Chapter 11

Matching in 2D

This chapter explores how to make and use correspondences between images and maps, images and models, and images and other images. All work is done in two dimensions; methods are extended in Chapter 14 to 3D-2D and 3D-3D matching. There are many immediate applications of 2D matching which do not require the more general 3D analysis.

Consider the problem of taking inventory of land use in a township for the purpose of planning development or collecting taxes. A plane is dispatched on a clear day to take aerial images of all the land in the township. These pictures are then compared to the most recent map of the area to create an updated map. Moreover, other databases are updated to record the presence of buildings, roads, oil wells, etc., or perhaps the kind of crops growing in the various fields. This work can all be done by hand, but is now commonly done using a computer. A second example is from the medical area. The bloodflow in a patient's heart and lungs is to be examined. An X-ray image is taken under normal conditions, followed by one taken after the injection into the bloodstream of a special dye. The second image should reveal the bloodflow, except that there is a lot of noise due to other body structures such as bone. Subtracting the first image from the second will reduce noise and artifact and emphasize only the changes due to the dye. Before this can be done, however, the first image must be *geometrically transformed* or *warped* to compensate for small motions of the body due to body positioning, heart motion, breathing, etc.

11.1 Registration of 2D Data

A simple general mathematical model applies to all the cases in this Chapter and many others not covered. Equation 11.1 and Figure 11.1 show an invertible mapping between points of a model M and points of an image I. Actually, M and I can each be any 2D coordinate space and can each represent a map, model, or image.

$$\begin{aligned} M[x,y] &= I[g(x,y),\ h(x,y)] \\ I[r,c] &= M[g^{-1}(r,c), h^{-1}(r,c)] \end{aligned} \tag{11.1}$$

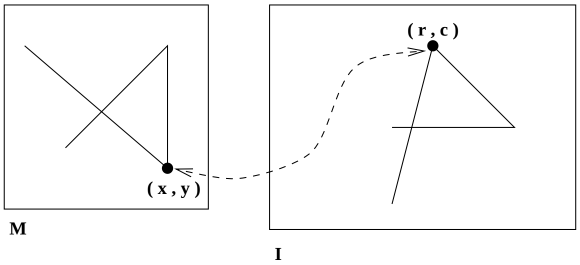


Figure 11.1: A mapping between 2D spaces **M** and **I**. **M** may be a model and **I** an image, but in general any 2D spaces are possible.

1 Definition *The mapping from one 2D coordinate space to another as defined in Equation 11.1 is sometimes called a spatial transformation, geometric transformation, or warp (although to some the term warp is reserved for only nonlinear transformations).*

The functions g and h create a correspondence between model points $[x, y]$ and image points $[r, c]$ so that a point feature in the model can be located in the image: we assume that the mapping is invertible so that we can go in the other direction using their inverses. Having such mapping functions in the tax record problem allows one to transform property boundaries from a map into an aerial image. The region of the image representing a particular property can then be analyzed for new buildings or crop type, etc. (Currently, the analysis is likely to be done by a human using an interactive graphics workstation.) Having such functions in the medical problem allows the radiologist to analyze the difference image $I_2[r_2, c_2] - I[g(r_2, c_2), h(r_2, c_2)]$: here the mapping functions *register* like points in the two images.

2 Definition *Image registration is the process by which points of two images from similar viewpoints of essentially the same scene are geometrically transformed so that corresponding feature points of the two images have the same coordinates after transformation.*

Another common and important application, although not actually a matching operation, is creation of a new image by collecting sample pixels from another image. For example, as shown in Figure 11.2, we might want to cut out a single face I_2 from an image I_1 of several people as shown in Figure 11.2. Although the content of the new image I_2 is a subset of the original image I_1, it is possible that I_2 can have the same number of pixels as I_1 (as is the case in the figure), or even more.

There are several issues of this theory which have practical importance. What is the form of the functions g and h, are they linear, continuous, etc? Are straight lines in one space mapped into straight or curved lines in the other space? Are the distances between point pairs the same in both spaces? More important, how do we use the properties of different functions to achieve the mappings needed? Is the 2D space of the model or image continuous or discrete? If at least one of the spaces is a digital image then quantization effects will impact both accuracy and visual quality. (The quantization effects have deliberately been kept in the right image of Figure 11.2 to demonstrate this point.)

Figure 11.2: New image of a face (right) cut out of original image (left) using a sampling transformation. The face at the right is the rightmost face of the five at the left.

Exercise 1

Describe how to enhance the right image of Figure 11.2 to lessen the quantization or aliasing effect.

