CHAPTER 14

Registration

Many camera movements cannot be modelled by a pure translation of the image. There might be rotation or scaling or (as Section 41.2 shows) a projective transformation. It is natural to consider a mosaic that uses a richer family of transformations. However, the current registration procedure – compute the cost function at many different translations, and choose the best – becomes unwieldy when there are more transformation parameters to deal with. The number of objective function values needed grows exponentially in the number of parameters in the transformation, so even translation and rotation become unwieldy.

14.1 REGISTRATION PROBLEMS

A natural approach to building mosaics with richer families of transformation uses interest points. You find interest points in first and second image; establish correspondences by determining which points represent the same image window; then compute a transformation from these correspondences. This is an instance of a general recipe that applies to a very wide range of problems

14.1.1 General Registration

The general problem looks like this. You have two *point clouds* – two sets of points with no other structure. The locations of the interest points in an image form a 2D point cloud, but point clouds can have any dimension. Write $\mathcal{P}$ for a point cloud whose i'th point is $\mathbf{p}_i$ and so on. Write $\mathcal{X}$ and $\mathcal{Y}$ for the two point clouds, $\mathcal{T}$ for a transformation, $\mathcal{T}(\mathbf{y})$ for the transformed version of the point $\mathbf{y}$, and $\mathcal{T}(\mathcal{Y})$ for the transformed version of the point cloud. Further, you know that there is a transformation $\mathcal{T}$ so that $\mathcal{T}(\mathcal{Y})$ is "close" to $\mathcal{X}$. You must find this transformation.

In the simplest case, for each point in $\mathcal{Y}$ there is a unique corresponding point in $\mathcal{X}$ *and* for each point in $\mathcal{X}$ there is a unique corresponding point in $\mathcal{Y}$ *and you know which point corresponds to which.* Here the point clouds must have the same number of points in them. Solutions to this case (Section 14.2) are relatively straightforward and provide building blocks for more complicated cases. In most cases, you need to estimate correspondences from the data. Doing so creates issues that vary from application to application.

14.1.2 Application: Registering Images

Building mosaics is one reason to register two images, but it isn't the only one. Another application is change detection. You have (say) an aerial image of a suburb taken ten years ago, and another taken recently. One way to know what has changed is to register the images and look at the differences. This line of reasoning is widely useful. For example, you have a x-ray image of a breast at a previous exam and now – looking at the differences might reveal changes that indicate disease or the progress

of disease. Yet another application registers different types of image. You might have a thermal image of the suburb and a color image of that suburb: registering them allows you to describe imaged points in more detail – how hot they are *and* what color they are. This recipe is particularly useful in medical applications, where it is common to have two images of some structure obtained using different procedures.

Point clouds aren't an excellent abstration for image registration, because you know more about the interest points than just where they are. You know what the image looks like around the interest point as well. This means you could obtain correspondences by finding interest points, computing local coordinate systems and descriptors, then matching interest points in $\mathcal{A}$ to those in $\mathcal{B}$ using approximate nearest neighbors.

The procedure looks like this. For each interest point in $\mathcal{A}$ (write the feature vector of the i'th as $\mathbf{a}_i$), you find the interest point in $\mathcal{B}$ whose feature vector is most similar. Assume that this is the j'th point in $\mathcal{B}$, so that $(\mathbf{a}_i - \mathbf{b}_j)^T(\mathbf{a}_i - \mathbf{b}_j)$ is smaller than $(\mathbf{a}_i - \mathbf{b}_k)^T(\mathbf{a}_i - \mathbf{b}_k)$ for $k \neq j$. Now check the matching is *symmetric* – if $\mathbf{b}_j$ is closest to $\mathbf{a}_i$ in $\mathcal{B}$, then $\mathbf{a}_i$ should be the closest point in $\mathcal{A}$ to $\mathbf{b}_j$. Now look at pairs where the matching is symmetric *and* the distance between matched feature vectors is small – these are likely to be corresponding points if the distance threshold is small enough.

Some of the resulting correspondences are likely to be wrong. As an extreme example, consider registering two images of checkerboards. In each image, there will be an awful lot of corners of the same pattern in each image. The interest point matching procedure finds pairs that look like one another, and so has no way to choose which pairs are good. Generally, with good interest point descriptors, the image registration case produces a relatively large fraction of good correspondences.

14.1.3 Application: Registering 3D Point Clouds

LIDAR sensors query depth at a grid of sampling directions which usually lie in a cylinder around the sensor, and report (x, y, z) points. The sensor does not usually report anything else about each sample, so the point cloud is a fairly good abstraction here. You have a vehicle with a LIDAR sensor, and drive it around an indoor area taking LIDAR measurements. You estimate the location of the car each time you measure by looking at, say, wheel revolutions or GPS. This gives you a fair estimate of the registration between measurements, and want to improve this registration to build a LIDAR map of the area. This case is rather different to image registration because the points will have no associated descriptions so you can't establish correspondence using descriptors. However, you have good initial estimates of the registration.

Once you have the map of the area, the car moves to some unknown location and takes a LIDAR measurement. You can tell where the car is by registering the LIDAR measurement to the map. Again, you can't establish correspondence using descriptors. This version of the registration problem has some interesting problems that come from sampling issues (below).

14.1.4 Application: Registering Meshes to Point Clouds

Another standard problem is to find instances of a CAD model in data from a LIDAR sensor. For example, you might have a CAD model of a car, and want to find if that car appears in the LIDAR data and where it appears. Further, CAD models can always be reduced to triangle meshes. A natural procedure is to sample points on the mesh model to get a point cloud, then treat the problem as a point cloud registration problem. Again, you can't get descriptions of points that are good enough to estimate correspondence accurately. There are quite likely to be many bad correspondences, because the LIDAR data has many points that don't lie on the car.

14.2 WEIGHTED LEAST SQUARES REGISTRATION WITH EXACT CORRESPONDENCES

The core engine of any registration solution is weighted least squares where the correspondences are known. How one uses this engine depends somewhat on the problem. Finding a solution is always an optimization problem, but the details of this problem differ from transformation to transformation.

14.2.1 Affine Transformations

For an affine transformation, $\mathcal{T}(\mathbf{y})$ is $\mathcal{M}\mathbf{y}+\mathbf{t}$. Further, there is a transformation $\mathcal{T}$ so that $\mathcal{T}(\mathbf{y}_i)$ is close to $\mathbf{x}_i$ for each i. Write $\mathbf{r}_i$ for the vector from the transformed $\mathbf{y}_i$ to $\mathbf{x}_i$, so

$$\mathbf{r}_i(\mathcal{M},\mathbf{t}) = (\mathbf{x}_i - (\mathcal{M}\mathbf{y}_i + \mathbf{t}))$$

and

$$C_u(\mathcal{M},\mathbf{t}) = (1/N)\sum_i \mathbf{r}_i^T\mathbf{r}_i$$

should be small. Because it will be useful later, assume that there is a weight w_i for each pair and work with

$$C(\mathcal{M},\mathbf{t}) = \sum_i w_i\mathbf{r}_i^T\mathbf{r}_i$$

where $w_i = 1/N$ if points all have the same weight. The gradient of this cost with respect to $\mathbf{t}$ is

$$-2\sum_i w_i\left(\mathbf{x}_i - \mathcal{M}\mathbf{y}_i - \mathbf{t}\right)$$

which vanishes at the solution, so that

$$\mathbf{t} = \frac{\sum_i w_i\mathbf{x}_i - \mathcal{M}\sum w_i\mathbf{y}_i}{\sum_i w_i}.$$

