CHAPTER 15

Registration and Overlapping Images

Section 4.3.2 slid color separations over one another to get a best match, and so a registered color image. Generalizing this procedure is really useful. Back in the days when photographs were printed on paper by special stores, one way to make a photograph of a large object was to take several different, overlapping pictures; print them; place one printed image down on a corkboard; then slide the other printed pictures around on a corkboard until the overlapping sections of each pair of pictures show the same things; and then pin them down. The result is a *mosaic* – a collection of pictures that have been registered (Figure 15.2). Two pictures are *registered* when the overlapping sections of picture show the same things as much as possible.

Section 4.3.2 showed how to register when the transformation was a translation. In Section 15.1, I describe how to build mosaics in some detail for this case, because this allows me to abstract away from most of the machinery of registration and focus on the useful and interesting product. Registering images when the transformation is more complicated than a translation requires quite different procedures; Section 15.2 gives an overview, and Section **??** describes these procedures for simple cases. The harder cases appear in Chapter **??**.

15.1 REGISTERING IMAGES BY TRANSLATION

There are a number of reasons to build mosaics. You might simply not have the right camera, and so have to assemble a big picture out of small ones. You might be able to improve resolution in the overlapping portions, because there you have multiple estimates of pixel values. You might be able to identify moving objects or important changes by comparing registered images.

For the moment, assume we have two images $\mathcal{A}$ (which is $a_M \times a_N$) and $\mathcal{B}$ (which is $b_M \times b_N$). These two images are overlapping views of a scene that can be aligned exactly by a translation. The fact that they can be aligned means that there is some t_x, t_y so that $\mathcal{A}_{ij} \approx \mathcal{B}_{i-t_x,j-t_y}$ for those pixels where the images overlap. Visualize this as placing $\mathcal{B}$ on top of $\mathcal{A}$, then sliding $\mathcal{B}$ by t_x, t_y; then the parts of $\mathcal{A}$ and $\mathcal{B}$ that overlap look the same (Figure 15.1). We are given $\mathcal{A}$ and $\mathcal{B}$ and must find t_x, t_y. For the moment, assume that t_x, t_y are integers.

15.1.1 Building Mosaics by Search

Registering $\mathcal{A}$ and $\mathcal{B}$ follows the recipe of Section 4.3.2: choose a function that is smallest when $\mathcal{A}$ is registered with $\mathcal{B}$; now search for the (t_x, t_y) that yield best value of the cost function.

If the two images really do agree on the overlap, then the SSD is a good choice of cost function. Find the minimum by computing the value of the SSD for each translation that results in an overlap, then taking the smallest. If the range of translations is large, this procedure could be inefficient (improvements in

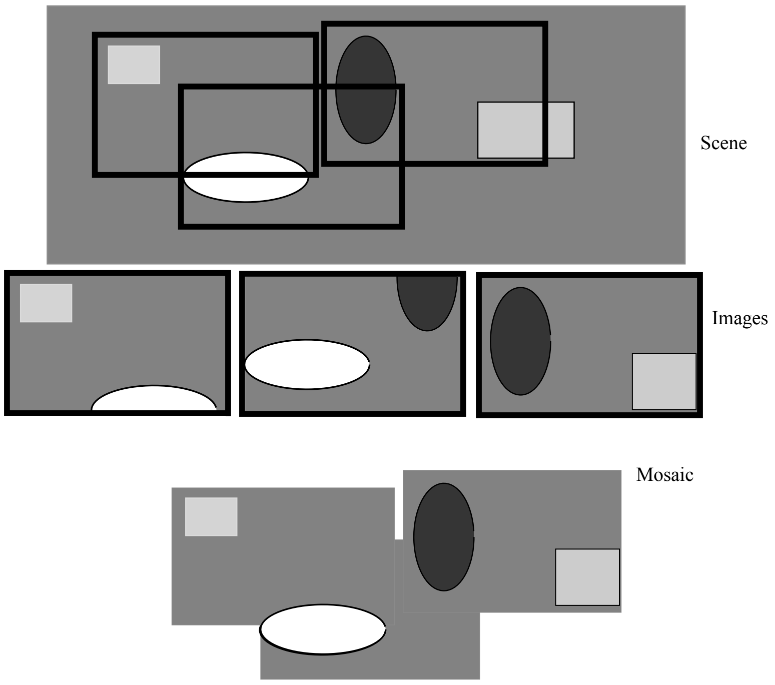

FIGURE 15.1: **Top** *shows a set of overhead images of a simple scene. The dark boundaries show each of three image frames.* **Center** *shows the actual images obtained in these frames. Notice how the first agrees with the second in a box (the intersection of first and second frames in the top part of the figure) and the second agrees with the third in another box.* **Bottom** *shows the mosaic that can be recovered by sliding images with respect to one another. This mosaic can't show features that haven't been imaged, but does show the relative configuration of scene components.*

Section 15.1.3), but it is a reasonable model for the moment. Notice that there is no point in translating both images (**exercises**). Choosing one image to stay in a fixed position is equivalent to choosing the coordinate system for the mosaic. The other image is then translated to the right position in that coordinate system.

