
- SVO (semi-direct visual odometry) (Forster, Zhang *et al.* 2017), which combines patch-based tracking with classic bundle adjustment; and
- LSD-SLAM (large-scale direct SLAM) (Engel, Schöps, and Cremers 2014) and DSO (direct sparse odometry) (Engel, Koltun, and Cremers 2018), which only keep depth estimates at strong gradient locations (Figure 11.24).

- BAD SLAM (bundle adjusted direct RGB-D SLAM) (Schöps, Sattler, and Pollefeys 2019a).

Many of these systems have open source implementations. Some widely used benchmarks include a benchmark for RGB-D SLAM systems (Sturm, Engelhard *et al.* 2012), the KITTI Visual Odometry / SLAM benchmark (Geiger, Lenz *et al.* 2013), the synthetic ICL-NUIM dataset (Handa, Whelan *et al.* 2014), the TUM monoVO dataset (Engel, Usenko, and Cremers 2016), the EuRoC MAV dataset (Burri, Nikolic *et al.* 2016), the ETH3D SLAM benchmark (Schöps, Sattler, and Pollefeys 2019a), and the GSLAM general SLAM benchmark (Zhao, Xu *et al.* 2019).

The most recent trend in SLAM has been the integration with visual-inertial odometry (VIO) algorithms (Mourikis and Roumeliotis 2007; Li and Mourikis 2013; Forster, Carlone *et al.* 2016), which combine higher-frequency inertial measurement unit (IMU) measurements with visual tracks, which serve to remove low-frequency drift. Because IMUs are now commonplace in consumer devices such as cell phones and action cameras, VIO-enhanced SLAM systems serve as the foundation for widely used mobile augmented reality frameworks such as ARKit and ARCore (Section 11.5.2). A dataset and evaluation of open-source VIO systems can be found at Schubert, Goll *et al.* (2018).

As you can tell from this very brief overview, SLAM is an incredibly rich and rapidly evolving field of research, full of challenging robust optimization and real-time performance problems. A good source for finding a list of the most recent papers and algorithms is the KITTI Visual Odometry/SLAM Evaluation[37] (Geiger, Lenz, and Urtasun 2012) and the recent survey paper on computer vision for autonomous driving (Janai, Güney *et al.* 2020, Section 13.2).

11.5.1 *Application*: Autonomous navigation

Since the early days of artificial intelligence and robotics, computer vision has been used to enable manipulation for dextrous robots and navigation for autonomous robots (Janai, Güney *et al.* 2020; Kubota 2019). Some of the earliest vision-based navigation systems include the Stanford Cart (Figure 11.25a) and CMU Rover (Moravec 1980, 1983), the Terregator (Wallace, Stentz *et al.* 1985), and the CMU Nablab (Thorpe, Hebert *et al.* 1988), which originally could only advance 4m every 10 sec ($<$ 1 mph), and which was also the first system to use a neural network for driving (Pomerleau 1989).

The early algorithms and technologies advanced rapidly, with the VaMoRs system of Dickmanns and Mysliwetz (1992) operating a 25Hz Kalman filter loop and driving with good lane markings at 100 km/h. By the mid 2000s, when DARPA introduced their Grand Challenge and Urban Challenge, vehicles equipped with both range-finding lidar cameras and stereo cameras were able to traverse rough outdoor terrain and navigate city streets at regular human driving speeds (Urmson, Anhalt *et al.* 2008; Montemerlo, Becker *et al.* 2008).[38] These systems led to the formation of industrial research projects at companies such as Google and Tesla,[39] as well numerous startups, many of which exhibit their vehicles at computer vision conferences (Figure 11.25c–d).