Now if $\sum_i w_i\mathbf{x}_i = \sum_i w_i\mathcal{M}\mathbf{y}_i = \mathcal{M}(\sum_i w_i\mathbf{y}_i)$, then $\mathbf{t} = \mathbf{0}$. An easy way to achieve $\mathbf{t} = \mathbf{0}$ is to ensure $\sum_i w_i\mathbf{x}_i = 0$ and $\sum_i w_i\mathbf{y}_i = 0$. Write

$$\mathbf{c}_x = \frac{\sum_i w_i\mathbf{x}_i}{\sum_i w_i}$$

for the center of gravity of the observations (etc.) Now form

$$\mathbf{u}_i = \mathbf{x}_i - \mathbf{c}_x \text{ and } \mathbf{v}_i = \mathbf{y}_i - \mathbf{c}_y$$

and if you use $\mathcal{U}$ and $\mathcal{V}$, then the translation will be zero and must only estimate $\mathcal{M}$. Further, the estimate $\hat{\mathcal{M}}$ of this matrix yields that the translation from the original reference points to the original observations is $\mathbf{c}_x - \hat{\mathcal{M}}\mathbf{c}_y$.

Finding $\mathcal{M}$ now reduces to minimizing

$$\sum_i w_i \left(\mathbf{u}_i - \mathcal{M}\mathbf{v}_i\right)^T \left(\mathbf{u}_i - \mathcal{M}\mathbf{v}_i\right)$$

as a function of $\mathcal{M}$. The natural procedure – take a derivative and set to zero, and obtain a linear system (**exercises**) – works fine, but it is helpful to apply some compact and decorative notation.

Write $\mathcal{W} = \mathsf{diag}\left([w_1, \ldots, w_N]\right)$, $\mathcal{U} = \left[\mathbf{u}_1^T, \ldots, \mathbf{u}_N^T\right]$ (and so on). Recall all vectors are column vectors, so $\mathcal{U}$ is $N \times d$. You should check that the objective can be rewritten as

$$\mathsf{Tr}\left(\mathcal{W}(\mathcal{U} - \mathcal{V}\mathcal{M}^T)(\mathcal{U} - \mathcal{V}\mathcal{M}^T)^T\right).$$

exercises Now the trace is linear; $\mathcal{U}^T\mathcal{W}\mathcal{U}$ is constant;

$$\mathsf{Tr}\left(\mathcal{A}\right) = \mathsf{Tr}\left(\mathcal{A}^T\right);$$

and

$$\mathsf{Tr}\left(\mathcal{A}\mathcal{B}\mathcal{C}\right) = \mathsf{Tr}\left(\mathcal{B}\mathcal{C}\mathcal{A}\right) = \mathsf{Tr}\left(\mathcal{C}\mathcal{A}\mathcal{B}\right)$$

(check this by writing it out, and remember it; it's occasionally quite useful). This means the cost is equivalent to

$$\mathsf{Tr}\left(-2\mathcal{U}^T\mathcal{W}\mathcal{V}\mathcal{M}^T\right) + \mathsf{Tr}\left(\mathcal{M}\mathcal{V}^T\mathcal{W}\mathcal{V}\mathcal{M}^T\right)$$

which will be minimized when

$$\mathcal{M}\mathcal{V}^T\mathcal{W}\mathcal{V} = \mathcal{U}^T\mathcal{W}\mathcal{V}$$

(which you should check **exercises**). The exercises establish cases where $\mathcal{V}^T\mathcal{W}\mathcal{V}^T$ will have full rank, and in these – the usual – cases $\mathcal{M}$ is easily obtained **exercises** . Notice this derivation works *whatever* the dimension of the points.

Procedure: 14.1 *Weighted Least Squares for Affine Transformations*

You have N correspondences $(\mathbf{x}_i, \mathbf{y}_i)$ each with a weight w_i and wish to find an affine transformation ($\mathcal{M}$, translation $\mathbf{t}$). by minimizing

$$\sum_i w_i(\mathbf{x}_i - \mathcal{M}\mathbf{y}_i - \mathbf{t})^T(\mathbf{x}_i - \mathcal{M}\mathbf{y}_i - \mathbf{t})$$

Write

$$\begin{aligned}\mathbf{c}_x &= \frac{\sum_i w_i\mathbf{x}_i}{\sum_i w_i} \\ \mathbf{c}_y &= \frac{\sum_i w_i\mathbf{y}_i}{\sum_i w_i} \\ \mathbf{u}_i &= \mathbf{x}_i - \mathbf{c}_x \\ \mathbf{v}_i &= \mathbf{y}_i - \mathbf{c}_y\end{aligned}$$

Write $\mathcal{U} = \left[\mathbf{u}_1^T, \mathbf{u}_2^T, \ldots, \mathbf{u}_N^T\right]$ (etc) and $\mathcal{W} = \text{diag}(w_1, \ldots, w_N)$. The least squares estimate $\hat{\mathcal{M}}$ satisfies the linear system

$$\hat{\mathcal{M}}\mathcal{V}^T\mathcal{W}\mathcal{V} = \mathcal{U}^T\mathcal{W}\mathcal{V}$$

and the least squares estimate $\hat{\mathbf{t}}$ of $\mathbf{t}$ is

$$\hat{\mathbf{t}} = \mathbf{c}_x - \hat{\mathcal{M}}\mathbf{c}_y$$

14.2.2 Euclidean Motion

One encounters affine transformations relatively seldom in practice, though they do occur. Much more interesting is the case where the transformation is Euclidean. The least squares solution above isn't good enough, because the $\mathcal{M}$ obtained that way won't be a rotation matrix. But a neat trick yields a least squares solution for a rotation matrix.

As in the previous section, subtract the centers of gravity to get the translation, and work with $\mathbf{u}_i$ and $\mathbf{v}_i$. The problem is now to choose $\mathcal{R}$ to minimize

$$\sum_i w_i(\mathbf{u}_i - \mathcal{R}\mathbf{v}_i)^T(\mathbf{u}_i - \mathcal{R}\mathbf{v}_i).$$

This can be done in closed form (a fact you should memorize, because it is extremely