One more step is required to go from two registered images to a mosaic. For pixels in the overlapping region, there are two estimates of the pixel value (one from $\mathcal{A}$ and one from $\mathcal{B}$). These need to be combined in some way. Not much can be done with two values, but a mosaic will be generally be constructed out of N different images, so there might be as many as N different estimates at a given location. It turns out that there are a variety of interesting possibilities here (Section **??**). This yields the following recipe for building a mosaic, sketched in Figure 15.2.

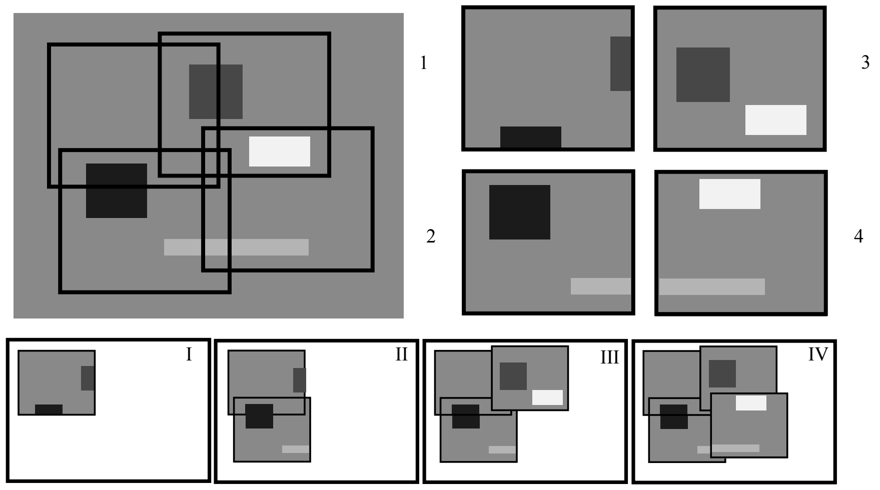

FIGURE 15.2: **Top left** *shows a simple scene with the location of four images; these are shown on the* **top right**. *The steps of creating a mosaic are sketched in the Roman numeral panes on the* **bottom**. *In step I, $\mathcal{I}_1$ is placed in the mosaic; in step II, $\mathcal{I}_2$ is registered to $\mathcal{I}_1$. This proceeds until there aren't any images that overlap (step III, $\mathcal{I}_3$ to $\mathcal{I}_1$; step IV, $\mathcal{I}_4$ to $\mathcal{I}_3$; check this yields a tree*exercises *). Finally, the overlapping image information is turned into pixel values.*

Procedure: 15.1 *Building a Mosaic as a Simple Tree*

Start with a set of N images $\{\mathcal{I}_1, \ldots, \mathcal{I}_N\}$. Choose $\mathcal{I}_1$ to serve as the *root image* – the image that is never translated. This image is the initial mosaic, equivalently, the root of the tree. Now iterate the following step until there are no images left:

- **Place another image** by finding an image $\mathcal{I}_l$ which overlaps an image $\mathcal{I}_k$ already in the mosaic and register it to that image. Record the translation required to register $\mathcal{I}_k$ to $\mathcal{I}_l$.

You now have a tree of translations attached to each image exercises . **Summarize** the overlapping images into a large image.

The choice of root image may have consequences, and the choice of which image to place and which image it overlaps will certainly have consequences. For concreteness, choose $\mathcal{I}_k$ (the image outside the mosaic) and $\mathcal{I}_l$ (the image outside the mosaic) to be the outside-inside pair that look most like one another. You could test this, for example, by comparing very heavily downsampled versions of the images. In some applications, the image with a good overlap will be obvious. For example, if you build a mosaic out of overhead aerial images, the images are going to be timestamped, and the next image will overlap rather well with the

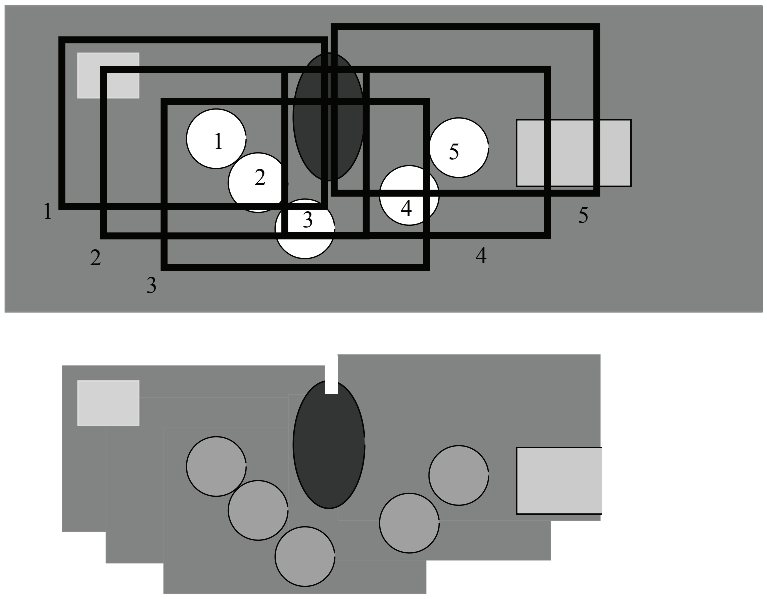

FIGURE 15.3: *Summarizing the mosaic with a mean shows a ghostly path of moving objects.* **Top** *shows a set of images of a simple scene with a moving object (the white circle). This is at location 1 in frame 1, and so on (location number on the object, frame number next to the frame). The background does not move. The frames are registered to one another to produce a mosaic.* **Bottom** *shows what happens if one averages. The circles affect the average, and appear – one can both background and object. Compare with Figure 15.4 and Figure 15.5.*

current image.

