CHAPTER 16

Registration, Correspondence and Outliers

The general point registration 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.

Section 15.3 dealt with the case where 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.

In most cases, you need to estimate correspondences from the data. Correspondences that are wrong tend to be badly wrong, creating a robustness issue. Section 16.1 shows how to deal with this using IRLS (slightly adapted from Section 14.2.2) and RANSAC (slightly adapted from Section 14.3). For some kinds of data, it is better to estimate correspondences from an estimate of the transformation (Section 16.2).

16.1 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 16.1), which IRLS or RANSAC can deal with as long as there are not too many bad correspondences.

16.1.1 IRLS for Registration

The IRLS recipe can be applied with very little modification to registration. Choose a robust cost function from Section 14.2.1 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

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

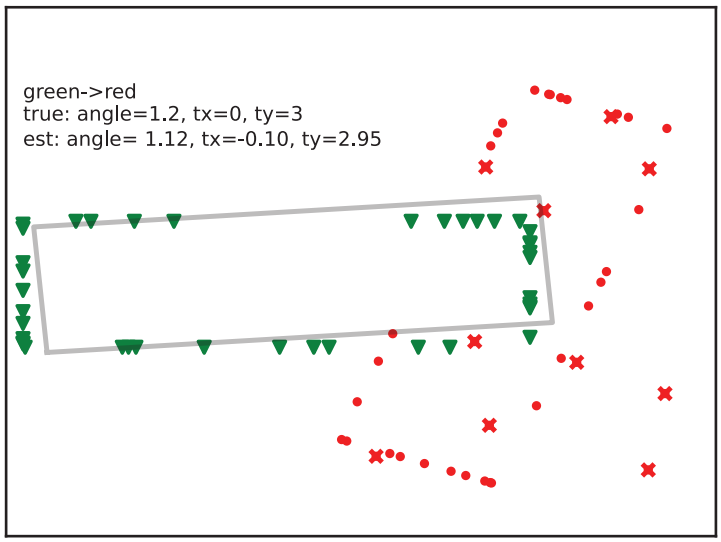

FIGURE 16.1: *Significant registration errors can be caused by small numbers of outliers. This figure uses the same markers as 15.7. In this case, rather than adding gaussian noise to the red points, I have replaced five of them with points drawn uniformly and at random from a box surrounding the red 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.*

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

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

As Figure 16.3 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.