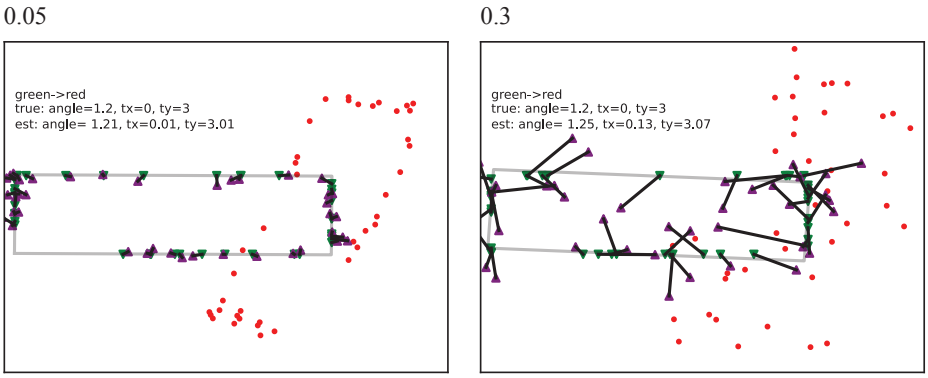

FIGURE 14.1: *Least squares registration is quite well-behaved under even quite pronounced gaussian noise. In each figure, the 40* **green** *(downward pointing) triangles, which lie on a rectangle one unit high and three units wide are subject to a Euclidean transformation, then noise is added, to obtain the* **red** *circles. I then used least squares to estimate a Euclidean transformation using corresponding green and red points, and applied this transformation to register the red points to the green, yielding the* **purple** *(upward pointing) triangles. I have joined each registered point to the original with a dark line. The thin dark rectangle shows the result of the estimated transformation applied to the true rectangle underlying the red points. The green triangles lie on it* **i***f the transformation is correctly estimated.* **Left:** *the noise is isotropic Gaussian noise, with standard deviation 0.05 (so 1/20 of the rectangle height);* **right**, *the standard deviation is 0.3 (or about 1/3 of the rectangle height). In each case, the parameters estimated by least squares are close to the transformation actually applied.*

useful). The objective function can be transformed to

$$\begin{aligned}\sum_i w_i(\mathbf{u}_i - \mathcal{R}\mathbf{v}_i)^T(\mathbf{u}_i - \mathcal{R}\mathbf{v}_i) &= \mathsf{Tr}\left(\mathcal{W}(\mathcal{U} - \mathcal{V}\mathcal{R}^T)(\mathcal{U} - \mathcal{V}\mathcal{R})^T\right) \\ &= \mathsf{Tr}\left(-2\mathcal{V}^T\mathcal{W}\mathcal{U}\mathcal{R}\right) + K \\ &\quad \text{(because } \mathcal{R}^T\mathcal{R} = \mathcal{I}\text{)}\end{aligned}$$

Here K is a constant that doesn't involve $\mathcal{R}$ and so is of no interest. Now compute an SVD of $\mathcal{V}^T\mathcal{W}\mathcal{U}$ to obtain $\mathcal{V}^T\mathcal{W}\mathcal{U} = \mathcal{A}\Sigma\mathcal{B}^T$ where $\mathcal{A}$, $\mathcal{B}$ are orthonormal, and $\mathcal{S}$ is diagonal (Section 41.2 if you're not sure). Now $\mathcal{B}^T\mathcal{R}\mathcal{A}$ is orthonormal, and we must maximize $\mathsf{Tr}\left(\mathcal{B}^T\mathcal{R}\mathcal{A}\mathcal{S}\right)$, meaning $\mathcal{B}^T\mathcal{R}\mathcal{A} = \mathcal{I}$ (check this if you're not certain), and so $\mathcal{R} = \mathcal{B}\mathcal{A}^T$.

Procedure: 14.2 *Weighted Least Squares for Euclidean Transformations*

We have N reference points $\mathbf{x}_i$ whose location is measured in the agent's coordinate system. Each corresponds to a point in the world coordinate system with known coordinates $\mathbf{y}_i$, and the change of coordinates is a Euclidean transformation (rotation $\mathcal{R}$, translation $\mathbf{t}$). For each $(\mathbf{x}_i, \mathbf{y}_i)$ pair, we have a weight w_i. We wish to minimize

$$\sum_i w_i(\mathbf{x}_i - \mathcal{R}\mathbf{y}_i - \mathbf{t})^T(\mathbf{x}_i - \mathcal{R}\mathbf{y}_i - \mathbf{t}) \tag{14.1}$$

Write

$$\begin{aligned}
\mathbf{c}_x &= \frac{\sum_i w_i \mathbf{x}_i}{\sum_i w_i} \\
\mathbf{c}_y &= \frac{\sum_i w_i \mathbf{y}_i}{\sum_i w_i} \\
\mathbf{u}_i &= \mathbf{x}_i - \mathbf{c}_x \\
\mathbf{v}_i &= \mathbf{y}_i - \mathbf{c}_y
\end{aligned}$$

Write $\mathcal{U} = \left[\mathbf{u}_1^T, \mathbf{u}_2^T, \ldots, \mathbf{u}_N^T\right]$ (etc); $\mathcal{W} = \text{diag}(w_1, \ldots, w_N)$; and $\text{SVD}(\mathcal{U}^T\mathcal{W}\mathcal{V}) = \left[\mathcal{A}, \Sigma, \mathcal{B}^T\right]$. The least squares estimate $\hat{\mathcal{R}}$ is

$$\hat{\mathcal{R}} = \mathcal{B}\mathcal{A}^T$$

and the least squares estimate $\hat{\mathbf{t}}$ of $\mathbf{t}$ is

$$\hat{\mathbf{t}} = \mathbf{c}_x - \hat{\mathcal{R}}\mathbf{c}_y$$

14.2.3 Projective Transformations

Recall from Section 3.2 that a projective transformation of an image is given by a 3×3 matrix $\mathcal{M}$ that has full rank. The transformation can be written

$$\begin{bmatrix} x_1 \\ \\ x_2 \end{bmatrix} = \begin{bmatrix} \frac{m_{11}y_1 + m_{12}y_2 + m_{13}}{m_{31}y_1 + m_{32}y_2 + m_{33}} \\ \\ \frac{m_{21}y_1 + m_{22}y_2 + m_{23}}{m_{31}y_1 + m_{32}y_y + m_{33}} \end{bmatrix}.$$

Higher dimensions follow the pattern. A projective transformation in d dimensions is given by a $d+1 \times d+1$ matrix $\mathcal{M}$ that has full rank. The transformation is now

$$\begin{bmatrix} x_1 \\ \dots \\ x_d \end{bmatrix} = \begin{bmatrix} \frac{m_{11}y_1 + \ldots + m_{1d}y_d + m_{1(d+1)}}{m_{(d+1)1}y_1 + \ldots + m_{(d+1)d}y_d + m_{(d+1)(d+1)}} \\ \dots \\ \frac{m_{d1}y_1 + \ldots + m_{dd}y_d + m_{d(d+1)}}{m_{(d+1)1}y_1 + \ldots + m_{(d+1)d}y_d + m_{(d+1)(d+1)}} \end{bmatrix}.$$

and the residual error between $\mathbf{x}_i$ and $\mathcal{M}(\mathbf{y}_i)$ is

$$\mathbf{r}_i = \mathbf{x}_i - \begin{bmatrix} \frac{m_{11}y_1 + \ldots + m_{1d}y_d + m_{1(d+1)}}{m_{(d+1)1}y_1 + \ldots + m_{(d+1)d}y_d + m_{(d+1)(d+1)}} \\ \dots \\ \frac{m_{d1}y_1 + \ldots + m_{dd}y_d + m_{d(d+1)}}{m_{(d+1)1}y_1 + \ldots + m_{(d+1)d}y_d + m_{(d+1)(d+1)}} \end{bmatrix}.$$

A weighted least squares solution now solves

$$\sum_i w_i \mathbf{r}_i^T \mathbf{r}_i.$$

There isn't a clean form for the solution, and numerical minimization is required. You should use a second order method (Levenberg-Marquardt is favored; Chapter 41.2). Experience teaches that this optimization is not well behaved without a strong start point.