15.1.2 Summarizing Registered Images to form a Mosaic

There are a number of useful ways to summarize the registered images. At many locations, the recipe will produce a vector containing many components that are not unknown. Each is an estimate of the “true” pixel value for that location, but there is some error in this estimate. For example, if a small object is moving in the scene, it will affect some pixels in each frame. The main options to use as a summary of this vector are:

- The **mean**, best if there is no moving object, and one expects that the registration is very good. Averaging will tend to blur details if the images aren't precisely registered, and will show a blurry trail of moving objects, as Figure 15.3 shows and Figure **??** confirms.
- The **median**, best if there are moving objects and you want to suppress them; the median will tend to show the background (because more of the values from the registered images will be background), and does not blur as much in the face of misregistration, as Figure 15.4 shows and Figure **??** confirms.

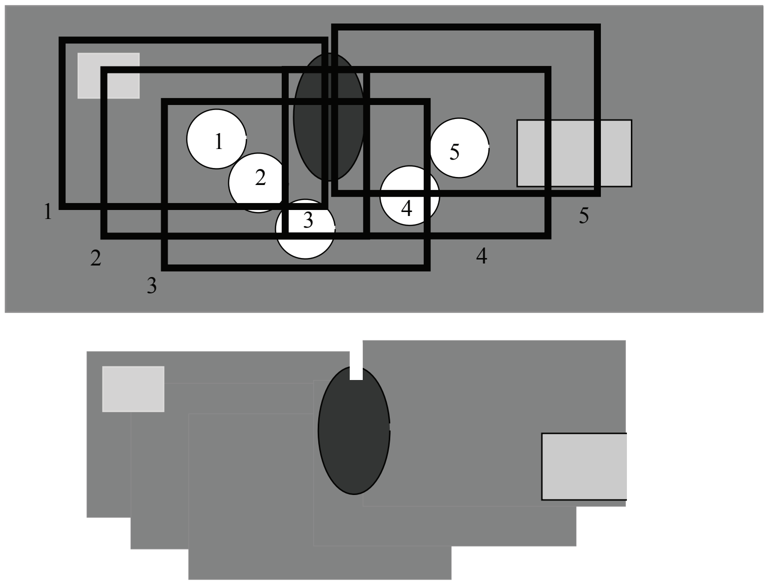

FIGURE 15.4: *Summarizing the mosaic with a median suppresses moving objects.* **Top** *shows a set of images of a simple scene with a moving object (the white circle). This is at location 1 in frame 1, and so on (location number on the object, frame number next to the frame). The background does not move. The frames are registered to one another to produce a mosaic.* **Bottom** *shows what happens if one summarizes using a median. The moving object spends little time at each pixel, and so does not change the median – the result shows the background. Compare with Figure 15.3 and Figure 15.5.*

- A **preferred image**, chosen in some way. Details will mostly not be blurred, and moving objects will largely be frozen in place (but might appear in more than one place). The outer edges of images can be prominent, but this can be suppressed by blending (Figures **??** and **??**) Figure 15.3 shows and Figure **??** confirms.

- The **value most different from the median**, if you want to show object motion. This will tend to accentuate details, and show a moving object in its different locations (Figures 15.5 and **??**).

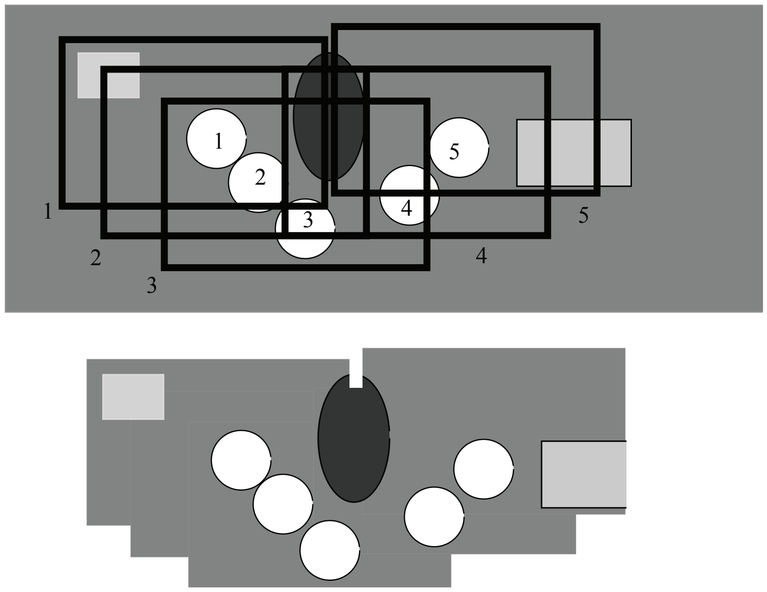

FIGURE 15.5: *Summarizing the mosaic with the value furthest from the median shows the path of moving objects.* **Top** *shows a set of images of a simple scene with a moving object (the white circle). This is at location 1 in frame 1, and so on (location number on the object, frame number next to the frame). The background does not move. The frames are registered to one another to produce a mosaic.* **Bottom** *shows what happens if one summarizes using the value most different from the median. The moving object spends little time at each pixel, and so is likely very different from the median – the result shows the moving object in different locations against the background. Compare with Figure 15.3 and Figure 15.4.*