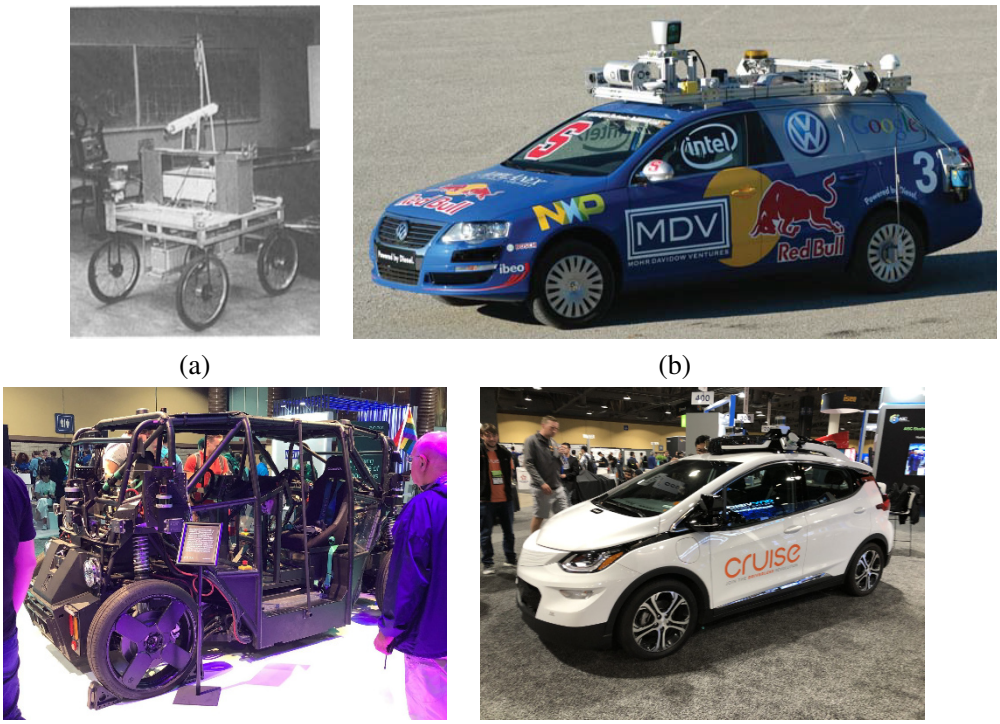
A comprehensive review of computer vision technologies for autonomous vehicles can be found in the survey by Janai, Güney *et al.* (2020), which also comes with a useful on-line visualization tool of relevant papers.[40] The survey contains chapters on the large number of vision algorithms and components that go into autonomous navigation, which include datasets and benchmarks, sensors, object detection and tracking, segmentation, stereo, flow and scene flow, SLAM, scene understanding, and end-to-end learning of autonomous driving behaviors.

[37] http://www.cvlibs.net/datasets/kitti/eval_odometry.php

[38] Algorithms that use range data as part of their map building and localization are commonly called *RGB-D SLAM* systems (Sturm, Engelhard *et al.* 2012).

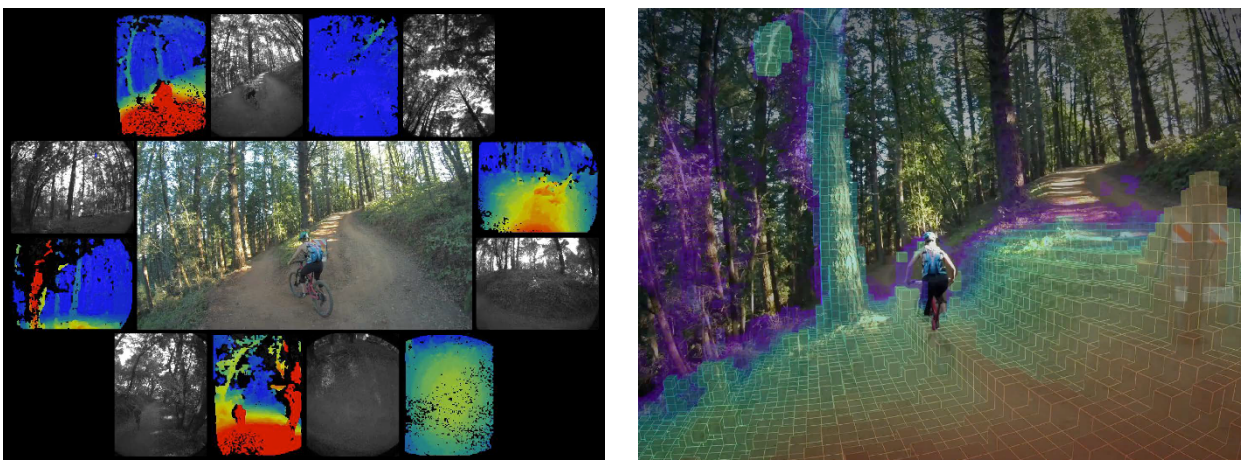
[39] You can find a number of talks about Tesla's efforts on Andrej Karpathy's web page, https://karpathy.ai.

