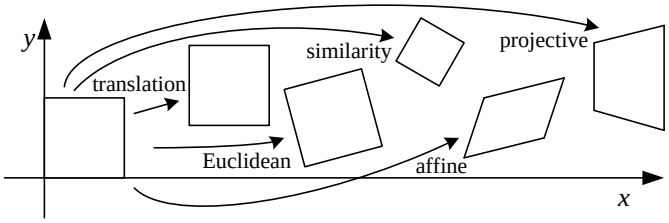

Figure 6.2 Basic set of 2D planar transformations

Once we have extracted features from images, the next stage in many vision algorithms is to match these features across different images (Section 4.1.3). An important component of this matching is to verify whether the set of matching features is geometrically consistent, e.g., whether the feature displacements can be described by a simple 2D or 3D geometric transformation. The computed motions can then be used in other applications such as image stitching (Chapter 9) or augmented reality (Section 6.2.3).

In this chapter, we look at the topic of geometric image registration, i.e., the computation of 2D and 3D transformations that map features in one image to another (Section 6.1). One special case of this problem is *pose estimation*, which is determining a camera's position relative to a known 3D object or scene (Section 6.2). Another case is the computation of a camera's *intrinsic calibration*, which consists of the internal parameters such as focal length and radial distortion (Section 6.3). In Chapter 7, we look at the related problems of how to estimate 3D point structure from 2D matches (*triangulation*) and how to simultaneously estimate 3D geometry and camera motion (*structure from motion*).

6.1 2D and 3D feature-based alignment

Feature-based alignment is the problem of estimating the motion between two or more sets of matched 2D or 3D points. In this section, we restrict ourselves to global *parametric* transformations, such as those described in Section 2.1.2 and shown in Table 2.1 and Figure 6.2, or higher order transformation for curved surfaces (Shashua and Toelg 1997; Can, Stewart, Roysam *et al.* 2002). Applications to non-rigid or elastic deformations (Bookstein 1989; Szeliski and Lavallée 1996; Torresani, Hertzmann, and Bregler 2008) are examined in Sections 8.3 and 12.6.4.

6.1.1 2D alignment using least squares

Given a set of matched feature points $\{(\boldsymbol{x}_i, \boldsymbol{x}'_i)\}$ and a planar parametric transformation[1] of the form

$$\boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{p}), \tag{6.1}$$

[1] For examples of non-planar parametric models, such as quadrics, see the work of Shashua and Toelg (1997); Shashua and Wexler (2001).

Transform	Matrix	Parameters $\boldsymbol{p}$	Jacobian $\boldsymbol{J}$
translation	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$	(t_x, t_y)	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
Euclidean	$\begin{bmatrix} c_\theta & -s_\theta & t_x \\ s_\theta & c_\theta & t_y \end{bmatrix}$	(t_x, t_y, θ)	$\begin{bmatrix} 1 & 0 & -s_\theta x - c_\theta y \\ 0 & 1 & c_\theta x - s_\theta y \end{bmatrix}$
similarity	$\begin{bmatrix} 1+a & -b & t_x \\ b & 1+a & t_y \end{bmatrix}$	(t_x, t_y, a, b)	$\begin{bmatrix} 1 & 0 & x & -y \\ 0 & 1 & y & x \end{bmatrix}$
affine	$\begin{bmatrix} 1+a_{00} & a_{01} & t_x \\ a_{10} & 1+a_{11} & t_y \end{bmatrix}$	$(t_x, t_y, a_{00}, a_{01}, a_{10}, a_{11})$	$\begin{bmatrix} 1 & 0 & x & y & 0 & 0 \\ 0 & 1 & 0 & 0 & x & y \end{bmatrix}$
projective	$\begin{bmatrix} 1+h_{00} & h_{01} & h_{02} \\ h_{10} & 1+h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix}$	$(h_{00}, h_{01}, \ldots, h_{21})$	(see Section 6.1.3)

Table 6.1 Jacobians of the 2D coordinate transformations $\boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{p})$ shown in Table 2.1, where we have re-parameterized the motions so that they are identity for $\boldsymbol{p} = 0$.

how can we produce the best estimate of the motion parameters $\boldsymbol{p}$? The usual way to do this is to use least squares, i.e., to minimize the sum of squared residuals

$$E_{\mathrm{LS}} = \sum_i \|\boldsymbol{r}_i\|^2 = \sum_i \|\boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{p}) - \boldsymbol{x}'_i\|^2, \tag{6.2}$$

where

$$\boldsymbol{r}_i = \boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{p}) - \boldsymbol{x}'_i = \hat{\boldsymbol{x}}'_i - \tilde{\boldsymbol{x}}'_i \tag{6.3}$$

is the *residual* between the measured location $\hat{\boldsymbol{x}}'_i$ and its corresponding current *predicted* location $\tilde{\boldsymbol{x}}'_i = \boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{p})$. (See Appendix A.2 for more on least squares and Appendix B.2 for a statistical justification.)