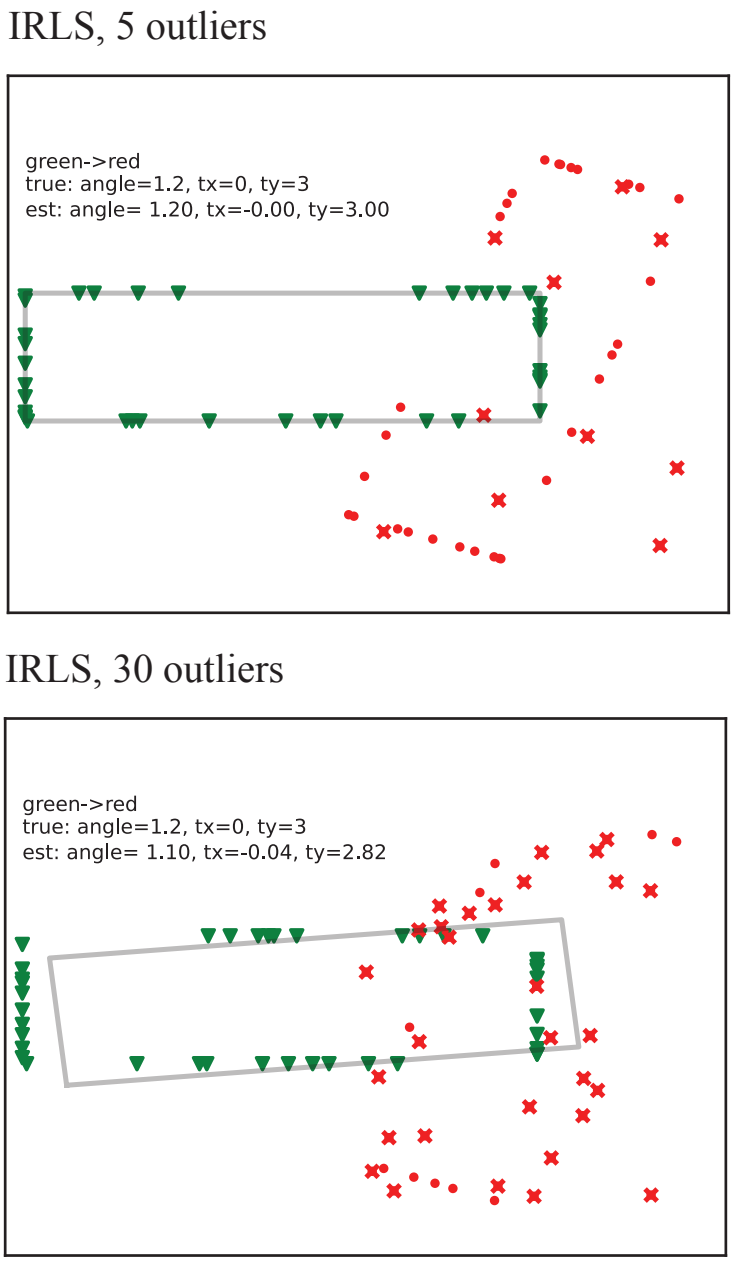

FIGURE 16.2: *IRLS is effective at controlling registration errors caused by small numbers of outliers, but can be overwhelmed by large numbers of outliers. This figure uses the same markers as 15.7. As in Figure 16.1, I have replaced some red points with points drawn uniformly and at random from a box 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.*

Procedure: 16.1 *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 14.2.1 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.

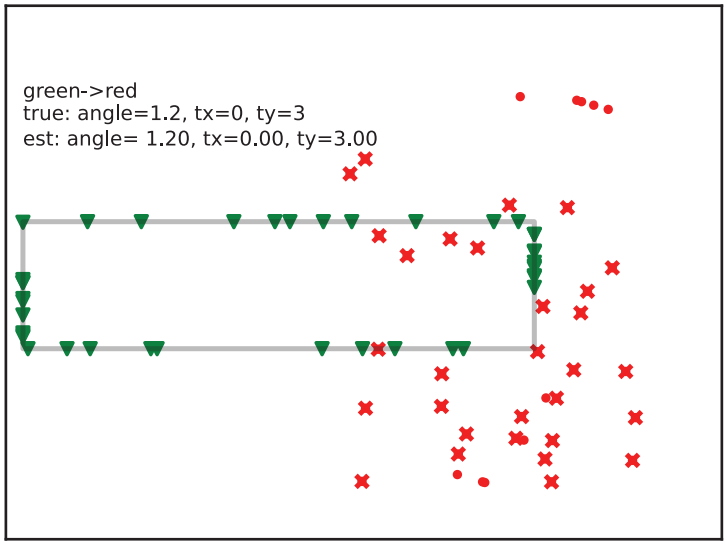

FIGURE 16.3: *RANSAC can be effective at controlling registration errors caused by outliers. This figure uses the same markers as 15.7. As in Figure 16.1, I have replaced some red points with points drawn uniformly and at random from a box surrounding the red points, marked with a red x. I estimated the transformation with RANSAC. Here most points are outliers (30 in 40 points) (compare Figure 16.3) and the estimate is very good.*

16.1.2 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 14.4 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: 16.2 *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

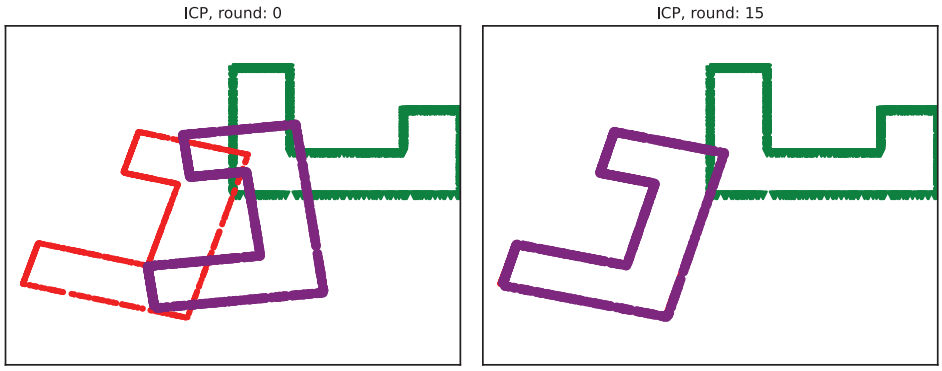

FIGURE 16.4: *ICP can converge quickly to the right transformation. The* **green** *(upward pointing, to the right) u shape must be transformed to lie on the* **red** *(sideways pointing, to the left) u shape. The running shape is* **purple**. **Left** *shows the initial transformation.* **Right** *has 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.*

16.2 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.