11.2 Representation of Points

In this chapter, we work specifically with points from 2D spaces. Extension of the definitions and results to 3D is done later in Chapter 13; most, but not all, extensions are straightforward. It is good for the student to master the basic concepts and notation before handling the increased complexity sometimes present in 3D. A 2D point has two coordinates and is conveniently represented as either a row vector $\mathbf{P} = [x, y]$ or column vector $\mathbf{P} = [x, y]^t$. The column vector notation will be used in our equations here to be consistent with most engineering books which apply transformations $\mathbf{T}$ on the left to points $\mathbf{P}$ on the right . For convenience, in our text we will often use the row vector form and omit the formal transpose notation t. Also, we will separate coordinates by commas, something that is not needed when a column vector is displayed vertically.

$$\mathbf{P} \;=\; [x, y]^t \;=\; \begin{bmatrix} x \\ y \end{bmatrix}$$

Sometimes we will have need to label a point according to the type of feature from which it was determined. For example, a point could be the center of a hole, a vertex of a polygon, or the computed location where two extended line segments intersect. Point type will be used to advantage in the automatic matching algorithms discussed later in the chapter.

Reference Frames

The coordinates of a point are always relative to some coordinate frame. Often, there are several coordinate frames needed to analyze an environment, as discussed at the end of Chapter 2. When multiple coordinate frames are in use, we may use a special superscript to denote which frame is in use for the coordinates being given for the point.

3 DEFINITION *If* $\mathbf{P_j}$ *is some feature point and* $\mathbf{C}$ *is some reference frame, then we denote the coordinates of the point relative to the coordinate system as* ${}^{c}\mathbf{P_j}$.

Homogeneous Coordinates

As will soon become clear, it is often convenient notationally and for computer processing to use *homogeneous coordinates* for points, especially when affine transformations are used.

4 Definition *The homogeneous coordinates of a 2D point* $\mathbf{P} = [x, y]^t$ *are* $[sx, sy, s]^t$*, where s is a scale factor, commonly 1.0.*

Finally, we need to note the conventions of coordinate systems and programs that display pictures. The coordinate systems in the drawn figures of this chapter are typically plotted as they are in mathematics books with the first coordinate (x or u or even r) increasing to the right from the origin and the second coordinate (y or v or even c) increasing upward from the origin. However, our image display programs display an image of n rows and m columns with the first row (row $r = 0$) at the top and the last row (row $r = n - 1$) at the bottom. Thus r increases from the top downward and c increases from left to right. This presents no problem to our algebra, but may give our intuition trouble at times since the displayed image needs to be mentally rotated counterclockwise 90 degrees to agree with the conventional orientation in a math book.

11.3 Affine Mapping Functions

A large class of useful spatial transformations can be represented by multiplication of a matrix and a homogeneous point. Our treatment here is brief but fairly complete: for more details, consult one of the computer graphics texts or robotics texts listed in the references.

Scaling

A common operation is scaling. Uniform scaling changes all coordinates in the same way, or equivalently changes the size of all objects in the same way. Figure 11.3 shows a 2D point $\mathbf{P} = [1, 2]$ scaled by a factor of 2 to obtain the new point $\mathbf{P}' = [2, 4]$. The same scale factor is applied to the three vertices of the triangle yielding a triangle twice as large. Scaling is a *linear transformation*, meaning that it can be easily represented in terms of the scale factor applied to the two basis vectors for 2D Euclidean space. For example, $[1, 2] = 1[1, 0] + 2[0, 1]$ and $2[1, 2] = 2(1[1, 0] + 2[0, 1]) = 2[1, 0] + 4[0, 1] = [2, 4]$. Equation 11.2 shows how scaling of a 2D point is conveniently represented using multiplication by a simple matrix containing the scale factors on the diagonal. The second case is the general case where the x and y unit vectors are scaled differently and is given in Equation 11.3. Recall the five coordinate frames introduced in Chapter 2: the change of coordinates of real image points expressed in mm units to pixel image points expressed in row and column units is one such scaling. In the case of a square pixel camera, $c_x = c_y = c$, but these constants will be in the ratio of 4/3 for cameras built using TV standards.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} c & 0 \\ 0 & c \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} cx \\ cy \end{bmatrix} = c \begin{bmatrix} x \\ y \end{bmatrix} \tag{11.2}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} c_x & 0 \\ 0 & c_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} c_x x \\ c_y y \end{bmatrix} \tag{11.3}$$

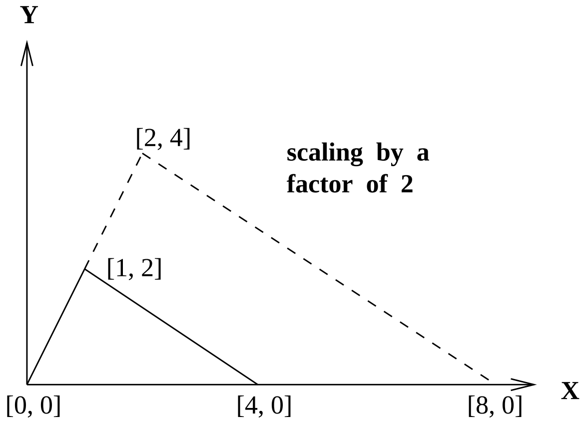


Figure 11.3: Scaling both coordinates of a 2D vector by scale factor 2.