Many of the motion models presented in Section 2.1.2 and Table 2.1, i.e., translation, similarity, and affine, have a *linear* relationship between the amount of motion $\Delta\boldsymbol{x} = \boldsymbol{x}' - \boldsymbol{x}$ and the unknown parameters $\boldsymbol{p}$,

$$\Delta\boldsymbol{x} = \boldsymbol{x}' - \boldsymbol{x} = \boldsymbol{J}(\boldsymbol{x})\boldsymbol{p}, \tag{6.4}$$

where $\boldsymbol{J} = \partial\boldsymbol{f}/\partial\boldsymbol{p}$ is the *Jacobian* of the transformation $\boldsymbol{f}$ with respect to the motion parameters $\boldsymbol{p}$ (see Table 6.1). In this case, a simple *linear* regression (linear least squares problem) can be formulated as

$$\begin{aligned} E_{\mathrm{LLS}} &= \sum_i \|\boldsymbol{J}(\boldsymbol{x}_i)\boldsymbol{p} - \Delta\boldsymbol{x}_i\|^2 && (6.5) \\ &= \boldsymbol{p}^T \left[\sum_i \boldsymbol{J}^T(\boldsymbol{x}_i)\boldsymbol{J}(\boldsymbol{x}_i)\right] \boldsymbol{p} - 2\boldsymbol{p}^T \left[\sum_i \boldsymbol{J}^T(\boldsymbol{x}_i)\Delta\boldsymbol{x}_i\right] + \sum_i \|\Delta\boldsymbol{x}_i\|^2 && (6.6) \\ &= \boldsymbol{p}^T \boldsymbol{A}\boldsymbol{p} - 2\boldsymbol{p}^T \boldsymbol{b} + c. && (6.7) \end{aligned}$$

The minimum can be found by solving the symmetric positive definite (SPD) system of *normal equations*[2]

$$\boldsymbol{A}\boldsymbol{p} = \boldsymbol{b}, \tag{6.8}$$

where

$$\boldsymbol{A} = \sum_i \boldsymbol{J}^T(\boldsymbol{x}_i)\boldsymbol{J}(\boldsymbol{x}_i) \tag{6.9}$$

is called the *Hessian* and $\boldsymbol{b} = \sum_i \boldsymbol{J}^T(\boldsymbol{x}_i)\Delta\boldsymbol{x}_i$. For the case of pure translation, the resulting equations have a particularly simple form, i.e., the translation is the average translation between corresponding points or, equivalently, the translation of the point centroids.

Uncertainty weighting. The above least squares formulation assumes that all feature points are matched with the same accuracy. This is often not the case, since certain points may fall into more textured regions than others. If we associate a scalar variance estimate σ_i^2 with each correspondence, we can minimize the *weighted least squares* problem instead,[3]

$$E_{\mathrm{WLS}} = \sum_i \sigma_i^{-2}\|\boldsymbol{r}_i\|^2. \tag{6.10}$$

As shown in Section 8.1.3, a covariance estimate for patch-based matching can be obtained by multiplying the inverse of the *patch Hessian* $\boldsymbol{A}_i$ (8.55) with the per-pixel noise covariance σ_n^2 (8.44). Weighting each squared residual by its inverse covariance $\Sigma_i^{-1} = \sigma_n^{-2}\boldsymbol{A}_i$ (which is called the *information matrix*), we obtain

$$E_{\mathrm{CWLS}} = \sum_i \|\boldsymbol{r}_i\|^2_{\Sigma_i^{-1}} = \sum_i \boldsymbol{r}_i^T\Sigma_i^{-1}\boldsymbol{r}_i = \sum_i \sigma_n^{-2}\boldsymbol{r}_i^T\boldsymbol{A}_i\boldsymbol{r}_i. \tag{6.11}$$

6.1.2 *Application*: Panography

One of the simplest (and most fun) applications of image alignment is a special form of image stitching called *panography*. In a panograph, images are translated and optionally rotated and scaled before being blended with simple averaging (Figure 6.3). This process mimics the photographic collages created by artist David Hockney, although his compositions use an opaque overlay model, being created out of regular photographs.

In most of the examples seen on the Web, the images are aligned by hand for best artistic effect.[4] However, it is also possible to use feature matching and alignment techniques to perform the registration automatically (Nomura, Zhang, and Nayar 2007; Zelnik-Manor and Perona 2007).