[40] http://www.cvlibs.net/projects/autonomous_vision_survey



(a) (b) (c) (d)

Figure 11.25 Autonomous vehicles: (a) the Stanford Cart (Moravec 1983) ©1983 IEEE; (b) Junior: The Stanford entry in the Urban Challenge (Montemerlo, Becker *et al.* 2008) © 2008 Wiley; (c–d) self-driving car prototypes from the CVPR 2019 exhibit floor.



(a) (b)

Figure 11.26 Fully autonomous Skydio R1 drone flying in the wild © 2019 Skydio: (a) multiple input images and depth maps; (b) fully integrated 3D map (Cross 2019).

(a) (b)

Figure 11.27 3D augmented reality: (a) Darth Vader and a horde of Ewoks battle it out on a table-top recovered using real-time, keyframe-based structure from motion (Klein and Murray 2007) © 2007 IEEE; (b) a virtual teapot is fixed to the top of a real-world coffee cup, whose pose is re-recognized at each time frame (Gordon and Lowe 2006) © 2007 Springer.

In addition to autonomous navigation for wheeled (and legged) robots and vehicles, computer vision algorithms are widely used in the control of autonomous drones for both recreational applications (Ackerman 2019) (Figure 11.26) and drone racing (Jung, Hwang *et al.* 2018; Kaufmann, Gehrig *et al.* 2019). A great talk describing Skydio's approach to visual autonomous navigation by Gareth Cross (2019) can be found in the ICRA 2019 Workshop on Algorithms and Architectures for Learning In-The-Loop Systems in Autonomous Flight[41] as well as Lecture 23 in Pieter Abbeel's (2019) class on Advanced Robotics, which has dozens of other interesting related lectures.

11.5.2 *Application*: Smartphone augmented reality

Another closely related application is *augmented reality*, where 3D objects are inserted into a video feed in real time, often to annotate or help users understand a scene (Azuma, Baillot *et al.* 2001; Feiner 2002; Billinghurst, Clark, and Lee 2015). While traditional systems require prior knowledge about the scene or object being visually tracked (Rosten and Drummond 2005), newer systems can simultaneously build up a model of the 3D environment and then track it so that graphics can be superimposed (Reitmayr and Drummond 2006; Wagner, Reitmayr *et al.* 2008).

Klein and Murray (2007) describe a *parallel tracking and mapping* (PTAM) system, which simultaneously applies full bundle adjustment to keyframes selected from a video stream, while performing robust real-time pose estimation on intermediate frames (Figure 11.27a). Once an initial 3D scene has been reconstructed, a dominant plane is estimated (in this case, the table-top) and 3D animated characters are virtually inserted. Klein and Murray (2008) extend this system to handle even faster camera motion by adding edge features, which can still be tracked even when interest points become too blurred. They also use a direct (intensity-based) rotation estimation algorithm for even faster motions.