Exercise 2 scaling for a non-square pixel camera

Suppose a square CCD chip has side 0.5 inches and contains 480 rows of 640 pixels each on this active area. Give the scaling matrix needed to convert pixel coordinates [r, c] to coordinates [x, y] in inches. The *center of* pixel [0,0] corresponds to [0, 0] in inches. Using your conversion matrix, what are the integer coordinates of the center of the pixel in row 100 column 200?

Rotation

A second common operation is rotation about a point in 2D space. Figure 11.4 (left) shows a 2D point $\mathbf{P} = [x, y]$ rotated by angle θ counterclockwise about the origin to obtain the new point $\mathbf{P'} = [x', y']$. Equation 11.4 shows how rotation of a 2D point about the origin is conveniently represented using multiplication by a simple matrix. As for any linear transformation, we take the columns of the matrix to be the result of the transformation applied to the basis vectors (Figure 11.4 (right)); transformation of any other vector can be expressed as a linear combination of the basis vectors.

$$\begin{aligned} R_\theta([x, y]) &= R_\theta(x[1, 0] \ + \ y[0, 1]) \\ &= xR_\theta([1, 0]) \ + \ yR_\theta([0, 1]) \ = \ x[cos\theta, sin\theta] \ + \ y[-sin\theta, cos\theta] \\ &= [xcos\theta - ysin\theta, \ \ xsin\theta + ycos\theta] \end{aligned}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} xcos\theta - ysin\theta \\ xsin\theta + ycos\theta \end{bmatrix} \tag{11.4}$$

2D rotations can be made about some arbitrary point in the 2D plane, which need not be the origin of the reference frame. Details are left for a guided exercise later in this section.

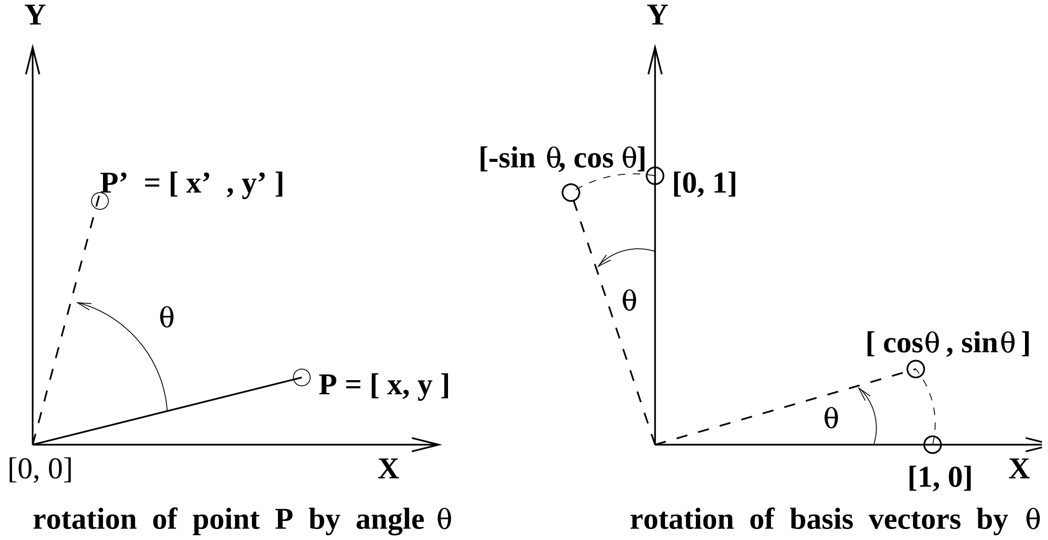


Figure 11.4: Rotation of any 2D point in terms of rotation of the basis vectors.

Exercise 3

(a) Sketch the three points [0,0], [2,2], and [0,2] using an XY coordinate system. (b) Scale these points by 0.5 using Equation 11.2 and plot the results. (c) Using a new plot, plot the result of rotating the three points by 90 degrees about the origin using Equation 11.4. (d) Let the scaling matrix be S and the rotation matrix be R. Let SR be the matrix resulting from multiplying matrix S on the left of matrix R. Is there any difference if we transform the set of three points using SR and RS?

---

* Orthogonal and Orthonormal Transforms

5 DEFINITION *A set of vectors is said to be orthogonal if all pairs of vectors in the set are perpendicular; or equivalently, have scalar product of zero.*

6 DEFINITION *A set of vectors is said to be orthonormal if it is an orthogonal set and if all the vectors have unit length.*

A rotation preserves both the length of the basis vectors and their othogonality. This can be seen both intuitively and algebraically. As a direct result, the distance between any two transformed points is the same as the distance between the points before transformation. A rigid transformation has this same property: a rigid transformation is a composition of rotation and translation. Rigid transformations are commonly used for rigid objects or for change of coordinate system. A uniform scaling that is not 1.0 does not preserve length; however, it does preserve the angle between vectors. These issues are important when we seek properties of objects that are invariant to how they are placed in the scene or how a camera views them.