Consider a simple translational model. We want all the corresponding features in different images to line up as best as possible. Let $\boldsymbol{t}_j$ be the location of the jth image coordinate frame in the global composite frame and $\boldsymbol{x}_{ij}$ be the location of the ith matched feature in the jth image. In order to align the images, we wish to minimize the least squares error

$$E_{\mathrm{PLS}} = \sum_{ij} \|(\boldsymbol{t}_j + \boldsymbol{x}_{ij}) - \boldsymbol{x}_i\|^2, \tag{6.12}$$

[2] For poorly conditioned problems, it is better to use QR decomposition on the set of linear equations $\boldsymbol{J}(\boldsymbol{x}_i)\boldsymbol{p} = \Delta\boldsymbol{x}_i$ instead of the normal equations (Björck 1996; Golub and Van Loan 1996). However, such conditions rarely arise in image registration.

[3] Problems where each measurement can have a different variance or certainty are called *heteroscedastic models*.

[4] http://www.flickr.com/groups/panography/.

Figure 6.3 A simple panograph consisting of three images automatically aligned with a translational model and then averaged together.

where $\boldsymbol{x}_i$ is the consensus (average) position of feature i in the global coordinate frame. (An alternative approach is to register each pair of overlapping images separately and then compute a consensus location for each frame—see Exercise 6.2.)

The above least squares problem is indeterminate (you can add a constant offset to all the frame and point locations $\boldsymbol{t}_j$ and $\boldsymbol{x}_i$). To fix this, either pick one frame as being at the origin or add a constraint to make the average frame offsets be 0.

The formulas for adding rotation and scale transformations are straightforward and are left as an exercise (Exercise 6.2). See if you can create some collages that you would be happy to share with others on the Web.

6.1.3 Iterative algorithms

While linear least squares is the simplest method for estimating parameters, most problems in computer vision do not have a simple linear relationship between the measurements and the unknowns. In this case, the resulting problem is called *non-linear least squares* or *non-linear regression*.

Consider, for example, the problem of estimating a rigid Euclidean 2D transformation (translation plus rotation) between two sets of points. If we parameterize this transformation by the translation amount (t_x, t_y) and the rotation angle θ, as in Table 2.1, the Jacobian of this transformation, given in Table 6.1, depends on the current value of θ. Notice how in Table 6.1, we have re-parameterized the motion matrices so that they are always the identity at the origin $\boldsymbol{p} = 0$, which makes it easier to initialize the motion parameters.

To minimize the non-linear least squares problem, we iteratively find an update $\Delta\boldsymbol{p}$ to the current parameter estimate $\boldsymbol{p}$ by minimizing

$$E_{\mathrm{NLS}}(\Delta\boldsymbol{p}) = \sum_i \|\boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{p} + \Delta\boldsymbol{p}) - \boldsymbol{x}_i'\|^2 \tag{6.13}$$

$$\approx \sum_i \|\boldsymbol{J}(\boldsymbol{x}_i; \boldsymbol{p})\Delta\boldsymbol{p} - \boldsymbol{r}_i\|^2 \tag{6.14}$$

$$= \Delta \boldsymbol{p}^T \left[\sum_i \boldsymbol{J}^T \boldsymbol{J} \right] \Delta \boldsymbol{p} - 2\Delta \boldsymbol{p}^T \left[\sum_i \boldsymbol{J}^T \boldsymbol{r}_i \right] + \sum_i \|\boldsymbol{r}_i\|^2 \quad (6.15)$$

$$= \Delta \boldsymbol{p}^T \boldsymbol{A} \Delta \boldsymbol{p} - 2\Delta \boldsymbol{p}^T \boldsymbol{b} + c, \quad (6.16)$$

where the "Hessian"[5] $\boldsymbol{A}$ is the same as Equation (6.9) and the right hand side vector

$$\boldsymbol{b} = \sum_i \boldsymbol{J}^T(\boldsymbol{x}_i)\boldsymbol{r}_i \quad (6.17)$$

is now a Jacobian-weighted sum of residual vectors. This makes intuitive sense, as the parameters are pulled in the direction of the prediction error with a strength proportional to the Jacobian.

Once $\boldsymbol{A}$ and $\boldsymbol{b}$ have been computed, we solve for $\Delta \boldsymbol{p}$ using

$$(\boldsymbol{A} + \lambda \text{diag}(\boldsymbol{A}))\Delta \boldsymbol{p} = \boldsymbol{b}, \quad (6.18)$$

and update the parameter vector $\boldsymbol{p} \leftarrow \boldsymbol{p} + \Delta \boldsymbol{p}$ accordingly. The parameter λ is an additional damping parameter used to ensure that the system takes a "downhill" step in energy (squared error) and is an essential component of the Levenberg–Marquardt algorithm (described in more detail in Appendix A.3). In many applications, it can be set to 0 if the system is successfully converging.