Instead of modeling the whole scene as one rigid reference frame, Gordon and Lowe (2006) first build a 3D model of an individual object using feature matching and structure from motion. Once the system has been initialized, for every new frame they find the object and its pose using a 3D

[41] https://uav-learning-icra.github.io/2019

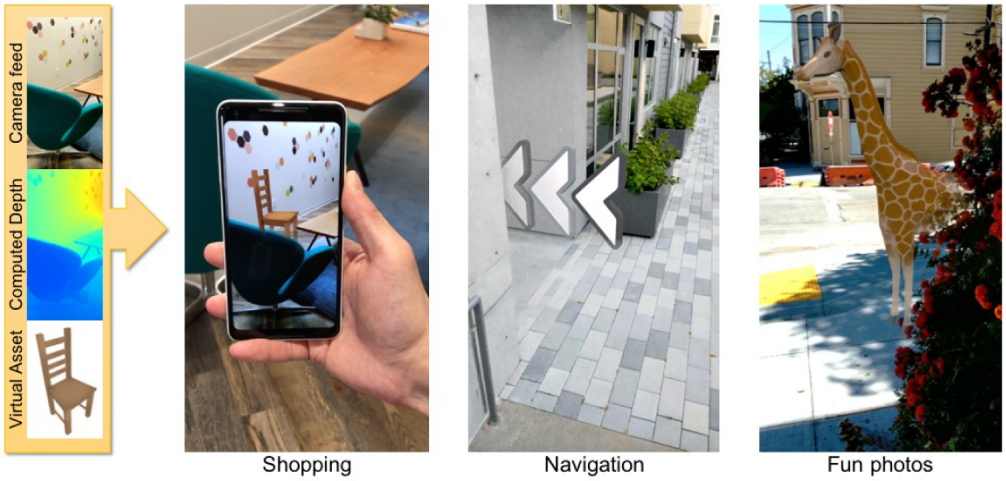


Figure 11.28 Smartphone augmented reality showing real-time depth occlusion effects (Valentin, Kowdle *et al.* 2018) © 2018 ACM.

instance recognition algorithm, and then superimpose a graphical object onto that model, as shown in Figure 11.27b.

While reliably tracking such objects and environments is now a well-solved problem, with frameworks such as ARKit,[42] ARCore,[43] and Spark AR[44] being widely used for mobile AR application development, determining which pixels should be occluded by foreground scene elements (Chuang, Agarwala *et al.* 2002; Wang and Cohen 2009) still remains an active research area.

One recent example of such work is the Smartphone AR system developed by Valentin, Kowdle *et al.* (2018) shown in Figure 11.28. The system proceeds by generating a semi-dense depth map by matching the current frame to a previous keyframe using a CRF followed by a filtering step. This map is then interpolated to full resolution using a novel planar bilateral solver, and the resulting depth map used for occlusion effects. As accurate per-pixel depth is such an essential component of augmented reality effects, we are likely to see rapid progress in this area, using both active and passive depth sensing technologies.

11.6 Additional reading

Camera calibration was first studied in photogrammetry (Brown 1971; Slama 1980; Atkinson 1996; Kraus 1997) but it has also been widely studied in computer vision (Tsai 1987; Gremban, Thorpe, and Kanade 1988; Champleboux, Lavallée *et al.* 1992b; Zhang 2000; Grossberg and Nayar 2001). Vanishing points observed either from rectahedral calibration objects or architecture are often used to perform rudimentary calibration (Caprile and Torre 1990; Becker and Bove 1995; Liebowitz and Zisserman 1998; Cipolla, Drummond, and Robertson 1999; Antone and Teller 2002; Criminisi, Reid, and Zisserman 2000; Hartley and Zisserman 2004; Pflugfelder 2008). Performing camera calibration without using known targets is known as *self-calibration* and is discussed in textbooks and surveys on structure from motion (Faugeras, Luong, and Maybank 1992; Hartley and Zisserman

[42] https://developer.apple.com/augmented-reality
[43] https://developers.google.com/ar
[44] https://sparkar.facebook.com/ar-studio

2004; Moons, Van Gool, and Vergauwen 2010). One popular subset of such techniques uses pure rotational motion (Stein 1995; Hartley 1997b; Hartley, Hayman *et al.* 2000; de Agapito, Hayman, and Reid 2001; Kang and Weiss 1999; Shum and Szeliski 2000; Frahm and Koch 2003).