Translation

Often, point coordinates need to be shifted by some constant amount, which is equivalent to changing the origin of the coordinate system. For example, row-column coordinates of a pixel image might need to be shifted to transform to latitude-longitude coordinates of a map. Since translation does not map the origin [0, 0] to itself, we cannot model it using a simple 2x2 matrix as has been done for scaling and rotation: in other words, it is not a linear operation. We can extend the dimension of our matrix to 3x3 to handle translation as well as some other operations: accordingly, another coordinate is added to our point vector to obtain *homogeneous coordinates.* Typically, the appended coordinate is 1.0, but other values may sometimes be convenient.

$$\mathbf{P} = [x, y] \ \simeq \ [wx, wy, w] \ = \ [x, y, 1] \ \ for\ w = 1$$

The matrix multiplication shown in Equation 11.5 can now be used to model the translation $\mathbf{D}$ of point $[x, y]$ so that $[x', y'] \ = \ \mathbf{D}([x, y]) \ = \ [x + x_0,\ y + y_0]$.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + x_0 \\ y + y_0 \\ 1 \end{bmatrix} \tag{11.5}$$

Exercise 4 rotation about a point

Give the 3x3 matrix that represents a $\pi/2$ rotation of the plane about the point [5, 8]. *Hint*: first derive the matrix $\mathbf{D}_{-5,-8}$ that translates the point [5, 8] to the origin of a new coordinate frame. The matrix which we want will be the combination $\mathbf{D}_{5,8}\ \mathbf{R}_{\pi/2}\ \mathbf{D}_{-5,-8}$. Check that your matrix correctly transforms points [5, 8],[6, 8] and [5, 9].

Exercise 5 reflection about a coordinate axis

A *reflection* about the y-axis maps the basis vector [1, 0] onto [-1, 0] and the basis vector [0, 1] onto [0, 1]. (a) Construct the matrix representing this reflection. (b) Verify that the matrix is correct by transforming the three points [1, 1], [1, 0], and [2, 1].

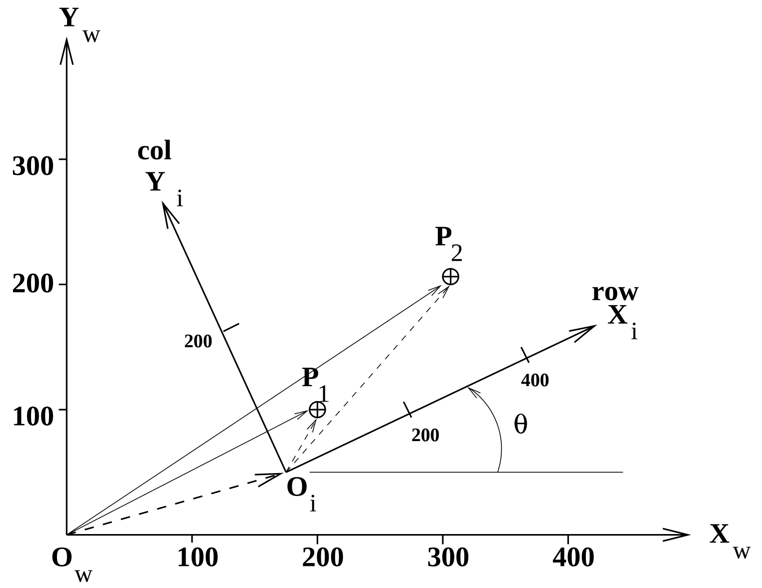


Figure 11.5: Image from a square-pixel camera looking vertically down on a workbench: feature points in image coordinates need to be rotated, scaled, and translated to obtain workbench coordinates.

Rotation, Scaling and Translation

Figure 11.5 shows a common situation: an image **I[r, c]** is taken by a square-pixel camera looking perpendicularly down on a planar workspace **W[x,y]**. We need a formula that converts any pixel coordinates $[r, c]$ in units of rows and columns to, say, mm units $[x, y]$. This can be done by composing a rotation **R**, a scaling **S**, and a translation **D** as given in Equation 11.6 and denoted ${}^w\mathbf{P_j} = \mathbf{D}_{x_0,y_0}\ \mathbf{S}_s\ \mathbf{R}_\theta {}^i\mathbf{P_j}$. There are four parameters that determine the mapping between row-column coordinates and x-y coordinates on the workbench; angle of rotation θ, scale factor s that converts pixels to mm, and the two displacements x_0, y_0. These four parameters can be obtained from the coordinates of two *control points* $\mathbf{P_1}$ and $\mathbf{P_2}$. These points are determined by clearly marked and easily measured features in the workspace that are also readily observed in the image – '+' marks, for example. In the land use application, road intersections, building corners, sharp river bends, etc. are often used as control points. It is important to emphasize that the same point feature, say $\mathbf{P_1}$, can be represented by two (or more) distinct vectors, one with row-column coordinates relative to **I** and one with mm x-y coordinates relative to **W**. We denote these representations as ${}^i\mathbf{P_1}$ and ${}^w\mathbf{P_1}$ respectively. For example, in Figure 11.5 we have ${}^i\mathbf{P_1} = [100, 60]$ and ${}^w\mathbf{P_1} = [200, 100]$.