For the case of our 2D translation+rotation, we end up with a 3×3 set of normal equations in the unknowns $(\delta t_x, \delta t_y, \delta\theta)$. An initial guess for (t_x, t_y, θ) can be obtained by fitting a four-parameter similarity transform in (t_x, t_y, c, s) and then setting $\theta = \tan^{-1}(s/c)$. An alternative approach is to estimate the translation parameters using the centroids of the 2D points and to then estimate the rotation angle using polar coordinates (Exercise 6.3).

For the other 2D motion models, the derivatives in Table 6.1 are all fairly straightforward, except for the projective 2D motion (homography), which arises in image-stitching applications (Chapter 9). These equations can be re-written from (2.21) in their new parametric form as

$$x' = \frac{(1+h_{00})x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + 1} \text{ and } y' = \frac{h_{10}x + (1+h_{11})y + h_{12}}{h_{20}x + h_{21}y + 1}. \quad (6.19)$$

The Jacobian is therefore

$$\boldsymbol{J} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{p}} = \frac{1}{D} \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'y \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y \end{bmatrix}, \quad (6.20)$$

where $D = h_{20}x + h_{21}y + 1$ is the denominator in (6.19), which depends on the current parameter settings (as do x' and y').

An initial guess for the eight unknowns $\{h_{00}, h_{01}, \ldots, h_{21}\}$ can be obtained by multiplying both sides of the equations in (6.19) through by the denominator, which yields the linear set of equations,

$$\begin{bmatrix} \hat{x}' - x \\ \hat{y}' - y \end{bmatrix} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -\hat{x}'x & -\hat{x}'y \\ 0 & 0 & 0 & x & y & 1 & -\hat{y}'x & -\hat{y}'y \end{bmatrix} \begin{bmatrix} h_{00} \\ \vdots \\ h_{21} \end{bmatrix}. \quad (6.21)$$

[5] The "Hessian" $\boldsymbol{A}$ is not the true Hessian (second derivative) of the non-linear least squares problem (6.13). Instead, it is the approximate Hessian, which neglects second (and higher) order derivatives of $\boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{p} + \Delta \boldsymbol{p})$.

However, this is not optimal from a statistical point of view, since the denominator D, which was used to multiply each equation, can vary quite a bit from point to point.[6]

One way to compensate for this is to *reweight* each equation by the inverse of the current estimate of the denominator, D,

$$\frac{1}{D}\begin{bmatrix} \hat{x}' - x \\ \hat{y}' - y \end{bmatrix} = \frac{1}{D}\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -\hat{x}'x & -\hat{x}'y \\ 0 & 0 & 0 & x & y & 1 & -\hat{y}'x & -\hat{y}'y \end{bmatrix} \begin{bmatrix} h_{00} \\ \vdots \\ h_{21} \end{bmatrix}. \tag{6.22}$$

While this may at first seem to be the exact same set of equations as (6.21), because least squares is being used to solve the over-determined set of equations, the weightings *do* matter and produce a different set of normal equations that performs better in practice.

The most principled way to do the estimation, however, is to directly minimize the squared residual equations (6.13) using the Gauss–Newton approximation, i.e., performing a first-order Taylor series expansion in $\boldsymbol{p}$, as shown in (6.14), which yields the set of equations

$$\begin{bmatrix} \hat{x}' - \tilde{x}' \\ \hat{y}' - \tilde{y}' \end{bmatrix} = \frac{1}{D}\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -\tilde{x}'x & -\tilde{x}'y \\ 0 & 0 & 0 & x & y & 1 & -\tilde{y}'x & -\tilde{y}'y \end{bmatrix} \begin{bmatrix} \Delta h_{00} \\ \vdots \\ \Delta h_{21} \end{bmatrix}. \tag{6.23}$$

While these look similar to (6.22), they differ in two important respects. First, the left hand side consists of unweighted *prediction errors* rather than point displacements and the solution vector is a *perturbation* to the parameter vector $\boldsymbol{p}$. Second, the quantities inside $\boldsymbol{J}$ involve *predicted* feature locations $(\tilde{x}', \tilde{y}')$ instead of *sensed* feature locations $(\hat{x}', \hat{y}')$. Both of these differences are subtle and yet they lead to an algorithm that, when combined with proper checking for downhill steps (as in the Levenberg–Marquardt algorithm), will converge to a local minimum. Note that iterating Equations (6.22) is not guaranteed to converge, since it is not minimizing a well-defined energy function.

Equation (6.23) is analogous to the *additive* algorithm for direct intensity-based registration (Section 8.2), since the change to the full transformation is being computed. If we prepend an incremental homography to the current homography instead, i.e., we use a *compositional* algorithm (described in Section 8.2), we get $D = 1$ (since $\boldsymbol{p} = 0$) and the above formula simplifies to

$$\begin{bmatrix} \hat{x}' - x \\ \hat{y}' - y \end{bmatrix} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x^2 & -xy \\ 0 & 0 & 0 & x & y & 1 & -xy & -y^2 \end{bmatrix} \begin{bmatrix} \Delta h_{00} \\ \vdots \\ \Delta h_{21} \end{bmatrix}, \tag{6.24}$$

where we have replaced $(\tilde{x}', \tilde{y}')$ with (x, y) for conciseness. (Notice how this results in the same Jacobian as (8.63).)