Remember this: *Mosaics are built by registering images to one another. In the simplest case, you start with a root image; repeatedly register an image to the best overlapping image in the mosaic; then summarize all the images into a large image. Pixels in the large image may be covered by no mosaic image or by many. Different choices of summarization procedure yield different mosaics which emphasize different properties of the collection.*

15.1.3 Improved Translation Estimates

The obvious strategy for minimizing the cost function is: apply each translation; compute the objective function; then take the translation with smallest value. This will be slow if the range of translations is large. Notice that if we smoothed and subsampled $\mathcal{A}$ and $\mathcal{B}$ to produce a very coarse scale version (say, 8×8) of each image, we could compute a coarse estimate of m, n from those fairly easily because there are very few values to search. A coarse estimate of the translation that registers two coarse scale images can be refined using larger versions of the images. This observation leads to a really powerful search strategy: register very small versions of the images, then repeatedly refine the registration estimate using increasingly large versions until the full size images are registered. This strategy is known as *coarse to fine search.*

For example, look at the zebra's muzzle in Figure 3.9, and think about registering this image to itself. The 8×8 version has very few pixels, and looks like a medium dark bar, darker at the muzzle end. Finding a translation to register this image to itself should be fairly straightforward, and unambiguous. Assume we find m_8, n_8. In the 16×16 version, some stripes are visible. Registering this image to itself might be more difficult, because the stripes will create local minima of the cost function (check you follow this remark; think about what happens if you have the images registered, and then shift the muzzle perpendicular to the stripes). But if we have an estimate of the translation from the 8×8 version, we do not need to search a large range of translations to register the 16×16 version. We need to look only at four translations: $2 * m_8, 2 * n_8$; $2 * m_8 + 1, 2 * n_8$; $2 * m_8, 2 * n_8 + 1$; and $2 * m_8 + 1, 2 * n_8 + 1$. The same reasoning applies when going from the 16×16 version to the 32×32 version, and so on. It should be obvious that this is a natural application of a gaussian pyramid (Section **??**). Notice that the recipe applies even if one subsamples by less than two, though some details change (exercises).

Coarse-to-fine search suggests a way to find pairs of images with a good overlap: try to register coarse scale versions of the images with one another. Not all pairs will register, but those that do with a small enough translation will likely have a good overlap, and you can use this to obtain an estimate of the extent to which images overlap.

The translation that registers images can be estimated at a resolution that is finer than a single pixel. You should expect this – each image has very large numbers of pixels that are being used to estimate the translation. The main question is how to get estimates of the translation at a fine resolution. One straightforward procedure is to upsample the images, using an interpolate, and register the results. The coarse-to-fine search procedure still applies. For example, you could add layers to the gaussian pyramid by simply upsampling the images to appropriate sizes, then proceed as before.

An alternative strategy is to interpolate the cost function. Recall the original translations were estimated by evaluating the cost function at a set of translations, then finding the translation with the best cost. Now interpolate the cost function, and obtain the minimum of the interpolate. This works only if the interpolate has a minimum that isn't on a grid point, so at least the bicubic interpolate of Exercise **** is required. A minor issue results. If the translation isn't an integer, then the sampled values of one of two registered images aren't on grid points. This is easily dealt with by interpolation.

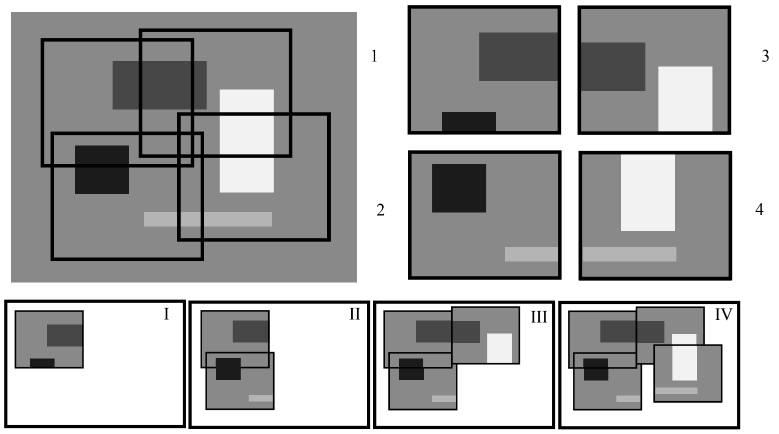

FIGURE 15.6: **Top left** *shows a simple scene with the location of four images; these are shown on the* **top right**. *The steps of creating a mosaic are sketched in the Roman numeral panes on the* **bottom**. *In this case, the translation of the fourth image with respect to the first is poorly estimated; more detail in the text.*

Remember this: *Translation estimates can be made more efficient by coarse-to-fine search. You can estimate translations at a finer resolution than the pixel grid by either upsampling the images, or by interpolating the cost function.*