7 DEFINITION *Control points are clearly distinguishable and easily measured points used to establish known correspondences between different coordinate spaces.*

Given coordinates for point $\mathbf{P_1}$ in both coordinate systems, matrix equation 11.6 yields two separate equations in the four unknowns.

$$\begin{bmatrix} x_w \\ y_w \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{11.6}$$

$$x_w = x_i s \cos\theta - y_i s \sin\theta + x_0 \tag{11.7}$$

$$y_w = x_i s \sin\theta + y_i s \cos\theta + y_0 \tag{11.8}$$

Using point $\mathbf{P_2}$ yields two more equations and we should be able to solve the system to determine the four parameters of the conversion formula. θ is easily determined independent of the other parameters as follows. First, the direction of the vector $\mathbf{P_1 P_2}$ in $\mathbf{I}$ is determined as $\theta_i = \arctan((^i y_2 - {}^i y_1)/(^i x_2 - {}^i x_1))$. Then, the direction of the vector in $\mathbf{W}$ is determined as $\theta_w = \arctan((^w y_2 - {}^w y_1)/(^w x_2 - {}^w x_1))$. The rotation angle is just the difference of these two angles: $\theta = \theta_w - \theta_i$. Once θ is determined, all the *sin* and *cos* elements are known: there are 3 equations and 3 unknowns which can easily be solved for s and x_0, y_0. The reader should complete this solution via the exercise below.

Exercise 6 converting image coordinates to workbench coordinates

Assume an environment as in Figure 11.5. (Perhaps the vision system must inform a pick-and-place robot of the locations of objects.) Give, in matrix form, the transformation that relates image coordinates $[x_i, y_i, 1]$ to workbench coordinates $[x_w, y_w, 1]$. Compute the four parameters using these control points: ${}^i\mathbf{P_1} = [100, 60]$, ${}^w\mathbf{P_1} = [200, 100]$; ${}^i\mathbf{P_2} = [380, 120]$, ${}^w\mathbf{P_2} = [300, 200]$.

An Example Affine Warp

It is easy to extract a parallelogram of data from a digital image by selecting 3 points. The first point determines the origin of the output image to be created, while the second and third points determine the extreme point of the parallelogram sides. The output image will be a rectangular pixel array of any size constructed from samples from the input image. Figure 11.6 shows results of using a program based on this idea. To create the center image of the figure, the three selected points defined nonorthogonal axes, thus creating shear in the output; this shear was removed by extracting a third image from the center image and aligning the new sampling axes with the skewed axes. Figure 11.7 shows another example: a distorted version of Andrew Jackson's head has been extracted from an image of a $20 bill (see Figure 11.24). In both of these examples, although only a portion of the input image was extracted, the output image contains the same number of pixels.

The program that created Figure 11.7 used the three user selected points to transform the input parallelogram defined at the left of the figure. The output image was n by m or 512x512 pixels with coordinates $[r, c]$; for each pixel $[r, c]$ of the output, the input image value was sampled at pixel $[x, y]$ computed using the transformation in Equation 11.9. The first form of the equation is the intuitive construction in terms of basis vectors, while the

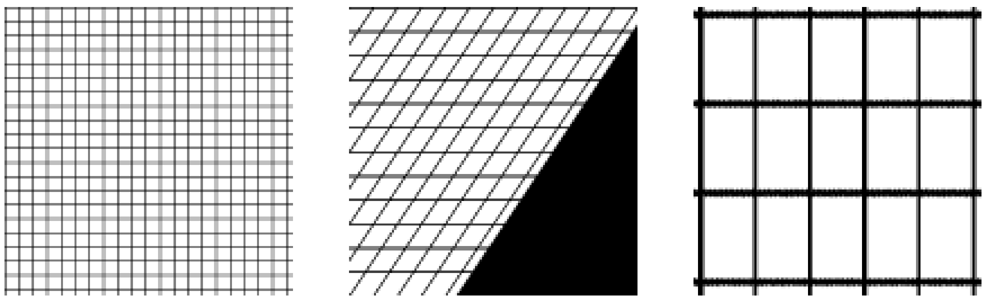


Figure 11.6: (Left) 128x128 digital image of grid; (center) 128x128 image extracted by an affine warp defined by three points in the left image; (right) 128x128 rectified version of part of the center image.