There is an easy construction for a good start point. For a pair of known points $\mathbf{x}$ and $\mathbf{y}$, you can cross multiply the equations for the projective transformation to get

$$\begin{bmatrix} 0 \\ \dots \\ 0 \end{bmatrix} = \begin{bmatrix} \left(m_{11}y_1 + \ldots + m_{1d}y_d + m_{1(d+1)}\right) - \\ x_1\left(m_{(d+1)1}y_1 + \ldots + m_{(d+1)d}y_d + m_{(d+1)(d+1)}\right) \\ \dots \\ \left(m_{d1}y_1 + \ldots + m_{dd}y_d + m_{d(d+1)}\right) - \\ x_d\left(m_{(d+1)1}y_1 + \ldots + m_{(d+1)d}y_d + m_{(d+1)(d+1)}\right) \end{bmatrix}.$$

Here the m_{ij} are unknown, so this is a set of d homogenous linear equations in $(d+1) \times (d+1)$ unknowns. In turn, if you have at least $d+1$ different $(\mathbf{x}, \mathbf{y})$ pairs that meet conditions **exercises** , you can solve the system up to scale. But the scale of the solution does not affect the transformation it implements, so you have a start point. The resulting estimate of $\mathcal{M}$ has a good reputation as a start point for a full optimization. Notice this construction does not take weights into account. If the weights come from IRLS, then you need this construction only at the start. For every other iteration, the previous iteration will supply an acceptable start point as well as weights.

Procedure: 14.3 *Estimating a Projective Transformation from Data*

Given N known source points $\mathbf{y}_i = (y_{i,1}, \ldots, y_{i,d})$ in affine coordinates and N corresponding target points $\mathbf{x}_i$ with measured locations $(x_{i,1}, \ldots, x_{i,d})$ and weights w_i, obtain the projective transformation $\mathcal{M}$ with i, j'th element m_{ij} mapping source to target by minimizing:

$$\sum_i w_i \xi_i^T \xi_i \tag{14.2}$$

where

$$\xi_i = \begin{bmatrix} x_{i,1} - \frac{m_{11}y_{i,1} + \ldots + m_{1d}y_{i,d} + m_{1(d+1)}}{m_{(d+1)1}x_{i,1} + \ldots + m_{(d+1)d}x_{i,d} + m_{(d+1)(d+1)}} \\ \cdots \\ x_{i,d} - \frac{m_{d1}y_{i,1} + \ldots + m_{dd}y_{i,d} + m_{d(d+1)}}{m_{(d+1)1}x_{i,1} + \ldots + m_{(d+1)d}x_{i,d} + m_{(d+1)(d+1)}} \end{bmatrix} \tag{14.3}$$

Obtain a start point by as a least squares solution to the set of homogeneous linear equations

$$\begin{aligned} 0 &= x_{i,1}(m_{(d+1)1}y_{i,1} + \ldots + m_{(d+1)d}y_{i,d} + m_{(d+1)(d+1)}) - \\ &\quad m_{11}y_{i,1} + \ldots + m_{1d}y_{i,d} + m_{1(d+1)} \\ &\cdots \\ 0 &= x_{i,d}(m_{(d+1)1}y_{i,1} + \ldots + m_{(d+1)d}y_{i,d} + m_{(d+1)(d+1)}) - \\ &\quad m_{(d+1)1}x_{i,1} + \ldots + m_{(d+1)d}x_{i,d} + m_{(d+1)(d+1)} \end{aligned}$$

14.2.4 Probabilistic Interpretations and Variants

If the weights are uniform, then solving

$$\sum_i w_i \mathbf{r}_i^T \mathbf{r}_i.$$

is equivalent to assuming that the error in point estimates is isotropic normal, and maximizing the likelihood of the error. This equivalence is sometimes helpful. If you know, for example, that errors in some directions are more likely than errors in others, it can be a good idea to use an estimate of the covariance between errors Σ, so the criterion becomes

$$\sum_i w_i \mathbf{r}_i^T \Sigma^{-1} \mathbf{r}_i.$$

The modification to each procedure is straightforward (**exercises**).

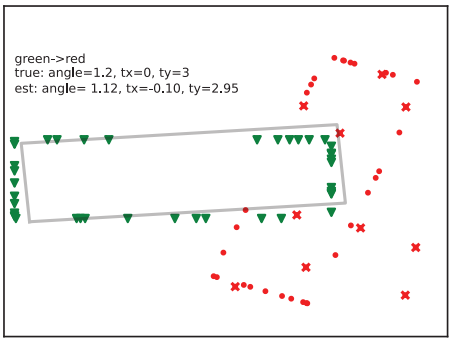

FIGURE 14.2: *Significant registration errors can be caused by small numbers of outliers. This figure uses the same markers as 14.1. In this case, rather than adding gaussian noise to the red points, I have replaced five of them with points drawn uniformly and at random from a box surrounding the red points. The outliers are marked with a* **red x**. *All others are in their transformed location and have not had noise added. The estimated transformation has been significantly affected – note how the fine dark rectangle doesn't pass through the green triangles.*

14.3 ROBUSTNESS, IRLS AND RANSAC

Correspondences are $(\mathbf{x}, \mathbf{y})$ pairs. Good correspondences are ones where $\mathcal{T}(\mathbf{y})$ is close to $\mathbf{x}$ for the true transformation $\mathcal{T}$. Errors are ones where $\mathcal{T}(\mathbf{y})$ is far from $\mathbf{x}$ for the true transformation $\mathcal{T}$. In the case of image registration, some correspondences are likely to be wrong, but you should expect a relatively large fraction of good correspondences. This is a robustness issue (Figure 14.2), which IRLS or RANSAC can deal with as long as there are not too many bad correspondences.

14.3.1 IRLS for Registration

The IRLS recipe can be applied with very little modification to registration. Choose a robust cost function from Section 12.2.1 or elsewhere. Recall this cost applies to the residual. Write θ for the parameters of the transformation $\mathcal{T}_\theta$, and the residual is now

$$r(\mathbf{x}_i, \mathbf{y}_i, \theta) = \sqrt{(\mathbf{x}_i - \mathcal{T}_\theta(\mathbf{y}_i))^T (\mathbf{x}_i - \mathcal{T}_\theta(\mathbf{y}_i))}.$$

The square root ensures that minimizing the least squares criterion is equivalent to

$$(1/2) \sum_i (r(\mathbf{x}_i, \mathbf{y}_i, \theta))^2.$$

For any given θ, the weights are now

$$w_i = \left(\frac{\frac{\partial \rho}{\partial u}}{r(\mathbf{x}_i, \mathbf{y}_i, \theta)} \right).$$