The topic of registering 3D point datasets is called *absolute orientation* (Horn 1987) and *3D pose estimation* (Lorusso, Eggert, and Fisher 1995). A variety of techniques has been developed for simultaneously computing 3D point correspondences and their corresponding rigid transformations (Besl and McKay 1992; Zhang 1994; Szeliski and Lavallée 1996; Gold, Rangarajan *et al.* 1998; David, DeMenthon *et al.* 2004; Li and Hartley 2007; Enqvist, Josephson, and Kahl 2009). When only 2D observations are available, a variety of algorithms for the *linear PnP* (*perspective n-point*) have been developed (DeMenthon and Davis 1995; Quan and Lan 1999; Moreno-Noguer, Lepetit, and Fua 2007; Terzakis and Lourakis 2020). More recent approaches to pose estimation use deep networks (Arandjelovic, Gronat *et al.* 2016; Brachmann, Krull *et al.* 2017; Xiang, Schmidt *et al.* 2018; Oberweger, Rad, and Lepetit 2018; Hu, Hugonot *et al.* 2019; Peng, Liu *et al.* 2019). Estimating pose from RGB-D images is also very active (Drost, Ulrich *et al.* 2010; Brachmann, Michel *et al.* 2016; Labbé, Carpentier *et al.* 2020). In addition to recognizing object pose for robotics tasks, pose estimation is widely used in location recognition (Sattler, Zhou *et al.* 2019; Revaud, Weinzaepfel *et al.* 2019; Zhou, Sattler *et al.* 2019; Sarlin, DeTone *et al.* 2020; Luo, Zhou *et al.* 2020).

The topic of structure from motion is extensively covered in books and review articles on multi-view geometry (Faugeras and Luong 2001; Hartley and Zisserman 2004; Moons, Van Gool, and Vergauwen 2010) with survey of more recent developments in Özyeşil, Voroninski *et al.* (2017). For two-frame reconstruction, Hartley (1997a) wrote a highly cited paper on the "eight-point algorithm" for computing an essential or fundamental matrix with reasonable point normalization. When the cameras are calibrated, the five-point algorithm of Nistér (2004) can be used in conjunction with RANSAC to obtain initial reconstructions from the minimum number of points. When the cameras are uncalibrated, various self-calibration techniques can be found in work by Hartley and Zisserman (2004) and Moons, Van Gool, and Vergauwen (2010).

Triggs, McLauchlan *et al.* (1999) provide a good tutorial and survey on bundle adjustment, while Lourakis and Argyros (2009) and Engels, Stewénius, and Nistér (2006) provide tips on implementation and effective practices. Bundle adjustment is also covered in textbooks and surveys on multi-view geometry (Faugeras and Luong 2001; Hartley and Zisserman 2004; Moons, Van Gool, and Vergauwen 2010). Techniques for handling larger problems are described by Snavely, Seitz, and Szeliski (2008b), Agarwal, Snavely *et al.* (2009), Agarwal, Snavely *et al.* (2010), Jeong, Nistér *et al.* (2012), Wu (2013), Heinly, Schönberger *et al.* (2015), Schönberger and Frahm (2016), and Dellaert and Kaess (2017). While bundle adjustment is often called as an inner loop inside incremental reconstruction algorithms (Snavely, Seitz, and Szeliski 2006), hierarchical (Fitzgibbon and Zisserman 1998; Farenzena, Fusiello, and Gherardi 2009) and global (Rother and Carlsson 2002; Martinec and Pajdla 2007; Sinha, Steedly, and Szeliski 2010; Jiang, Cui, and Tan 2013; Moulon, Monasse, and Marlet 2013; Wilson and Snavely 2014; Cui and Tan 2015; Özyeşil and Singer 2015; Holynski, Geraghty *et al.* 2020) approaches for initialization are also possible and perhaps even preferable.