16.2.1 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).

16.2.2 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.

16.2.3 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:

- **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 $\mathbf{m}_{c(i)}^{(n)}$.

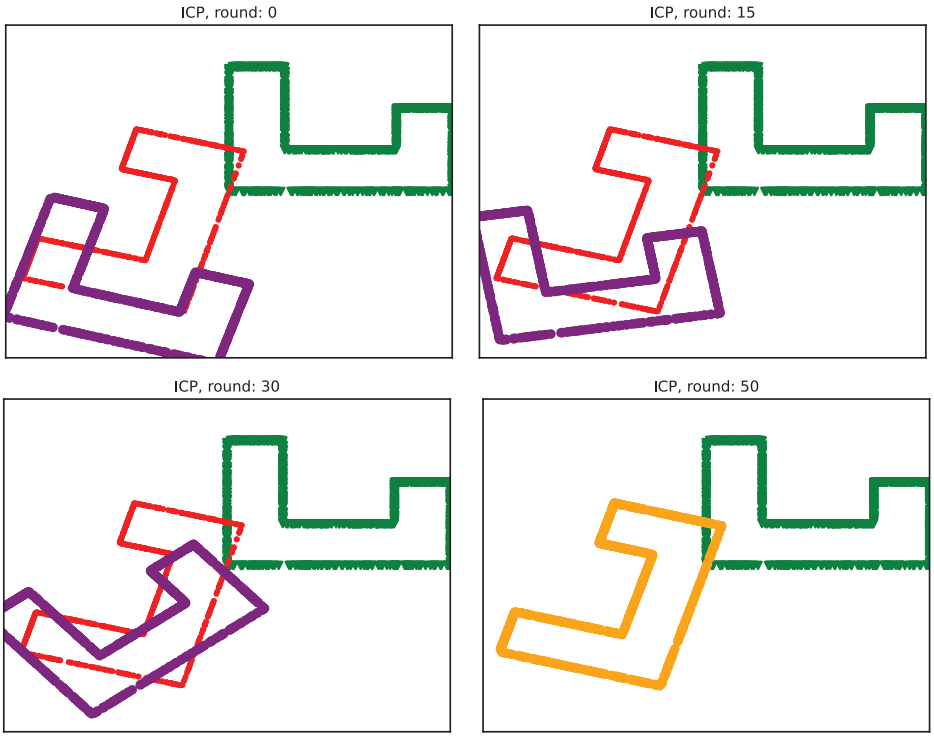

FIGURE 16.5: *Some initial transformations can result in slow convergence of ICP. The* **green** *(upward pointing, to the right) u shape must be transformed to lie on the* **red** *(sideways pointing, to the left) u shape. The running shape is* **purple**. **Top left** *shows the initial transformation;* **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.*

- **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

$$\mathcal{T}^{(n+1)} \circ \mathcal{T}^{(n)} \circ \ldots \mathcal{T}^{(1)} \tag{16.1}$$

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 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