15.1.4 Bundle Adjustment

Our simple procedure does not produce the best possible mosaic. To see this, assume you have images $\mathcal{I}_1, \ldots, \mathcal{I}_4$, as in Figure 15.6, and you introduce them into the mosaic in that order. Write $\mathcal{T}_{2\rightarrow 1}$ for the translation to align $\mathcal{I}_2$ with $\mathcal{I}_1$ by translating $\mathcal{I}_2$ to the right place in the mosaic coordinate system. The effect of this translation is shown in step II in Figure 15.6. Now you estimate $\mathcal{T}_{3\rightarrow 1}$ to register $\mathcal{I}_3$ to $\mathcal{I}_1$ (Step III). Notice that the patterns in the scene have been chosen to show an exaggerated case where the registration error between $\mathcal{I}_3$ and $\mathcal{I}_1$ is likely to be large (many different horizontal translations of $\mathcal{I}_3$ will register with $\mathcal{I}_1$ well). Finally, you estimate $\mathcal{T}_{4\rightarrow 3}$ (Step IV). Notice how error has cascaded. $\mathcal{I}_3$ is poorly registered to $\mathcal{I}_1$, and $\mathcal{I}_4$ is registered to $\mathcal{I}_3$, meaning that $\mathcal{I}_4$ is poorly registered to $\mathcal{I}_1$.

Notice that this isn't necessary. Some pixels of $\mathcal{I}_4$ overlap $\mathcal{I}_2$. If these pixels contributed to the registration, $\mathcal{I}_4$ and $\mathcal{I}_3$ could be properly registered. This problem occurs quite generally – it is not just a result of an odd scene – and is often referred to as a failure of *loop closure*. This refers to the idea that if 2 is registered to 1, 3 is registered to 2, 4 is registered to 3, all the way up to N is registered to $N-1$, N may not be registered to 1 at all well – the loop does not close.

There are several procedures to prevent or control this kind of error propagation. Start by registering the images by pairs as described. The easiest strategy is now to repeat: fix all but one image, then register that image to all the others it overlaps with. This approach can work, but may be slow, because it may take a long time for improvements to propagate across all the images. The alternative, which is better but can be onerous, is to fix one image in place, then adjust all others to register in every overlap. This isn't a straightforward optimization problem. There is no need to resolve it in detail yet (but see Section 22.6 if you're concerned).

Remember this: *The simple mosaic recipe can result in serious loop closure problems. This occurs because registering images only along the edges of a tree omits many important pairs of images. Bundle adjustment improves the registration by registering all pairs, usually starting with a mosaic built using the simple recipe.*

15.2 REGISTERING IMAGES WITH MORE THAN TRANSLATION

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.

However, most applications require that you deal with more interesting registration problems. Many camera movements cannot be modelled by a pure translation of the image. There might be rotation or scaling or (as Section 22.6 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.

Assume you have a set of N points in one image *and* you know the corresponding N points in a second image. Determining a transformation that registers the two images is now straightforward, if fiddly. Solutions to this case (Section 15.3) provide building blocks for more complicated cases. The correspondence comes from interest points.

Procedure: 15.2 *Master Recipe: Registering Two Images with Interest Points*

Given two images $\mathcal{A}$ and $\mathcal{B}$ you wish to register, find interest points in first and second image.
Find corresponding pairs as follows. For each interest point in $\mathcal{A}$ (write the feature vector of the i'th as $\mathbf{a}_i$), find the interest point in $\mathcal{B}$ whose feature vector is most similar. Assume that this is the j'th point in $\mathcal{B}$. 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 $\mathbf{b}_j$. Finally, check the distance between matched feature vectors is small. Pairs which are best matches from $\mathcal{A}$ to $\mathcal{B}$ and from $\mathcal{B}$ to $\mathcal{A}$ and where the distance is small are likely to be corresponding points if the distance threshold is small enough.
Now compute a transformation from the corresponding pairs of points.

If all the correspondences are right (which happens), then the procedures of Section 15.3 are good enough. But if some of the correspondences are wrong, you need to procedures of Chapter **??**.

Remember this: *Register two images by finding interest points, establishing correspondences, then using an appropriate registration algorithm on the correspondences.*

15.3 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.

15.3.1 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

$$\mathsf{Tr}\left(\mathcal{ABC}\right) = \mathsf{Tr}\left(\mathcal{BCA}\right) = \mathsf{Tr}\left(\mathcal{CAB}\right)$$

(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 $\mathcal{V}^T\mathcal{W}\mathcal{V}$ 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: 15.3 *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

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

15.3.2 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.