In the robotics community, techniques for reconstructing a 3D environment from a moving robot are called *simultaneous localization and mapping* (SLAM) (Thrun, Burgard, and Fox 2005; Durrant-Whyte and Bailey 2006; Bailey and Durrant-Whyte 2006; Fuentes-Pacheco, Ruiz-Ascencio, and Rendón-Mancha 2015; Cadena, Carlone *et al.* 2016). SLAM differs from bundle adjustment in that it allows for a variety of sensing devices and that it solves the localization problem *online*. This makes it the method of choice for both time-critical robotics applications such as autonomous navigation (Janai, Güney *et al.* 2020) and real-time augmented reality (Valentin, Kowdle *et al.* 2018). Important papers in this field include (Davison, Reid *et al.* 2007; Klein and Murray 2007, 2009; Newcombe,

Lovegrove, and Davison 2011; Kaess, Johannsson *et al.* 2012; Engel, Schöps, and Cremers 2014; Mur-Artal and Tardós 2017; Forster, Zhang *et al.* 2017; Dellaert and Kaess 2017; Engel, Koltun, and Cremers 2018; Schöps, Sattler, and Pollefeys 2019a) as well as papers that integrate SLAM with IMUs to obtain *visual inertial odometry* (VIO) (Mourikis and Roumeliotis 2007; Li and Mourikis 2013; Forster, Carlone *et al.* 2016; Schubert, Goll *et al.* 2018).

11.7 Exercises

Ex 11.1: Rotation-based calibration. Take an outdoor or indoor sequence from a rotating camera with very little parallax and use it to calibrate the focal length of your camera using the techniques described in Section 11.1.3 or Sections 8.2.3–8.3.1.

1. Take out any radial distortion in the images using one of the techniques from Exercises 11.5–11.6 or using parameters supplied for a given camera by your instructor.
2. Detect and match feature points across neighboring frames and chain them into feature tracks.
3. Compute homographies between overlapping frames and use Equations (11.8–11.9) to get an estimate of the focal length.
4. Compute a full 360° panorama and update your focal length estimate to close the gap (Section 8.2.4).
5. (Optional) Perform a complete bundle adjustment in the rotation matrices and focal length to obtain the highest quality estimate (Section 8.3.1).

Ex 11.2: Target-based calibration. Use a three-dimensional target to calibrate your camera.

1. Construct a three-dimensional calibration pattern with known 3D locations. It is not easy to get high accuracy unless you use a machine shop, but you can get close using heavy plywood and printed patterns.
2. Find the corners, e.g, using a line finder and intersecting the lines.
3. Implement one of the iterative calibration and pose estimation algorithms described in Tsai (1987), Bogart (1991), or Gleicher and Witkin (1992) or the system described in Section 11.2.2.
4. Take many pictures at different distances and orientations relative to the calibration target and report on both your re-projection errors and accuracy. (To do the latter, you may need to use simulated data.)

Ex 11.3: Calibration accuracy. Compare the three calibration techniques (plane-based, rotation-based, and 3D-target-based).

One approach is to have a different student implement each one and to compare the results. Another approach is to use synthetic data, potentially re-using the software you developed for Exercise 2.3. The advantage of using synthetic data is that you know the ground truth for the calibration and pose parameters, you can easily run lots of experiments, and you can synthetically vary the noise in your measurements.

Here are some possible guidelines for constructing your test sets:

1. Assume a medium-wide focal length (say, 50° field of view).

2. For the plane-based technique, generate a 2D grid target and project it at different inclinations.

3. For a 3D target, create an inner cube corner and position it so that it fills most of field of view.

4. For the rotation technique, scatter points uniformly on a sphere until you get a similar number of points as for other techniques.

Before comparing your techniques, predict which one will be the most accurate (normalize your results by the square root of the number of points used).

Add varying amounts of noise to your measurements and describe the noise sensitivity of your various techniques.

Ex 11.4: Single view metrology. Implement a system to measure dimensions and reconstruct a 3D model from a single image of an architectural scene using visible vanishing directions (Section 11.1.2) (Criminisi, Reid, and Zisserman 2000).

1. Find the three orthogonal vanishing points from parallel lines and use them to establish the three coordinate axes (rotation matrix $\mathbf{R}$ of the camera relative to the scene). If two of the vanishing points are finite (not at infinity), use them to compute the focal length, assuming a known image center. Otherwise, find some other way to calibrate your camera; you could use some of the techniques described by Schaffalitzky and Zisserman (2000).