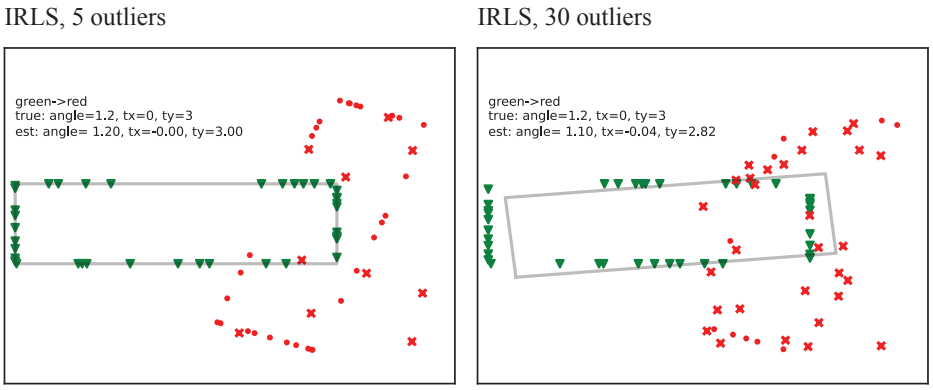

FIGURE 14.3: *IRLS is effective at controlling registration errors caused by small numbers of outliers, but can be overwhelmed by large numbers of outliers. This figure uses the same markers as 14.1. As in Figure 14.2, I have replaced some red points with points drawn uniformly and at random from a box surrounding the red points, marked with a red x. I estimated the transformation with IRLS. On the* **left**, *with a moderate fraction of outliers (5 in 40 points), the transformation estimate is very good; on the* **right**, *where most points are outliers (30 in 40 points), the estimate is much weaker.*

As Figure 14.4 shows, IRLS does very well for moderate numbers of outliers, but performance is degraded when there are too many. The procedure is important, so I have put it in a box.

Procedure: 14.4 *Fitting a Transformation using Iteratively Reweighted Least Squares*

This procedure takes a set of N putatively corresponding point pairs $(\mathbf{x}_i, \mathbf{y}_i)$ and obtains an affine, Euclidean or projective transformation $\mathcal{T}_\theta$ that registers the pairs while discounting the effect of some correspondence errors. Choose a robust cost function $\rho(u;\sigma)$ from Section 12.2.1 or somewhere else.

Initialize with an initial set of parameters $\theta^{(1)}$. One strategy is to choose a small subset of S correspondences at random, then fit a transformation with weights $1/S$ using the appropriate weighted least squares procedure. Compute

$$r_i^{(1)} = r(\mathbf{x}_i, \mathbf{y}_i, \theta^{(1)}) = \sqrt{(\mathbf{x}_i - \mathcal{T}_\theta(\mathbf{y}_i))^T(\mathbf{x}_i - \mathcal{T}_\theta(\mathbf{y}_i))}$$

for each correspondence. Obtain an initial scale $\sigma^{(1)}$ using either application considerations or

$$\sigma^{(1)} = 1.4826 \text{ median}_i \, |r_i^{(1)}| \, .$$

Compute

$$w_i^{(1)} = \frac{\frac{d\rho}{du}}{r_i^{(1)}}$$

where the derivative is evaluated at $u = r_i^{(1)}$ and $\sigma^{(1)}$.

Now use iterate three steps:

> **Estimate the transformation** using the appropriate weighted least squares procedure to obtain the new set of parameters $\theta^{(r+1)}$ and $r_i^{(r+1)}$ from $w_i^{(r)}$.
>
> **Estimate the scale**, possibly using
>
> $$\sigma^{(r+1)} = 1.4826 \text{ median}_i \, |r_i^{(r+1)}| \, .$$
>
> Alternatively, use a fixed scale obtained using application considerations.
>
> **Re-estimate weights** using
>
> $$w_i^{(r+1)} = \frac{\frac{d\rho}{du}}{r_i^{(i)}}$$
>
> where the derivative is evaluated at $u = r_i^{(r+1)}$ and $\sigma^{(r+1)}$.

Terminate iterations when either the change in the transformation is below a threshold or there have been too many.

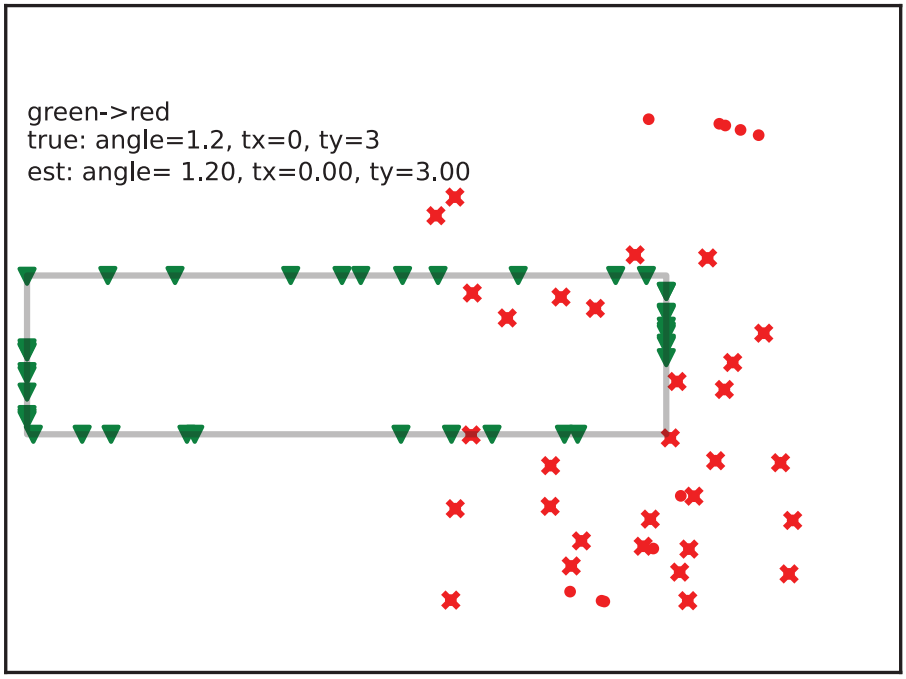

FIGURE 14.4: *RANSAC can be effective at controlling registration errors caused by outliers. This figure uses the same markers as 14.1. As in Figure 14.2, I have replaced some red points with points drawn uniformly and at random from a box surrounding the red points, marked with a red x. I estimated the transformation with RANSAC. Here most points are outliers (30 in 40 points) (compare Figure 14.4) and the estimate is very good.*

14.3.2 RANSAC

Adapting RANSAC to registration problems is mostly straightforward when there are relatively few outliers. A line is completely specified by two points (which is why Procedure 12.4 used two random samples). Different transformations require different numbers of correspondences, however. An affine transformation in d dimensions is exactly specified by $d+1$ correspondences (**exercises**) and a projective transformation in d dimensions is exactly specified by $d+2$ correspondences. Euclidean transformations are more tricky. For example, in the plane, one correspondence is not enough to specify a Euclidean transformation (you can rotate about a point) and there are many sets of two correspondences that can't be registered exactly with a Euclidean transformation (it doesn't change lengths). Use two correspondences for plane Euclidean transformations, and three for 3D Euclidean transformations.