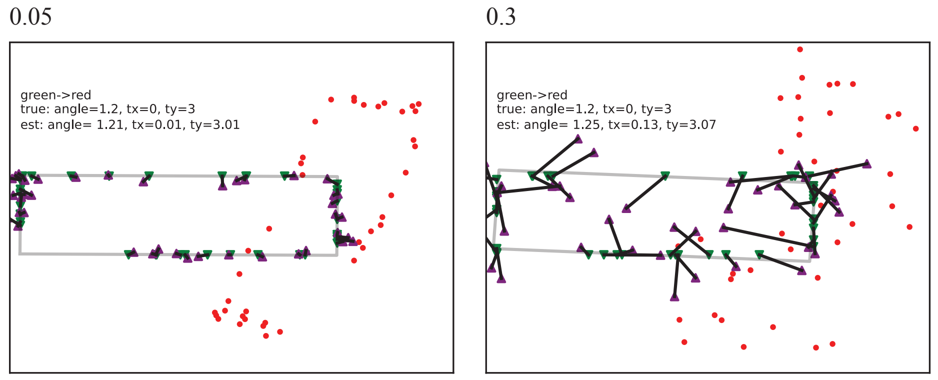

FIGURE 15.7: *Least squares registration is quite well-behaved under even quite pronounced gaussian noise. In each figure, the 40* **green** *(downward pointing) triangles, which lie on a rectangle one unit high and three units wide are subject to a Euclidean transformation, then noise is added, to obtain 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* **if** *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.*

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 useful). The objective function can be transformed to

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

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 22.6 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: 15.4 *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

$$\sum_i w_i(\mathbf{x}_i - \mathcal{R}\mathbf{y}_i - \mathbf{t})^T(\mathbf{x}_i - \mathcal{R}\mathbf{y}_i - \mathbf{t}) \tag{15.1}$$

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

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

15.4 PROJECTIVE TRANSFORMATIONS

Recall from Section 4.1 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

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

The reason to put up with the relative complexity of this model is that it explains what happens to a pattern on a plane when the pattern is viewed in a pinhole camera (Section 15.4.1).

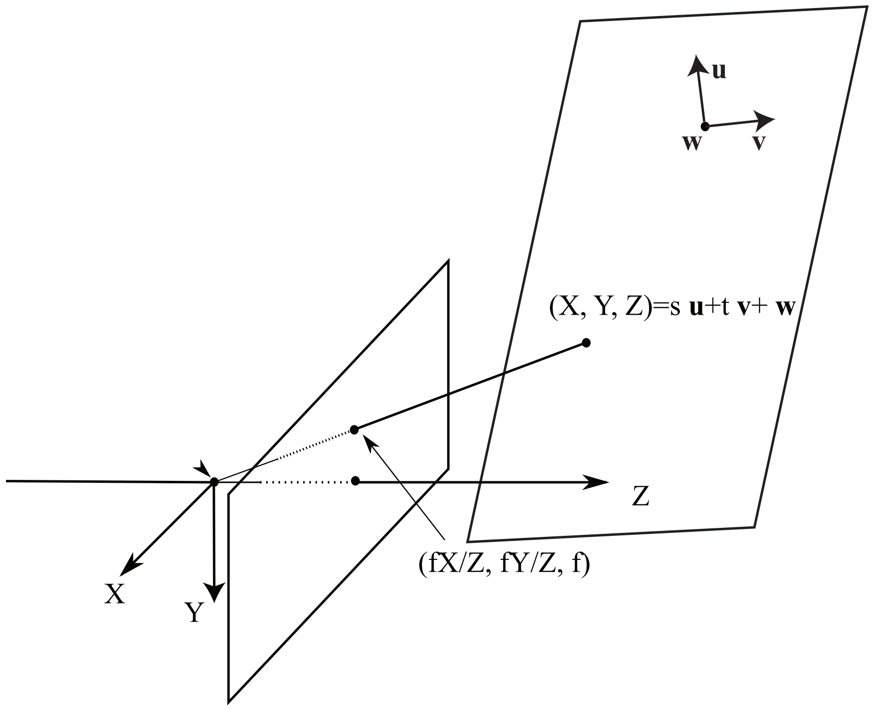

FIGURE 15.8: *The mapping from a pattern on a plane to a perspective image, or from a perspective image to a pattern on a plane, is a homography. The 3D points can be written $s\mathbf{u} + t\mathbf{v} + \mathbf{w}$, where each point on the plane has parameters (s, t). The derivation follows in the main text.*

Fitting a projective transformation to a set of points isn't (much) affected by dimension. Higher dimensions follow the pattern for the transformation. 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

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

15.4.1 Homographies: A Camera Viewing a Plane

A pinhole camera views a pattern on a plane (Figure 15.8). The mapping from the pattern on the plane, in the plane's coordinate system, to the pattern in the image, in the image coordinate system, is a projective transformation. This means the mapping from the image to the plane is also a projective transformation. If you know some information (for example, where the plane is with respect to the camera; the locations of some reference points on the plane), you can reconstruct the pattern in a frontal view. This is extremely useful in practice.

The coordinate system on the plane is $(s, t)^T$, and the points on the plane in 3D are parametrized by $s\mathbf{u} + t\mathbf{v} + \mathbf{w}$, where $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{w}$ are vectors in 3D and $\mathbf{u}$, $\mathbf{v}$ are not parallel. Recall from Section **??** the geometric model that the pinhole camera maps the point $(X, Y, Z)^T$ in 3D to the point $(fX/Z, fY/Z)^T$ on the image plane. In turn, the point $(s, t)^T$ on the plane maps to

$$\begin{bmatrix} f\frac{su_1+tv_1+w_1}{su_3+tv_3+w_3} \\ f\frac{su_2+tv_2+w_2}{su_3+tv_3+w_3} \end{bmatrix}$$

in the image. This means the map from plane to image is a projective transformation, as is the map from image to plane. In an extremely common case, you know the locations of four or more points on the plane *and* the location of those points in the image, and must reconstruct the pattern on the plane from the pattern on the image. You do so by estimating a projective transformation.