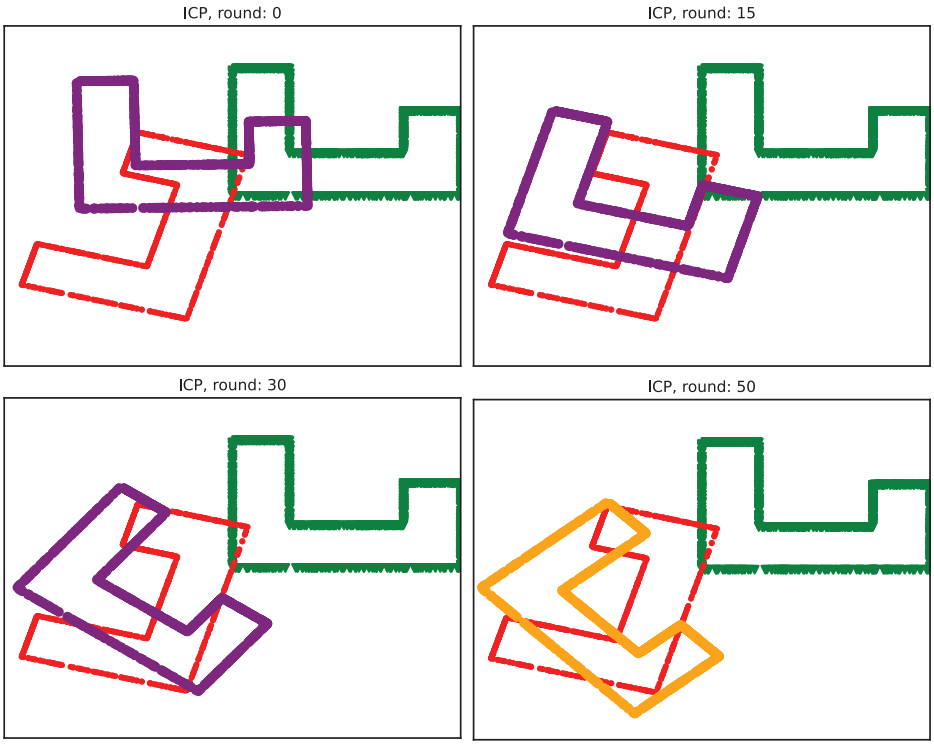

FIGURE 16.6: *ICP can converge to the wrong answer, typically when the initial transformation is very different from the right answer. The* **green** *(upward pointing, to the right) u shape must be transformed to lie on the* **red** *(sideways pointing, to the left) u shape. The running shape is* **purple**. **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.*

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).

16.2.4 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

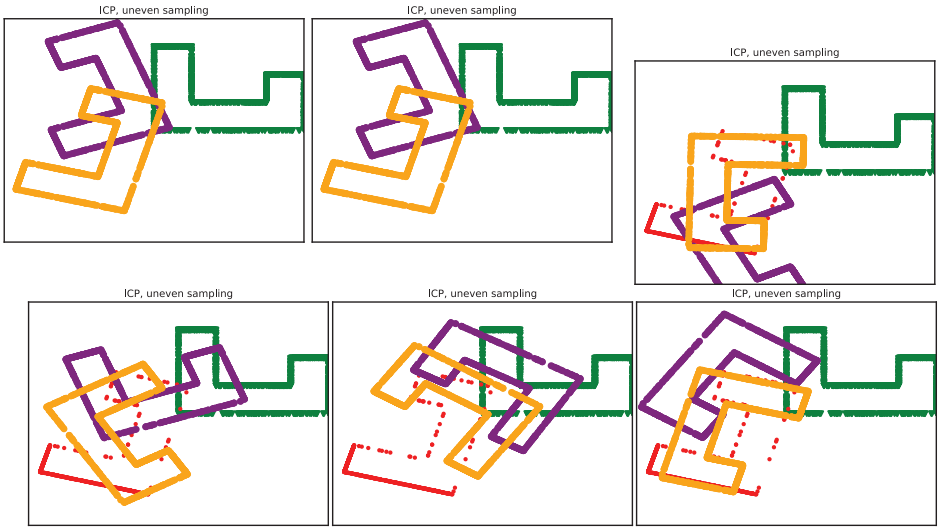

FIGURE 16.7: *If the two shapes that have been sampled differently, ICP can produce poor results. The* **green** *(upward pointing, to the right) u shape must be transformed to lie on the* **red** *(sideways pointing, to the left) u shape. The initial transformation is shown by the* **purple** *shape. The ICP result is shown in* **orange**. *The two offset cases are successful; the others are not.*

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 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 19.1). 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 16.7). 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

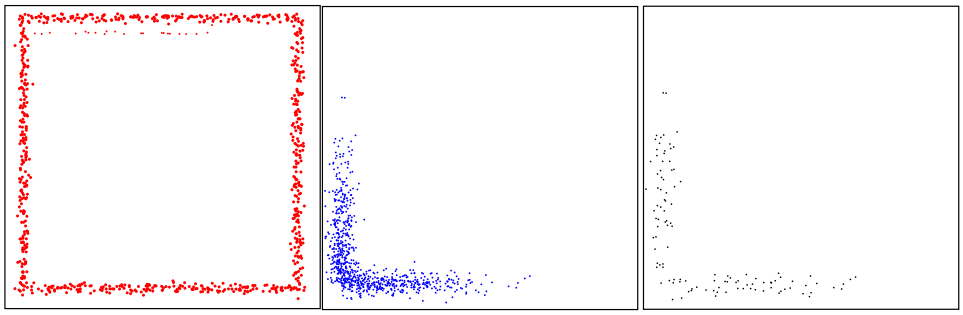

FIGURE 16.8: *On the* **left** *a map of a simple arena, represented as a point cloud. Such a map could be obtained by registering LIDAR measurements to one another. A LIDAR or depth sensor produces measurements in the sensor's coordinate system, and registering these measurements to the map will reveal 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

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 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 **??**).