Procedure: 14.5 *Registration Using RANSAC*

This procedure takes a set of N putative correspondences $(\mathbf{x}_i, \mathbf{y}_i)$ and obtains an estimate of the registration.
Start by choosing: the number of correspondences required to determine a transformation, n; the number of iterations required, k; the threshold used to identify a correspondence that is good, t; the number of good correspondences required to assert a model fits well, d. Set up a collection of good fits, currently empty.
Iterate until k iterations have occurred:

> Draw a sample of n distinct correspondences from the data uniformly and at random, and determine the transformation implied by those correspondences. If the transformation is acceptable:
>
> > For each correspondence outside the sample, if the length of the residual is less than t, the correspondence is good.
> >
> > If there are d or more good correspondences then there is a good fit. Refit the transformation using all these correspondences and a robust loss (likely using IRLS). Add the result to a collection of good fits.

Use the best fit from the collection of good fits, using the fitting error as a criterion.
Choosing n: Use: $d + 1$ correspondences for an affine transformation in d dimensions; $d + 2$ for a projective transformation in d dimensions; two correspondences for plane Euclidean transformations; and three for 3D Euclidean transformations.
Determining the transformation: Use the relevant weighted least squares procedure, with $w_i = 1/n$. For affine transformations and Euclidean transformations, check the eigenvalues of $\mathcal{V}\mathcal{W}\mathcal{V}^T$; if the smallest eigenvalue is too small, the solution will not be acceptable because of an accidental alignment between correspondences. For Euclidean transformations, check the eigenvalues of $\mathcal{U}^T\mathcal{W}\mathcal{V}$; if the smallest eigenvalue is too small, the solution will not be acceptable because of an accidental alignment between correspondences. For projective transformations, either check the eigenvalues of the hessian of the objective at the solution for a small eigenvalue (best test), or check the eigenvalues of $\hat{\mathcal{M}}$ for a large value (easiest); either is an indicator of an unacceptable solution

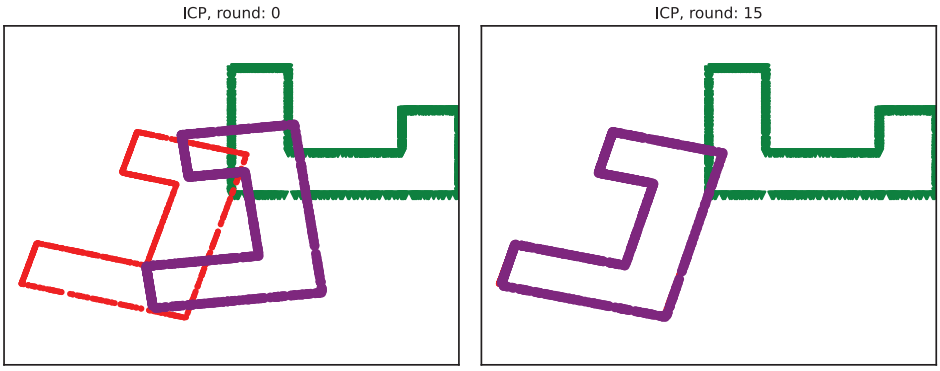

FIGURE 14.5: *ICP can converge quickly to the right transformation. The* **green** *(upward pointing, to the right) u shape must be transformed to lie on the* **red** *(sideways pointing, to the left) u shape. The running shape is* **purple**. **Left** *shows the initial transformation.* **Right** *has been registered by 15 iterations of ICP – you can see only two u-shapes, because the running points are now precisely registered to the target points.*

14.4 UNKNOWN CORRESPONDENCE

In cases like that of registering LIDAR point clouds to one another, or meshes to LIDAR point clouds, there isn't much – or, often, any – information at each point that you can use to match. Just forming $\mathcal{X} \times \mathcal{Y}$ – taking every pair of points, one from $\mathcal{X}$ and one from $\mathcal{Y}$ – and hoping that either IRLS will be able to tell good from bad correspondences is very likely to fail. IRLS fails because there are far too many bad correspondences and far too many local minima; you are highly unlikely to be lucky enough to find a good solution.

Relying on RANSAC to determine correspondences is unwise. This *doesn't* contradict the previous section: there, one of the points in a correspondence was replaced with an outlier, but the fraction of correspondences that were so affected was relatively small (0.75 in one example). RANSAC can require very large numbers of samples when the fraction of outliers is high. Say $\mathcal{X}$ has N points and $\mathcal{Y}$ has M points, you want to compute a Euclidean transformation, and the only thing you know about the correspondences is that they are one to one. Then at most $\min(N, M)$ correspondences can be good. This means that *in the best case* you will need to look at of the order of

$$\frac{1}{[\max(M, N)]^3}$$

samples to see one set of three good samples. If there are fewer good correspondences, the number of samples required will get worse. Anything you can do to reduce the number of bad correspondences would be helpful.

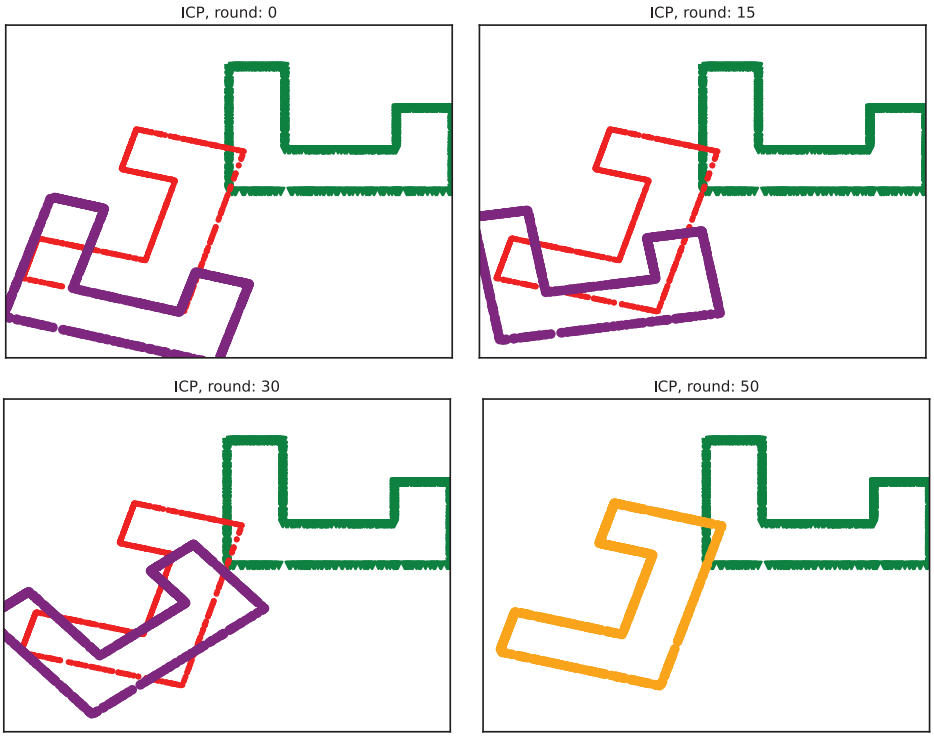

FIGURE 14.6: *Some initial transformations can result in slow convergence of ICP. The* **green** *(upward pointing, to the right) u shape must be transformed to lie on the* **red** *(sideways pointing, to the left) u shape. The running shape is* **purple**. **Top left** *shows the initial transformation;* **top right**, *the result after 15 iterations of ICP;* **bottom left**, *after 30 iterations; and* **bottom right** *after 50 iterations. In the last figure, you can see only two u-shapes, because the running points are now precisely registered to the target points.*