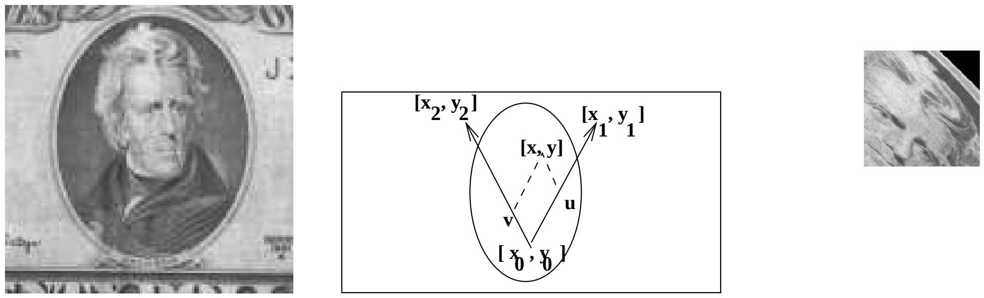


Figure 11.7: Distorted face of Andrew Jackson extracted from a $20 bill by defining an affine mapping with shear.

second is its equivalent in standard form.

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \frac{r}{n}\left(\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} - \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}\right) + \frac{c}{m}\left(\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} - \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}\right)$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} (x_1 - x_0)/n & (x_2 - x_0)/m & x_0 \\ (y_1 - y_0)/n & (y_2 - y_0)/m & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r \\ c \\ 1 \end{bmatrix} \tag{11.9}$$

Conceptually, the point $[x, y]$ is defined in terms of the new unit vectors along the new axes defined by the user selected points. The computed coordinates $[x, y]$ must be rounded to get integer pixel coordinates to access digital image ${}^1I$. If either x or y are out of bounds, then the output point is set to black, in this case ${}^2I[r, c] = 0$; otherwise ${}^2I[r, c] = {}^1I[x, y]$. One can see a black triangle at the upper right of Jackson's head because the sampling parallelogram protrudes above the input image of the $20 bill image.

Object Recognition/Location Example

Consider the example of computing the transformation matching the model of an object shown at the left in Figure 11.8 to the object in the image shown at the right of the figure. Assume that automatic feature extraction produced only three of the holes in the object. The spatial transformation will map points $[x, y]$ of the model to points $[u, v]$ of the image. Assume that we have a controlled imaging environment and the known scale factor has already been applied to the image coordinates to produce the u-v coordinates shown. Only two image points are needed in order to derive the rotation and translation that will align all model points with correpsonding image points. Point locations in the model and image and interpoint distances are shown in Tables 11.1 and 11.2. We will use the hypothesized correspondences (A, H_2) and (B, H_3) to deduce the transformation. Note that these correspondences are consistent with the known interpoint distances. We will discuss algorithms for making such hypotheses later on in this chapter.

Table 11.1: Model Point Locations and Interpoint Distances (†coordinates are for the center of holes).

point	coordinates †	to A	to B	to C	to D	to E
A	(8,17)	0	12	15	37	21
B	(16,26)	12	0	12	30	26
C	(23,16)	15	12	0	22	15
D	(45,20)	37	30	22	0	30
E	(22,1)	21	26	15	30	0

The direction of the vector from A to B in the model is $\theta_1 = \arctan(9.0/8.0) = 0.844$ and the heading of the corresponding vector from H_2 to H_3 in the image is $\theta_2 = \arctan(12.0/0.0) = \pi/2 = 1.571$. The rotation is thus $\theta = 0.727$ radians. Using Equations 11.6 and substituting the known matching coordinates for points A in the model and H_2 in the image, we obtain the following system, where u_0, v_0 are the unknown translation components in the image plane. Note that the values of $\sin(\theta)$ and $\cos(\theta)$ are actually known

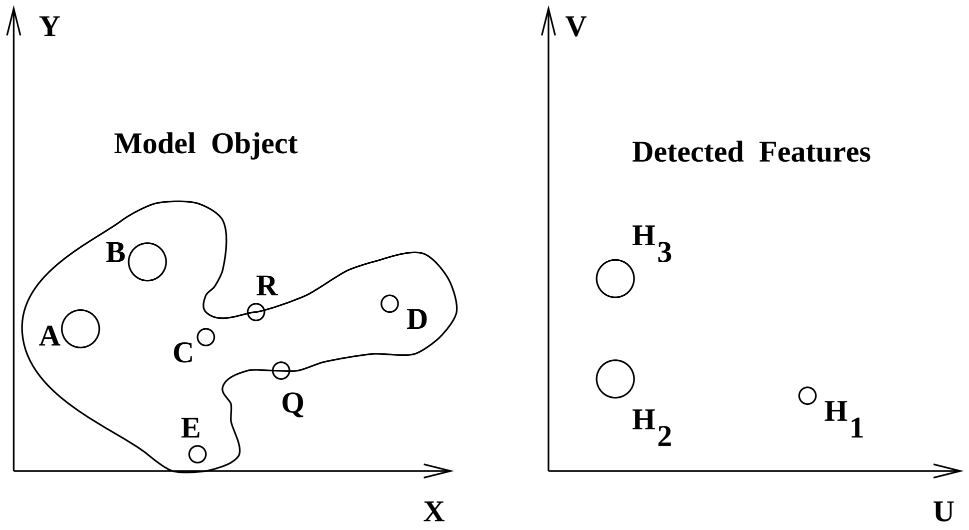


Figure 11.8: (Left) Model object and (right) three holes detected in an image.