16.2.5 Beyond ICP

ICP minimizes a cost function

$$\sum_i \left[\min_j \|\mathbf{x}_j - \mathcal{T}(\mathbf{y}_i)\|_2^2 \right] = \sum_i E_i(\mathbf{T}) \tag{16.2}$$

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 straightfor-

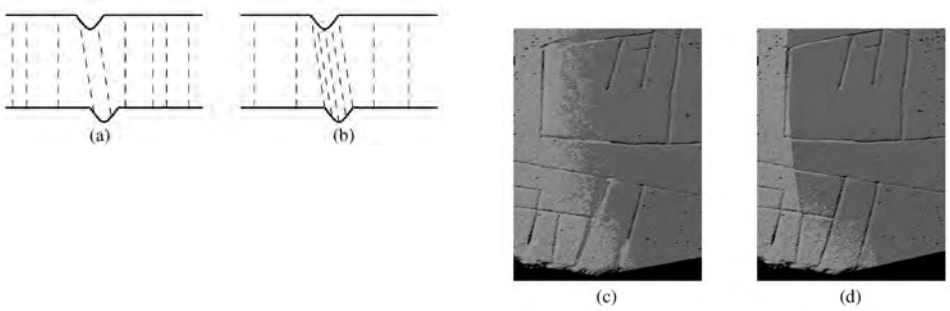

FIGURE 16.9: *The sample of points used in registration can be biased in useful ways. For example, (a) shows a cross section of a flat surface with a small groove (***above***) which needs to be registered to a similar surface (***below***). If point samples are drawn on the surface at random, then there 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

ward 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.

16.3 YOU SHOULD

16.3.1 remember these definitions:

16.3.2 remember these facts:

16.3.3 remember these procedures:

16.3.4 use these resources:

16.3.5 be able to:

- Apply IRLS and RANSAC to registration problems.
- Apply ICP to registration problems.

EXERCISES

QUICK CHECKS

16.1. In the lead, I say: " Correspondences that are wrong tend to be badly wrong". Why is this the case?

16.2. Check that I have correctly mapped IRLS (Section 14.2.2) onto registration in Section 16.1.

16.3. Check that I have correctly mapped RANSAC (Section 14.3) onto registration in Section 16.1.2.

16.4. Show how an affine transformation in d dimensions is exactly specified by $d+1$ correspondences (start with $d = 1$).

16.5. Produce a set of two correspondences that can't be exactly registered with a Euclidean transformation in 2D.

16.6. Produce a set of three correspondences that can be exactly registered with a Euclidean transformation in 2D.

16.7. Show that $d + 2$ correspondences are enough to exactly specify a projective transformation in d dimensions.

16.8. Section 16.2 has: " 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." Explain.

16.9. Imagine you obtain two LIDAR images of the same object from two different locations. Why do you not expect a near exact correspondence between the points in these two point clouds? (hint: this *isn't* about noise).

PROGRAMMING EXERCISES

16.10. Implement a simple IRLS for 2D Euclidean transformations.

- **(a)** Use this to reproduce Figure 16.1 and Figure 16.3, using a rectangle like that in the figures.
- **(b)** Now use your implementation to reproduce Figure 16.1 and Figure 16.3, but now using an ellipse with aspect ratio 3.
- **(c)** Now use your implementation to reproduce Figure 16.1 and Figure 16.3, but now using an ellipse with aspect ratio 1.05 – how often do you get the rotation right? is this what you expect? why?
- **(d)** Now use your implementation to reproduce Figure 16.1 and Figure 16.3, but now using a shape like that of Figure 16.4. Is this registration sensitive to uneven sampling of the shape?

16.11. Implement a simple RANSAC for 2D Euclidean transformations.

- **(a)** How well does this behave under the conditions of Figure 16.3, using a rectangle like that in the figures?
- **(b)** How well does this behave under the conditions of Figure 16.3, now using an ellipse with aspect ratio 3?
- **(c)** How well does this behave under the conditions of Figure 16.3, now using an ellipse with aspect ratio 1.05? how often do you get the rotation right? is this what you expect? why?
- **(d)** How well does this behave under the conditions of Figure 16.3, now using a shape like that of Figure 16.4. Is this registration sensitive to uneven sampling of the shape?

16.12. Implement a simple ICP for 2D Euclidean transformations.

- **(a)** Use this to produce figures like Figure 16.4 and Figure 16.5.
- **(b)** Investigate the effect of shape on how often at which your ICP converges to a bad transformation.
- **(c)** Investigate the effect of sampling on how often at which your ICP converges to a bad transformation. Trying to reproduce something like Figure 16.7 is a good place to start.