[6] Hartley and Zisserman (2004) call this strategy of forming linear equations from rational equations the *direct linear transform*, but that term is more commonly associated with pose estimation (Section 6.2). Note also that our definition of the h_{ij} parameters differs from that used in their book, since we define h_{ii} to be the *difference* from unity and we do not leave h_{22} as a free parameter, which means that we cannot handle certain extreme homographies.

6.1.4 Robust least squares and RANSAC

While regular least squares is the method of choice for measurements where the noise follows a normal (Gaussian) distribution, more robust versions of least squares are required when there are outliers among the correspondences (as there almost always are). In this case, it is preferable to use an *M-estimator* (Huber 1981; Hampel, Ronchetti, Rousseeuw *et al.* 1986; Black and Rangarajan 1996; Stewart 1999), which involves applying a robust penalty function $\rho(r)$ to the residuals

$$E_{\mathrm{RLS}}(\Delta \boldsymbol{p}) = \sum_i \rho(\|\boldsymbol{r}_i\|) \tag{6.25}$$

instead of squaring them.

We can take the derivative of this function with respect to $\boldsymbol{p}$ and set it to 0,

$$\sum_i \psi(\|\boldsymbol{r}_i\|) \frac{\partial \|\boldsymbol{r}_i\|}{\partial \boldsymbol{p}} = \sum_i \frac{\psi(\|\boldsymbol{r}_i\|)}{\|\boldsymbol{r}_i\|} \boldsymbol{r}_i^T \frac{\partial \boldsymbol{r}_i}{\partial \boldsymbol{p}} = 0, \tag{6.26}$$

where $\psi(r) = \rho'(r)$ is the derivative of ρ and is called the *influence function*. If we introduce a *weight function*, $w(r) = \Psi(r)/r$, we observe that finding the stationary point of (6.25) using (6.26) is equivalent to minimizing the *iteratively reweighted least squares* (IRLS) problem

$$E_{\mathrm{IRLS}} = \sum_i w(\|\boldsymbol{r}_i\|) \|\boldsymbol{r}_i\|^2, \tag{6.27}$$

where the $w(\|\boldsymbol{r}_i\|)$ play the same local weighting role as σ_i^{-2} in (6.10). The IRLS algorithm alternates between computing the influence functions $w(\|\boldsymbol{r}_i\|)$ and solving the resulting weighted least squares problem (with fixed w values). Other incremental robust least squares algorithms can be found in the work of Sawhney and Ayer (1996); Black and Anandan (1996); Black and Rangarajan (1996); Baker, Gross, Ishikawa *et al.* (2003) and textbooks and tutorials on robust statistics (Huber 1981; Hampel, Ronchetti, Rousseeuw *et al.* 1986; Rousseeuw and Leroy 1987; Stewart 1999).

While M-estimators can definitely help reduce the influence of outliers, in some cases, starting with too many outliers will prevent IRLS (or other gradient descent algorithms) from converging to the global optimum. A better approach is often to find a starting set of *inlier* correspondences, i.e., points that are consistent with a dominant motion estimate.[7]

Two widely used approaches to this problem are called RANdom SAmple Consensus, or RANSAC for short (Fischler and Bolles 1981), and *least median of squares* (LMS) (Rousseeuw 1984). Both techniques start by selecting (at random) a subset of k correspondences, which is then used to compute an initial estimate for $\boldsymbol{p}$. The *residuals* of the full set of correspondences are then computed as

$$\boldsymbol{r}_i = \tilde{\boldsymbol{x}}_i'(\boldsymbol{x}_i; \boldsymbol{p}) - \hat{\boldsymbol{x}}_i', \tag{6.28}$$

where $\tilde{\boldsymbol{x}}_i'$ are the *estimated* (mapped) locations and $\hat{\boldsymbol{x}}_i'$ are the sensed (detected) feature point locations.

The RANSAC technique then counts the number of *inliers* that are within ϵ of their predicted location, i.e., whose $\|\boldsymbol{r}_i\| \leq \epsilon$. (The ϵ value is application dependent but is often around 1–3 pixels.) Least median of squares finds the median value of the $\|\boldsymbol{r}_i\|^2$ values. The

[7] For pixel-based alignment methods (Section 8.1.1), hierarchical (coarse-to-fine) techniques are often used to lock onto the *dominant motion* in a scene.

k	p	S
3	0.5	35
6	0.6	97
6	0.5	293

Table 6.2 Number of trials S to attain a 99% probability of success (Stewart 1999).

random selection process is repeated S times and the sample set with the largest number of inliers (or with the smallest median residual) is kept as the final solution. Either the initial parameter guess $\boldsymbol{p}$ or the full set of computed inliers is then passed on to the next data fitting stage.