Table 11.2: Image Point Locations and Interpoint Distances (†coordinates are for the center of holes).

point	coordinates †	to H1	to H2	to H3
H1	(31,9)	0	21	26
H2	(10,12)	21	0	12
H3	(10,24)	26	12	0

since θ has been computed.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 10 \\ 12 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & u_0 \\ \sin\theta & \cos\theta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 8 \\ 17 \\ 1 \end{bmatrix} \tag{11.10}$$

The two resulting linear equations readily produce $u_0 = 15.3$ and $v_0 = -5.95$. As a check, we can use the matching points B and H_3, obtaining a similar result. Each distinct pair of points will result in a slightly different transformation. Methods of using many more points to obtain a transformation that is more accurate across the entire 2D space are discussed later in this chapter. Having a complete spatial transformation, we can now compute the location of any model points in the image space, including the *grip points* $R = [29, 19]$ and $Q = [32, 12]$. As shown below, model point R transforms to image point ${}^iR = [24.4, 27.4]$: using $Q = [32, 12]$ as the input to the transformation outputs the image location $i^Q = [31.2, 24.2]$ for the other grip point.

$$\begin{bmatrix} u_R \\ v_R \\ 1 \end{bmatrix} = \begin{bmatrix} 24.4 \\ 27.4 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 15.3 \\ \sin\theta & \cos\theta & -5.95 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 29 \\ 19 \\ 1 \end{bmatrix} \tag{11.11}$$

Given this knowledge, a robot could grasp the real object being imaged, provided that it had knowledge of the transformation from image coordinates to coordinates of the workbench supporting the object. Of course, a robot gripper would be opened a little wider than the transformed length obtained from ${}^iR\,{}^iQ$ indicates, to allow for small effects such as distortions in imaging, inaccuracies of feature detection, and computational errors. The required gripping action takes place on a real continuous object and real number coordinates make sense despite the fact that the image holds only discrete spatial samples. The image data itself is only defined at integer grid points. If our action were to verify the presence of holes ${}^iC$ and ${}^iD$ by checking for a bright image pixel, then the transformed model points should be rounded to access an image pixel. Or, perhaps an entire digital neighborhood containing the real transformed coordinates should be examined. With this example, we have seen the potential for recognizing 2D objects by *aligning* a model of the object with important feature points in its image.

8 DEFINITION *Recognizing an object by matching transformed model features to image features via a rotation, scaling, and translation (RST) is called recognition-by-alignment.*

Exercise 7 are transformations commutative?

Suppose we have matrices for three primitive transformations: $\mathbf{R}_\theta$ for a rotation about the origin, $\mathbf{S}_{s_x,s_y}$ for a scaling, and $\mathbf{D}_{x_0,y_0}$ for a translation. (a) Do scaling and translation commute; that is, does $\mathbf{S}_{s_x,s_y}\ \mathbf{D}_{x_0,y_0} = \mathbf{D}_{x_0,y_0}\ \mathbf{S}_{s_x,s_y}$? (b) Do rotation and scaling commute; that is, does $\mathbf{R}_\theta\ \mathbf{S}_{s_x,s_y} = \mathbf{S}_{s_x,s_y}\ \mathbf{R}_\theta$? (c) Same question for rotation and translation. (d) Same question for scaling and translation. Do both the algebra and intuitive thinking to derive your answers and explanations.

Exercise 8

Construct the matrix for a reflection about the line y=3 by composing a translation with $y_0 = -3$ followed by a reflection about the x-axis. Verify that the matrix is correct by transforming the three points [1, 1], [1, 0], and [2, 1] and plotting the input and output points.

Exercise 9

Verify that the product of the matrices $\mathbf{D}_{x_0,y_0}$ and $\mathbf{D}_{-x_0,-y_0}$ is the 3x3 identity matrix. Explain why this should be the case.

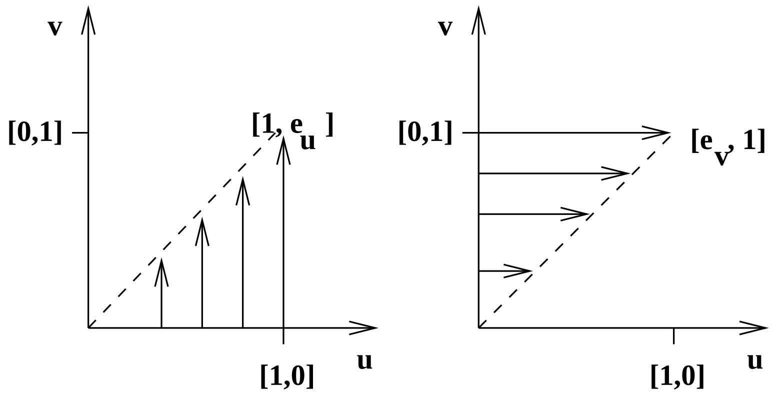


Figure 11.9: (Left) v-axis shear and (right) u-axis shear.