2. Click on a ground plane point to establish your origin and click on a point a known distance away to establish the scene scale. This lets you compute the translation $\mathbf{t}$ between the camera and the scene. As an alternative, click on a pair of points, one on the ground plane and one above it, and use the known height to establish the scene scale.

3. Write a user interface that lets you click on ground plane points to recover their 3D locations. (Hint: you already know the camera matrix, so knowledge of a point's z value is sufficient to recover its 3D location.) Click on pairs of points (one on the ground plane, one above it) to measure vertical heights.

4. Extend your system to let you draw quadrilaterals in the scene that correspond to axis-aligned rectangles in the world, using some of the techniques described by Sinha, Steedly *et al.* (2008). Export your 3D rectangles to a VRML or PLY[45] file.

5. (Optional) Warp the pixels enclosed by the quadrilateral using the correct homography to produce a texture map for each planar polygon.

Ex 11.5: Radial distortion with plumb lines. Implement a plumb-line algorithm to determine the radial distortion parameters.

1. Take some images of scenes with lots of straight lines, e.g., hallways in your home or office, and try to get some of the lines as close to the edges of the image as possible.

2. Extract the edges and link them into curves, as described in Section 7.2.2 and Exercise 7.8.

3. Fit quadratic or elliptic curves to the linked edges using a generalization of the successive line approximation algorithm described in Section 7.4.1 and Exercise 7.11 and keep the curves that fit this form well.

[45] https://meshlab.net.

4. For each curved segment, fit a straight line and minimize the perpendicular distance between the curve and the line while adjusting the radial distortion parameters.

5. Alternate between re-fitting the straight line and adjusting the radial distortion parameters until convergence.

Ex 11.6: Radial distortion with a calibration target. Use a grid calibration target to determine the radial distortion parameters.

1. Print out a planar calibration target, mount it on a stiff board, and get it to fill your field of view.

2. Detect the squares, lines, or dots in your calibration target.

3. Estimate the homography mapping the target to the camera from the central portion of the image that does not have any radial distortion.

4. Predict the positions of the remaining targets and use the differences between the observed and predicted positions to estimate the radial distortion.

5. (Optional) Fit a general spline model (for severe distortion) instead of the quartic distortion model.

6. (Optional) Extend your technique to calibrate a fisheye lens.

Ex 11.7: Chromatic aberration. Use the radial distortion estimates for each color channel computed in the previous exercise to clean up wide-angle lens images by warping all of the channels into alignment. (Optional) Straighten out the images at the same time.

Can you think of any reasons why this warping strategy may not always work?

Ex 11.8: Triangulation. Use the calibration pattern you built and tested in Exercise 11.2 to test your triangulation accuracy. As an alternative, generate synthetic 3D points and cameras and add noise to the 2D point measurements.

1. Assume that you know the camera pose, i.e., the camera matrices. Use the 3D distance to rays (11.24) or linearized versions of Equations (11.25–11.26) to compute an initial set of 3D locations. Compare these to your known ground truth locations.

2. Use iterative non-linear minimization to improve your initial estimates and report on the improvement in accuracy.

3. (Optional) Use the technique described by Hartley and Sturm (1997) to perform two-frame triangulation.

4. See if any of the failure modes reported by Hartley and Sturm (1997) or Hartley (1998) occur in practice.

Ex 11.9: Essential and fundamental matrix. Implement the two-frame $\mathbf{E}$ and $\mathbf{F}$ matrix estimation techniques presented in Section 11.3, with suitable re-scaling for better noise immunity.

1. Use the data from Exercise 11.8 to validate your algorithms and to report on their accuracy.

2. (Optional) Implement one of the improved **F** or **E** estimation algorithms, e.g., using renormalization (Zhang 1998b; Torr and Fitzgibbon 2004; Hartley and Zisserman 2004), RANSAC (Torr and Murray 1997), least median of squares (LMS), or the five-point algorithm developed by Nistér (2004).