When the number of measurements is quite large, it may be preferable to only score a subset of the measurements in an initial round that selects the most plausible hypotheses for additional scoring and selection. This modification of RANSAC, which can significantly speed up its performance, is called *Preemptive RANSAC* (Nistér 2003). In another variant on RANSAC called PROSAC (PROgressive SAmple Consensus), random samples are initially added from the most "confident" matches, thereby speeding up the process of finding a (statistically) likely good set of inliers (Chum and Matas 2005).

To ensure that the random sampling has a good chance of finding a true set of inliers, a sufficient number of trials S must be tried. Let p be the probability that any given correspondence is valid and P be the total probability of success after S trials. The likelihood in one trial that all k random samples are inliers is p^k. Therefore, the likelihood that S such trials will all fail is

$$1 - P = (1 - p^k)^S \tag{6.29}$$

and the required minimum number of trials is

$$S = \frac{\log(1 - P)}{\log(1 - p^k)}. \tag{6.30}$$

Stewart (1999) gives examples of the required number of trials S to attain a 99% probability of success. As you can see from Table 6.2, the number of trials grows quickly with the number of sample points used. This provides a strong incentive to use the *minimum* number of sample points k possible for any given trial, which is how RANSAC is normally used in practice.

Uncertainty modeling

In addition to robustly computing a good alignment, some applications require the computation of uncertainty (see Appendix B.6). For linear problems, this estimate can be obtained by inverting the Hessian matrix (6.9) and multiplying it by the feature position noise (if these have not already been used to weight the individual measurements, as in Equations (6.10) and 6.11)). In statistics, the Hessian, which is the inverse covariance, is sometimes called the (Fisher) *information matrix* (Appendix B.1.1).

When the problem involves non-linear least squares, the inverse of the Hessian matrix provides the *Cramer–Rao lower bound* on the covariance matrix, i.e., it provides the *minimum*

amount of covariance in a given solution, which can actually have a wider spread ("longer tails") if the energy flattens out away from the local minimum where the optimal solution is found.

6.1.5 3D alignment

Instead of aligning 2D sets of image features, many computer vision applications require the alignment of 3D points. In the case where the 3D transformations are linear in the motion parameters, e.g., for translation, similarity, and affine, regular least squares (6.5) can be used.

The case of rigid (Euclidean) motion,

$$E_{\mathrm{R3D}} = \sum_i \|\boldsymbol{x}'_i - \boldsymbol{R}\boldsymbol{x}_i - \boldsymbol{t}\|^2, \tag{6.31}$$

which arises more frequently and is often called the *absolute orientation* problem (Horn 1987), requires slightly different techniques. If only scalar weightings are being used (as opposed to full 3D per-point anisotropic covariance estimates), the weighted centroids of the two point clouds $\boldsymbol{c}$ and $\boldsymbol{c}'$ can be used to estimate the translation $\boldsymbol{t} = \boldsymbol{c}' - \boldsymbol{R}\boldsymbol{c}$.[8] We are then left with the problem of estimating the rotation between two sets of points $\{\hat{\boldsymbol{x}}_i = \boldsymbol{x}_i - \boldsymbol{c}\}$ and $\{\hat{\boldsymbol{x}}'_i = \boldsymbol{x}'_i - \boldsymbol{c}'\}$ that are both centered at the origin.

One commonly used technique is called the *orthogonal Procrustes algorithm* (Golub and Van Loan 1996, p. 601) and involves computing the singular value decomposition (SVD) of the 3×3 correlation matrix

$$\boldsymbol{C} = \sum_i \hat{\boldsymbol{x}}'\hat{\boldsymbol{x}}^T = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T. \tag{6.32}$$

The rotation matrix is then obtained as $\boldsymbol{R} = \boldsymbol{U}\boldsymbol{V}^T$. (Verify this for yourself when $\hat{\boldsymbol{x}}' = \boldsymbol{R}\hat{\boldsymbol{x}}$.)

Another technique is the absolute orientation algorithm (Horn 1987) for estimating the unit quaternion corresponding to the rotation matrix $\boldsymbol{R}$, which involves forming a 4×4 matrix from the entries in $\boldsymbol{C}$ and then finding the eigenvector associated with its largest positive eigenvalue.

Lorusso, Eggert, and Fisher (1995) experimentally compare these two techniques to two additional techniques proposed in the literature, but find that the difference in accuracy is negligible (well below the effects of measurement noise).

In situations where these closed-form algorithms are not applicable, e.g., when full 3D covariances are being used or when the 3D alignment is part of some larger optimization, the incremental rotation update introduced in Section 2.1.4 (2.35–2.36), which is parameterized by an instantaneous rotation vector $\boldsymbol{\omega}$, can be used (See Section 9.1.3 for an application to image stitching.)