15.4.2 Registering with a Projective Transformation

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

$$\mathbf{r}_i = \mathbf{x}_i - \mathcal{M}(\mathbf{y}_i).$$

A weighted least squares solution now solves

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

The main issue here is that $\mathcal{M}(\mathbf{y}_i)$ is not a linear function of the components of $\mathcal{M}$. Numerical minimization is required. You should use a second order method (Levenberg-Marquardt is favored **exercises**). Experience teaches that this optimization is not well behaved without a strong start point.

Procedure: 15.5 *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:

$$\sum_i w_i \xi_i^T \xi_i \tag{15.2}$$

where

$$\xi_i = \begin{bmatrix} x_{i,1} - \dfrac{m_{11}y_{i,1} + \ldots + m_{1d}y_{i,d} + m_{1(d+1)}}{m_{(d+1)1}x_{i,1} + \ldots + m_{(d+1)d}x_{i,d} + m_{(d+1)(d+1)}} \\ \ldots \\ x_{i,d} - \dfrac{m_{d1}y_{i,1} + \ldots + m_{dd}y_{i,d} + m_{d(d+1)}}{m_{(d+1)1}x_{i,1} + \ldots + m_{(d+1)d}x_{i,d} + m_{(d+1)(d+1)}} \end{bmatrix} \tag{15.3}$$

using a second-order method (a quasi-Newton method like Levenberg-Marquardt is favored). Use a start point obtained with the procedure below.

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

$$\begin{bmatrix} 0 \\ \ldots \\ 0 \end{bmatrix} = \begin{bmatrix} \left(m_{11}y_{1,i} + \ldots + m_{1d}y_{d,i} + m_{1(d+1)}\right) - \\ x_{1,i}\left(m_{(d+1)1}y_{1,i} + \ldots + m_{(d+1)d}y_{d,i} + m_{(d+1)(d+1)}\right) \\ \ldots \\ \left(m_{d1}y_{1,i} + \ldots + m_{dd}y_{d,i} + m_{d(d+1)}\right) - \\ x_{d,i}\left(m_{(d+1)1}y_{1,i} + \ldots + m_{(d+1)d}y_{d,i} + m_{(d+1)(d+1)}\right) \end{bmatrix} = \mathcal{D}\mathbf{m}.$$

Here the m_{ij} are unknown, so this is a set of d homogenous linear equations in $(d+1) \times (d+1)$ unknowns. I have arranged these unknowns into a vector and the coefficients into a matrix $\mathcal{D}$ for convenience. If you have $(d+2)$ 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.

If you have more than $(d+2)$ pairs, you can use least squares. Because the equations are homogenous, you must constrain the scale of $\mathbf{m}$, so minimize $\mathbf{m}^T\mathcal{D}^T\mathcal{D}\mathbf{m}$ subject to $\mathbf{m}^T\mathbf{m} = 1$. **exercises** The resulting estimate of $\mathcal{M}$ has a good reputation as a start point for a full optimization. It is straightforward to incorporate weights on the points into this estimate. 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: 15.6 *Obtaining a start point for fitting a projective transformation with least squares*

Write $\mathcal{D}$ for

$$\begin{bmatrix} y_{1,1} & y_{2,1} & \ldots & 1 & 0 & \ldots & 0 & x_{1,1}y_{1,1} & \ldots & x_{1,1}y_{d,1} & 1 \\ y_{1,1} & y_{2,1} & \ldots & 1 & 0 & \ldots & 0 & x_{2,1}y_{1,1} & \ldots & x_{2,1}y_{d,1} & 1 \\ \ldots \\ y_{1,1} & y_{2,1} & \ldots & 1 & 0 & \ldots & 0 & x_{d,1}y_{1,1} & \ldots & x_{d,1}y_{d,1} & 1 \\ \ldots \\ y_{1,N} & y_{2,N} & \ldots & 1 & 0 & \ldots & 0 & x_{d,N}y_{1,N} & \ldots & x_{d,N}y_{d,N} & 1 \end{bmatrix}$$

and $\mathbf{m}$ for

$$\begin{bmatrix} m_{11} \\ m_{12} \\ \ldots \\ m_{1d} \\ m_{21} \\ \ldots \\ m_{d,(d+1)} \\ m_{(d+1),1} \\ \ldots \\ m_{(d+1),d} \\ m_{(d+1),(d+1)} \end{bmatrix}$$

Set up the set of homogenous linear equations

$$\mathbf{0} = \mathcal{D}\mathbf{m}.$$

Write $\mathcal{W} = \mathsf{diag}\,(w_i)$, and obtain $\hat{\mathbf{m}}$, the eigenvector of $\mathcal{D}^T\mathcal{W}\mathcal{D}$ corresponding to the smallest eigenvalue. This is your start point.

There is one important question to attend to here. The coordinate system in which the points are presented can have effects. It is often a good idea to translate and scale both $\mathbf{x}_i$ and $\mathbf{y}_i$ so that coordinates are all in the range $[-1, 1]$ and each has a center of gravity at $\mathbf{0}$. This avoids an optimization problem with some large terms and some small terms, and often results in a better fit. Once you have the estimated transformation, you can apply translations and scales as appropriate to estimate the transformation between the original coordinate systems **exercises** .