14.4.1 Iterated Closest Points or ICP

There is an alternative strategy that applies *if* you have a reasonable estimate of the initial transformation. We have N reference points $\mathbf{y}_i$ and M observed points $\mathbf{x}_i$. For the moment, we will assume that all weights w_i are 1. A straightforward, and very effective, recipe for registering the points is *iterative closest points* or *ICP*. The key insight here is that, if the transformation is very close to the identity, then the $\mathbf{y}_{c(i)}$ that corresponds to $\mathbf{x}_i$ should be the closest reference point to $\mathbf{x}_i$. This finding the closest reference point to each measurement and computing the transformation using that correspondence. But the transformation might not be close to the identity, and so the correspondences might change. We could repeat the process until they stop changing.

Formally, start with a transformation estimate $\mathcal{T}_1$, a set of $\mathbf{m}_i^{(1)} = \mathcal{T}^{(1)}(\mathbf{y}_i)$ (the *running points*) and then repeat three steps:

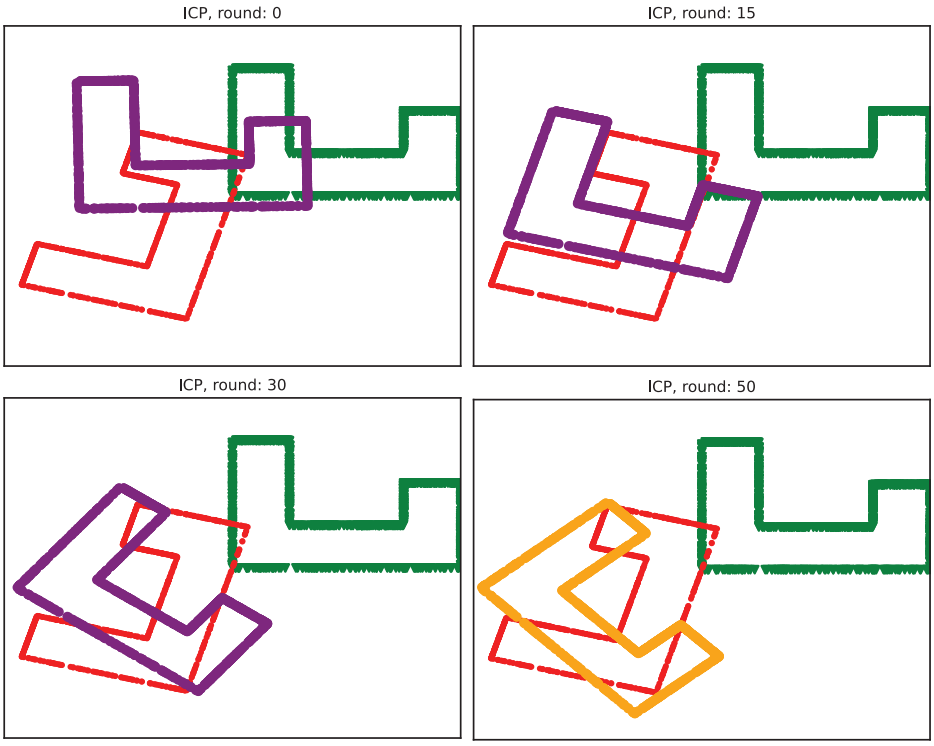

FIGURE 14.7: *ICP can converge to the wrong answer, typically when the initial transformation is very different from the right answer. The* **green** *(upward pointing, to the right) u shape must be transformed to lie on the* **red** *(sideways pointing, to the left) u shape. The running shape is* **purple**. **Top left** *shows the initial transformation;* **top right**, *the result after 15 iterations of ICP;* **bottom left**, *after 30 iterations; and* **bottom right** *after 50 iterations. You can still see three u-shapes, because the running points are incorrectly registered to the target points.*

- **Estimate correspondences** using the transformation estimate. Then, for each $\mathbf{x}_i$, we find the closest $\mathbf{m}^{(n)}$ (say $\mathbf{m}_c^{(n)}$); then $\mathbf{x}_i$ corresponds to $\mathbf{m}_{c(i)}^{(n)}$.
- **Estimate a transformation** $\mathcal{T}^{(n+1)}$ using the corresponding pairs.
- **Update the running points** by mapping $\mathbf{m}_i^{(n)}$ to $\mathcal{T}^{(n+1)}(\mathbf{m}_i^{(n)})$ and

These steps are repeated until convergence, which can be tested by checking if the correspondences don't change or if $\mathcal{T}^{(n+1)}$ is very similar to the identity. The required transformation is then

$$\mathcal{T}^{(n+1)} \circ \mathcal{T}^{(n)} \circ \ldots \mathcal{T}^{(1)} \tag{14.4}$$

There are a number of ways in which this very useful and very general recipe can be adapted. First, if there is any description of the points available, it can be

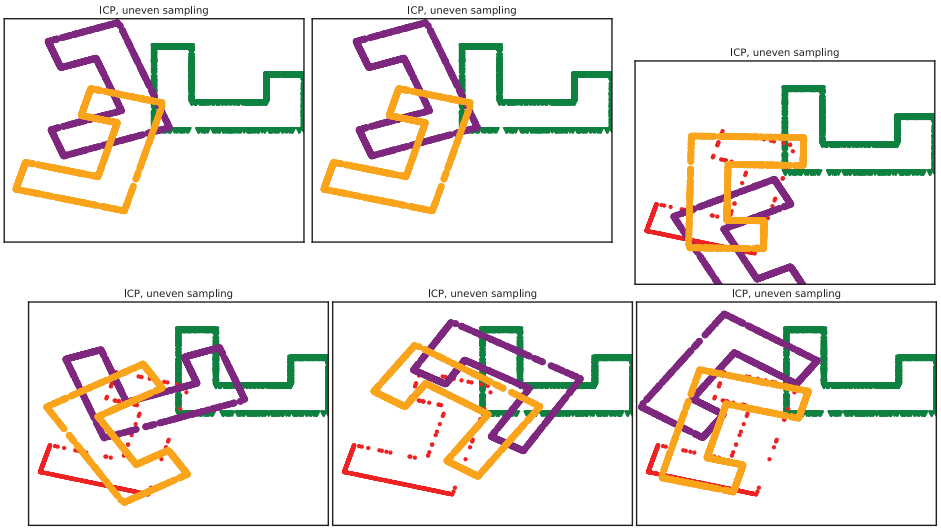

FIGURE 14.8: *If the two shapes that have been sampled differently, ICP can produce poor results. The* **green** *(upward pointing, to the right) u shape must be transformed to lie on the* **red** *(sideways pointing, to the left) u shape. The initial transformation is shown by the* **purple** *shape. The ICP result is shown in* **orange**. *The two offset cases are successful; the others are not.*

used to cut down on correspondences (so, for example, we match only red points to red points, green points to green points, and so on). Second, finding an exact nearest neighbor in a large point cloud is hard and slow, and we might need to subsample the point clouds or pass to approximate nearest neighbors (more details below). Third, points that are very far from the nearest neighbor might cause problems, and we might omit them (again, more details below).