Ex 11.10: View morphing and interpolation. Implement automatic view morphing, i.e., compute two-frame structure from motion and then use these results to generate a smooth animation from one image to the next (Section 11.3.5).

1. Decide how to represent your 3D scene, e.g., compute a Delaunay triangulation of the matched point and decide what to do with the triangles near the border. (Hint: try fitting a plane to the scene, e.g., behind most of the points.)

2. Compute your in-between camera positions and orientations.

3. Warp each triangle to its new location, preferably using the correct perspective projection (Szeliski and Shum 1997).

4. (Optional) If you have a denser 3D model (e.g., from stereo), decide what to do at the "cracks".

5. (Optional) For a non-rigid scene, e.g., two pictures of a face with different expressions, not all of your matched points will obey the epipolar geometry. Decide how to handle them to achieve the best effect.

Ex 11.11: Bundle adjuster. Implement a full bundle adjuster. This may sound daunting, but it really is not.

1. Devise the internal data structures and external file representations to hold your camera parameters (position, orientation, and focal length), 3D point locations (Euclidean or homogeneous), and 2D point tracks (frame and point identifier as well as 2D locations).

2. Use some other technique, such as factorization, to initialize the 3D point and camera locations from your 2D tracks (e.g., a subset of points that appears in all frames).

3. Implement the code corresponding to the forward transformations in Figure 11.14, i.e., for each 2D point measurement, take the corresponding 3D point, map it through the camera transformations (including perspective projection and focal length scaling), and compare it to the 2D point measurement to get a residual error.

4. Take the residual error and compute its derivatives with respect to all the unknown motion and structure parameters, using backward chaining, as shown, e.g., in Figure 11.14 and Equation (11.19). This gives you the sparse Jacobian **J** used in Equations (8.13–8.17) and Equation (11.15).

5. Use a sparse least squares or linear system solver, e.g., MATLAB, SparseSuite, or SPARSKIT (see Appendix A.4 and A.5), to solve the corresponding linearized system, adding a small amount of diagonal preconditioning, as in Levenberg–Marquardt.

6. Update your parameters, make sure your rotation matrices are still orthonormal (e.g., by recomputing them from your quaternions), and continue iterating while monitoring your residual error.

7. (Optional) Use the "Schur complement trick" (11.68) to reduce the size of the system being solved (Triggs, McLauchlan *et al.* 1999; Hartley and Zisserman 2004; Lourakis and Argyros 2009; Engels, Stewénius, and Nistér 2006).

8. (Optional) Implement your own iterative sparse solver, e.g., conjugate gradient, and compare its performance to a direct method.

9. (Optional) Make your bundle adjuster robust to outliers, or try adding some of the other improvements discussed in (Engels, Stewénius, and Nistér 2006). Can you think of any other ways to make your algorithm even faster or more robust?

Ex 11.12: Match move and augmented reality. Use the results of the previous exercise to superimpose a rendered 3D model on top of video. See Section 11.4.4 for more details and ideas. Check for how "locked down" the objects are.

Ex 11.13: Line-based reconstruction. Augment the previously developed bundle adjuster to include lines, possibly with known 3D orientations.

Optionally, use co-planar sets of points and lines to hypothesize planes and to enforce co-planarity (Schaffalitzky and Zisserman 2002; Robertson and Cipolla 2002).

Ex 11.14: Flexible bundle adjuster. Design a bundle adjuster that allows for arbitrary chains of transformations and prior knowledge about the unknowns, as suggested in Figures 11.14–11.15.

Ex 11.15: Unordered image matching. Compute the camera pose and 3D structure of a scene from an arbitrary collection of photographs (Brown and Lowe 2005; Snavely, Seitz, and Szeliski 2006).

Ex 11.16: Augmented reality toolkits. Write a simple mobile AR app based on one of the widely used augmented reality frameworks such as ARKit or ARCore. What fun effects can you create? What are the conditions that make your AR system lose track? Can you move a large distance and come back to your original location without too much drift?