In some situations, e.g., when merging range data maps, the correspondence between data points is not known *a priori*. In this case, iterative algorithms that start by matching nearby points and then update the most likely correspondence can be used (Besl and McKay 1992; Zhang 1994; Szeliski and Lavallée 1996; Gold, Rangarajan, Lu *et al.* 1998; David, DeMenthon, Duraiswami *et al.* 2004; Li and Hartley 2007; Enqvist, Josephson, and Kahl 2009). These techniques are discussed in more detail in Section 12.2.1.

[8] When full covariances are used, they are transformed by the rotation and so a closed-form solution for translation is not possible.

6.2 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 6.3. 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), with extensions to larger numbers of points collectively known as PnP (Haralick, Lee, Ottenberg *et al.* 1994; Quan and Lan 1999; Moreno-Noguer, Lepetit, and Fua 2007).

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.

6.2.1 Linear algorithms

The simplest way to recover the pose of the camera is to form a set of linear equations analogous to those used for 2D motion estimation (6.19) from the camera matrix form of perspective projection (2.55–2.56),

$$x_i = \frac{p_{00}X_i + p_{01}Y_i + p_{02}Z_i + p_{03}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}} \tag{6.33}$$

$$y_i = \frac{p_{10}X_i + p_{11}Y_i + p_{12}Z_i + p_{13}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}}, \tag{6.34}$$

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 6.4). As with (6.21), this system of equations can be solved in a linear fashion for the unknowns in the camera matrix $\boldsymbol{P}$ by multiplying the denominator on both sides of the equation.[9] The resulting algorithm is called the *direct linear transform* (DLT) and is commonly attributed to Sutherland (1974). (For a more in-depth discussion, refer to the work of Hartley and Zisserman (2004).) In order to compute the 12 (or 11) unknowns in $\boldsymbol{P}$, at least six correspondences between 3D and 2D locations must be known.

As with the case of estimating homographies (6.21–6.23), more accurate results for the entries in $\boldsymbol{P}$ can be obtained by directly minimizing the set of Equations (6.33–6.34) using non-linear least squares with a small number of iterations.

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

$$\boldsymbol{P} = \boldsymbol{K}[\boldsymbol{R}|\boldsymbol{t}]. \tag{6.35}$$

Since $\boldsymbol{K}$ is by convention upper-triangular (see the discussion in Section 2.1.5), both $\boldsymbol{K}$ and $\boldsymbol{R}$ can be obtained from the front 3×3 sub-matrix of $\boldsymbol{P}$ using RQ factorization (Golub and Van Loan 1996).[10]

[9] Because $\boldsymbol{P}$ 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.

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

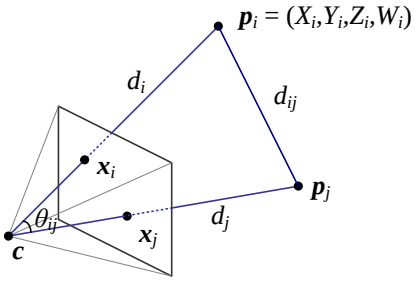

Figure 6.4 Pose estimation by the direct linear transform and by measuring visual angles and distances between pairs of points.

In most applications, however, we have some prior knowledge about the intrinsic calibration matrix $\boldsymbol{K}$, e.g., that the pixels are square, the skew is very small, and the optical center is near the center of the image (2.57–2.59). Such constraints can be incorporated into a non-linear minimization of the parameters in $\boldsymbol{K}$ and $(\boldsymbol{R}, \boldsymbol{t})$, as described in Section 6.2.2.

In the case where the camera is already calibrated, i.e., the matrix $\boldsymbol{K}$ is known (Section 6.3), we can perform pose estimation using as few as three points (Fischler and Bolles 1981; Haralick, Lee, Ottenberg *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 $\hat{\boldsymbol{x}}_i$ and $\hat{\boldsymbol{x}}_j$ must be the same as the angle between their corresponding 3D points $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ (Figure 6.4).