14.4.2 ICP and Sampling

The ICP recipe becomes difficult to apply to point clouds when M or N are very large. One obvious strategy to control this problem applies when something else – say, a color measurement – is known about each point. For example, we might get such data by using a range camera aligned with a conventional camera, so that every point in the depth map comes with a color. When extra information is available, one searches only compatible pairs for correspondences. As another example, you might estimate a normal at each $\mathbf{m}_i^{(n)}$ and each $\mathbf{y}_i$ by fitting a plane to it and a few nearby neighbors (Section **??**). Now assuming that the running points are quite like the reference points, use only correspondences where the normals are nearly parallel. Test this by testing whether the dot-product between normals is large enough.

Large point clouds are fairly common in autonomous vehicle applications. For example, the measurements might be LIDAR measurements of some geometry. It

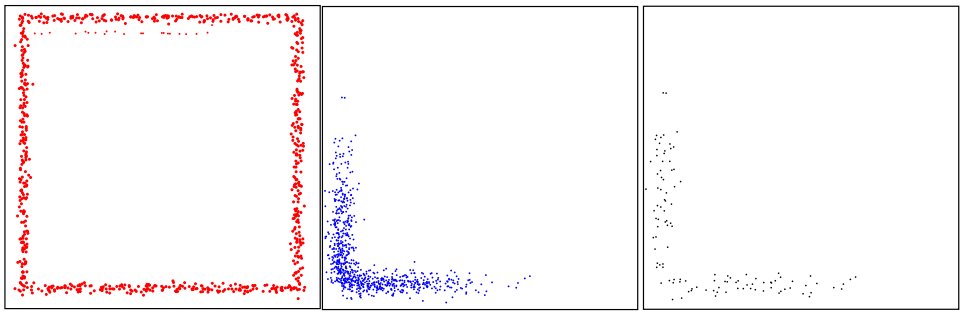

FIGURE 14.9: *On the* **left** *a map of a simple arena, represented as a point cloud. Such a map could be obtained by registering LIDAR measurements to one another. A LIDAR or depth sensor produces measurements in the sensor's coordinate system, and registering these measurements to the map will reveal where the sensor is. However, the sensor may measure points more densely at some positions than at others.* **Left** *shows such a measurement; note the heavy sampling of points near the corner and the light sampling on the edges. This can bias the registration, because the large number of points near the corner mean that the registration error consists mostly of errors from these points. It can also create significant computational problems, because finding the closest points will become slower as the number of points increases. A stratified sample of the measurements (***right***) is obtained by dividing the plane (in this case) into cells of equal area (usually a grid), then resampling the measurements at random* so there are no more than a fixed number of samples in each box. *Such a sample can both reduce bias and improve the speed of registration.* **TODO:** Source, Credit, Permission

is quite usual now to represent that geometry with another, perhaps enormous, point cloud, which you could think of as a map. Registration would then tell the vehicle where it was in the map. Notice that in this application, there is unlikely to be a measurement that exactly corresponds to each reference point. Instead, when the registration is correct, every $\mathbf{x}_i$ is very close to some transformed $\mathbf{y}_i$, so a least squares estimate is entirely justified. In cases like this, one can subsample the reference point cloud, the measurement point cloud, or both.

The sampling procedure depends on the application, and can have significant effects. For example, imagine you are working with LIDAR on a vehicle which is currently in an open space next to a wall (Figure 18.1). There will be many returns from the wall, and likely few from the open space. Uniformly sampled measurements would still have many returns from the wall, and few from the open space. This could bias the estimate of the vehicle's pose (Figure 14.8). A better alternative would be to build a *stratified sample* by breaking the space around the vehicle into blocks of fixed size, then choosing uniformly at random a fixed number of samples in each block. In this scheme, the wall would be undersampled, and the open space would be oversampled, somewhat resolving the bias.

Another stratified sampling strategy is to ensure that surface normal directions are evenly represented in the samples. Make an estimate of a surface normal

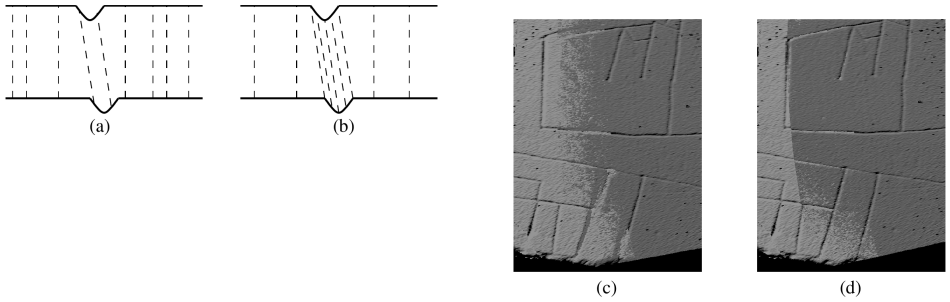

FIGURE 14.10: *The sample of points used in registration can be biased in useful ways. For example, (a) shows a cross section of a flat surface with a small groove (***above***) which needs to be registered to a similar surface (***below***). If point samples are drawn on the surface at random, then there will be few samples in the groove; the dashed lines indicate correspondences. In turn, the registration will be poor, because the surfaces can slide on one another. In (b), the samples have been drawn so that normal directions are evenly represented in the samples. Notice this means more samples concentrated in the groove, and fewer on the flat part. As a result, the surface is less free to slide, and the registration improves.*
TODO: what do c and d show? **TODO:** Source, Credit, Permission

at each point (for example, by fitting a plane to the point and some of its nearest neighbors). Now break the unit sphere, which encodes the surface normals, into even cells, and sample the points so that each cell has the same number of samples. This approach is particularly useful when we are trying to register flat surfaces with small relief details on them (Figure **??**).

14.4.3 Beyond ICP

ICP minimizes a cost function

$$\sum_i \left[\min_j \| \mathbf{x}_j - \mathcal{T}(\mathbf{y}_i) \|_2^2 \right] = \sum_i E_i(\mathbf{T}) \tag{14.5}$$

by finding the corresponding pairs (the $\mathbf{x}_j$ that corresponds to $\mathbf{y}_i$), then minimizing, then repeating. This is an easy way to exploit the closed form solution for $\mathcal{T}$ when correspondence is known, but it isn't the only way. The min means the objective function isn't differentiable everywhere (exercises), but it is continuous, and it is differentiable at most locations. This is usually a sign that straightforward optimization methods can be applied successfully, which is true here. The Levenberg-Marquardt algorithm (Section **??**) works particularly well here, because for a particular correspondence, the cost is a least squares cost, and because it doesn't require second derivatives. Notice that, to obtain the gradient of $E_i(\mathbf{T})$ with respect to $\mathbf{T}$, you need to know *which* $\mathbf{x}_j$ is closest to $\mathcal{T}(\mathbf{y}_i)$, so you still need to find the nearest neighbor.