15.4.3 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**).

15.5 YOU SHOULD

15.5.1 remember these definitions:

15.5.2 remember these facts:

15.5.3 remember these procedures:

15.5.4 use these resources:

15.5.5 be able to:

- Register two images using interest points when correspondences are accurate.
- Construct a simple mosaic by registering images.

EXERCISES

QUICK CHECKS

15.1. Section 15.1.1 has: "Notice that there is no point in translating both images (**exercises**)." Explain.

15.2. Procedure 15.1 produces a tree. Explain.

15.3. Why would Procedure 15.1 not produce a forest? Does a forest make sense?

15.4. Sketch the tree of Figure 15.2.

15.5. Section 15.1.1 has: "You could test this, for example, by comparing very heavily downsampled versions of the images." How would this work?

15.6. Section 15.3.1 says: " 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 – works fine" What is the linear system you would solve to find $\mathcal{M}$?

15.7. Check that

$$\mathsf{Tr}\left(\mathcal{ABC}\right) = \mathsf{Tr}\left(\mathcal{BCA}\right) = \mathsf{Tr}\left(\mathcal{CAB}\right)$$

for 1×1 matrices; now check this for 2×2 matrices by writing the whole thing out.

15.8. You have a dataset of N points $\mathbf{y}_i$. Write the center of gravity for these points as $\mathbf{c}$. Check the center of gravity of the points $\mathcal{M}\mathbf{y}_i + \mathbf{t}$ is $\mathcal{M}\mathbf{c} + \mathbf{t}$.

15.9. Section 15.4.2 has: "you can apply translations and scales as appropriate to estimate the transformation between the original coordinate systems." Explain.

LONGER PROBLEMS

15.10. This exercise uses the notation of Section 15.3.1.

(a) Check that

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

is equivalent to

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

(b) Now show

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

will be minimized when

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

15.11. This exercise uses the notation of Section 15.3.1, and establishes cases where $\mathcal{V}^T\mathcal{W}\mathcal{V}$ has full rank. Assume points have dimension d and there are $N > d$ points.

(a) Show that, if the N points $\mathbf{v}_i$ contain a set of d points that are linearly independent, then the only $\mathbf{a}$ such that $\mathcal{V}\mathbf{a} = \mathbf{0}$ is $\mathbf{0}$.

(b) Assume all diagonal elements of $\mathcal{W}$ are greater than zero. Use the result of the previous subexercise to show that, if the N points $\mathbf{v}_i$ contain a set of d points that are linearly independent, then $\mathcal{V}^T\mathcal{W}\mathcal{V}$ has full rank.

(c) Show that, if the N points $\mathbf{v}_i$ do not contain a set of d points that are linearly independent, then $\mathcal{V}^T\mathcal{W}\mathcal{V}$ does not have full rank, whatever $\mathcal{W}$.

(d) Deduce a condition for $\mathcal{V}^T\mathcal{W}\mathcal{V}$ to have full rank.

15.12. This exercise uses the notation of Section 15.4.2. Assume you have $N \geq d+1$ different $(\mathbf{x}, \mathbf{y})$ pairs, and must find a start point.

(a) Show that the expression for $\mathcal{D}$ given in Procedure 15.6 is correct.

(b) Show that minimizing $\mathbf{m}^T\mathcal{D}^T\mathcal{W}\mathcal{D}\mathbf{m}$ does not produce a solution.

(c) Show that the solutions to minimizing $\mathbf{m}^T\mathcal{D}^T\mathcal{W}\mathcal{D}\mathbf{m}$ subject to $\mathbf{m}^T\mathbf{m} = 1$ solve

$$\mathcal{D}^T\mathcal{W}\mathcal{D}\mathbf{m} = \lambda\mathbf{m}.$$

Which of the $(d+1)^2$ eigenvectors is the solution you want?

(d) Why should you *not* try to minimize $\mathbf{m}^T\mathcal{D}^T\mathcal{W}\mathcal{D}\mathbf{m}$ subject to $\mathbf{1}^T\mathbf{m} = 0$?

PROGRAMMING EXERCISES

15.13. Obtain an image of a plane with a pattern on it, where the pattern contains something you know to be a grid of square checks (I used an image search, with the query "perspective art" and found lots). Usually, there will be other stuff in the image; if this bothers you, mask it off by hand. Ensure you have more than 5 checks on the piece – if you don't, just find another image.

(a) Choose one check. Find the four vertices of this check in the image. Compute the homography that rectifies these four vertices to a unit square, and apply it to the rest of the pixels on the plane. You will likely observe that checks far from the one you used are not quite square. Why is this happening?

(b) Now compute the homography that rectifies these at least three of the checks to the appropriate points on a grid of unit squares. Apply it to the rest of the pixels on the plane. Are the other checks – the ones you did not use – better?

(c) Obtain a drawing – your choice – and place four equal dark squares cut from black paper at each corner. These squares should sit on the corner of a grid. Take a picture of this, ensuring the assembly is flat and that you get visible perspective distortions in the picture (for example, the squares might no longer look square). Now use the results of the previous exercise to rectify the picture. How good is the rectification? what could you use to improve it?