Given a set of corresponding 2D and 3D points $\{(\hat{\boldsymbol{x}}_i, \boldsymbol{p}_i)\}$, where the $\hat{\boldsymbol{x}}_i$ are unit directions obtained by transforming 2D pixel measurements $\boldsymbol{x}_i$ to unit norm 3D directions $\hat{\boldsymbol{x}}_i$ through the inverse calibration matrix $\boldsymbol{K}$,

$$\hat{\boldsymbol{x}}_i = \mathcal{N}(\boldsymbol{K}^{-1}\boldsymbol{x}_i) = \boldsymbol{K}^{-1}\boldsymbol{x}_i / \|\boldsymbol{K}^{-1}\boldsymbol{x}_i\|, \tag{6.36}$$

the unknowns are the distances d_i from the camera origin $\boldsymbol{c}$ to the 3D points $\boldsymbol{p}_i$, where

$$\boldsymbol{p}_i = d_i \hat{\boldsymbol{x}}_i + \boldsymbol{c} \tag{6.37}$$

(Figure 6.4). The cosine law for triangle $\Delta(\boldsymbol{c}, \boldsymbol{p}_i, \boldsymbol{p}_j)$ gives us

$$f_{ij}(d_i, d_j) = d_i^2 + d_j^2 - 2 d_i d_j c_{ij} - d_{ij}^2 = 0, \tag{6.38}$$

where

$$c_{ij} = \cos\theta_{ij} = \hat{\boldsymbol{x}}_i \cdot \hat{\boldsymbol{x}}_j \tag{6.39}$$

and

$$d_{ij}^2 = \|\boldsymbol{p}_i - \boldsymbol{p}_j\|^2. \tag{6.40}$$

We can take any triplet of constraints (f_{ij}, f_{ik}, f_{jk}) and eliminate the d_j and d_k using Sylvester resultants (Cox, Little, and O'Shea 2007) to obtain a quartic equation in d_i^2,

$$g_{ijk}(d_i^2) = a_4 d_i^8 + a_3 d_i^6 + a_2 d_i^4 + a_1 d_i^2 + a_0 = 0. \tag{6.41}$$

Given five or more correspondences, we can generate $\frac{(n-1)(n-2)}{2}$ triplets to obtain a linear estimate (using SVD) for the values of $(d_i^8, d_i^6, d_i^4, d_i^2)$ (Quan and Lan 1999). Estimates for

d_i^2 can computed as ratios of successive d_i^{2n+2}/d_i^{2n} estimates and these can be averaged to obtain a final estimate of d_i^2 (and hence d_i).

Once the individual estimates of the d_i distances have been computed, we can generate a 3D structure consisting of the scaled point directions $d_i\hat{\boldsymbol{x}}_i$, which can then be aligned with the 3D point cloud $\{\boldsymbol{p}_i\}$ using absolute orientation (Section 6.1.5) to obtained the desired pose estimate. Quan and Lan (1999) give 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 more recent alternatives and also gives a lower complexity algorithm that typically produces more accurate results.

Unfortunately, because minimal PnP solutions can be quite noise sensitive and also suffer from *bas-relief ambiguities* (e.g., depth reversals) (Section 7.4.3), it is often preferable to use the linear six-point algorithm to guess an initial pose and then optimize this estimate using the iterative technique described in Section 6.2.2.

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".

6.2.2 Iterative 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 $(\boldsymbol{R}, \boldsymbol{t})$ and optionally $\boldsymbol{K}$ using non-linear least squares (Tsai 1987; Bogart 1991; Gleicher and Witkin 1992). We can write the projection equations as

$$\boldsymbol{x}_i = \boldsymbol{f}(\boldsymbol{p}_i; \boldsymbol{R}, \boldsymbol{t}, \boldsymbol{K}) \tag{6.42}$$

and iteratively minimize the robustified linearized reprojection errors

$$E_{\mathrm{NLP}} = \sum_i \rho\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{R}}\Delta\boldsymbol{R} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{t}}\Delta\boldsymbol{t} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{K}}\Delta\boldsymbol{K} - \boldsymbol{r}_i\right), \tag{6.43}$$

where $\boldsymbol{r}_i = \tilde{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_i$ 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). 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 (6.11).

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 $\boldsymbol{p}_i$ by a simple transformation such as translation, rotation, or perspective division (Figure 6.5). The resulting projection equations can be written as

$$\boldsymbol{y}^{(1)} = \boldsymbol{f}_{\mathrm{T}}(\boldsymbol{p}_i; \boldsymbol{c}_j) = \boldsymbol{p}_i - \boldsymbol{c}_j, \tag{6.44}$$

$$\boldsymbol{y}^{(2)} = \boldsymbol{f}_{\mathrm{R}}(\boldsymbol{y}^{(1)}; \boldsymbol{q}_j) = \boldsymbol{R}(\boldsymbol{q}_j)\,\boldsymbol{y}^{(1)}, \tag{6.45}$$

$$\boldsymbol{y}^{(3)} = \boldsymbol{f}_{\mathrm{P}}(\boldsymbol{y}^{(2)}) = \frac{\boldsymbol{y}^{(2)}}{z^{(2)}}, \tag{6.46}$$

$$\boldsymbol{x}_i = \boldsymbol{f}_{\mathrm{C}}(\boldsymbol{y}^{(3)}; \boldsymbol{k}) = \boldsymbol{K}(\boldsymbol{k})\,\boldsymbol{y}^{(3)}. \tag{6.47}$$