* General Affine Transformations

We have already covered affine transformation components of rotation, scaling, and translation. A fourth component is shear. Figure 11.9 shows the effect of shear. Using $u-v$ coordinates, point vectors move along the v-axis in proportion to their distance from the v-axis. The point $[u, v]$ is transformed to $[u, e_u u + v]$ with v-axis shear and to $[u + e_v v, v]$ with u-axis shear. The matrix equations are given in Equation 11.12 and Equation 11.13. Recall that the column vectors of the shear matrix are just the images of the basis vectors under the transformation.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ e_u & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{11.12}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & e_v & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{11.13}$$

Reflections are a fifth type of component. A reflection about the u-axis maps the basis vectors $[1, 0], [0, 1]$ onto $[1, 0], [0, -1]$ respectively, while a reflection about the v-axis maps $[1, 0], [0, 1]$ onto $[-1, 0], [0, 1]$. The 2x2 or 3x3 matrix representation is straightforward. Any affine transformation can be constructed as a composition of any of the component types – rotations, scaling, translation, shearing, or reflection. Inverses of these components exist and are of the same type. Thus, it is clear that the matrix of a general affine transformation composed of any components has six parameters as shown in Equation 11.14. These six parameters can be determined using 3 sets of noncolinear points known to be in correspondence by solving 3 matrix equations of this type. We have already seen an example using shear in the case of the slanted grid in Figure 11.6.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{11.14}$$

11.4 * A Best 2D Affine Transformation

A general affine transformation from 2D to 2D as in Equation 11.15 requires six parameters and can be computed from only 3 matching pairs of points $([x_j, y_j], [u_j, v_j])_{j=1,3})$.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{11.15}$$

Error in any of the coordinates of any of the points will surely cause error in the transformation parameters. A much better approach is to use many more pairs of matching control points to determine a least-squares estimate of the six parameters. We can define an error criteria function similar to that used for fitting a straight line in Chapter 10.

$$\varepsilon(a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}) = \sum_{j=1}^{n}((a_{11}x_j + a_{12}y_j + a_{13} - u_j)^2 + (a_{21}x_j + a_{22}y_j + a_{23} - v_j)^2) \tag{11.16}$$

Taking the six partial derivatives of the error function with respect to each of the six variables and setting this expression to zero gives us the six equations represented in matrix form in Equation 11.17 .

$$\begin{bmatrix} \Sigma {x_j}^2 & \Sigma x_j y_j & \Sigma x_j & 0 & 0 & 0 \\ \Sigma x_j y_j & \Sigma y_j^2 & \Sigma y_j & 0 & 0 & 0 \\ \Sigma x_j & \Sigma y_j & \Sigma 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \Sigma {x_j}^2 & \Sigma x_j y_j & \Sigma x_j \\ 0 & 0 & 0 & \Sigma x_j y_j & \Sigma y_j^2 & \Sigma y_j \\ 0 & 0 & 0 & \Sigma x_j & \Sigma y_j & \Sigma 1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \end{bmatrix} = \begin{bmatrix} \Sigma u_j x_j \\ \Sigma u_j y_j \\ \Sigma u_j \\ \Sigma v_j x_j \\ \Sigma v_j y_j \\ \Sigma v_j \end{bmatrix} \tag{11.17}$$

Exercise 10

Solve Equation 11.17 using the following three pairs of matching control points: ([0,0],[0,0]), ([1,0],[0,2]), ([0,1],[0,2]). Do your computations give the same answer as reasoning about the transformation using basis vectors?

Exercise 11

Solve Equation 11.17 using the following three pairs of matching control points: ([0,0],[1,2]), ([1,0],[3,2]), ([0,1],[1,4]). Do your computations give the same answer as reasoning about the transformation using basis vectors?

It is common to use many control points to put an image and map or two images into correspondence. Figure 11.10 shows two images of approximately the same scene. Eleven pairs of matching control points are given at the bottom of the figure. Control points are corners of objects that are uniquely identifiable in both images (or map). In this case, the control points were selected using a display program and mouse. The list of residuals shows that, using the derived transformation matrix, no u or v coordinate in the right image will be off by 2 pixels from the transformed value. Most residuals are less than one pixel. Better results can be obtained by using automatic feature detection which locates feature points with subpixel accuracy: control point coordinates are often off by one pixel when chosen using a computer mouse and the human eye. Using the derived affine transformation, the right image can be searched for objects known to be in the left image. Thus we have reached the point of understanding how the tax-collectors map can be put into correspondence with an aerial image for the purpose of updating its inventory of objects.

Exercise 12

Take three pairs of matching control points from Figure 11.10 (for example, ([288, 210, 1], [31, 160, 1])) and verify that the affine tranformation matrix maps the first into the second.

11.5 2D Object Recognition via Affine Mapping

In this section we study a few methods of recognizing 2D objects through mappings of model points onto image points. We have already introduced one method of recognition-by-alignment in the section on affine mappings. The general methods work with general point features; however, each application domain will present distinguishing features which allow labels to be attached to feature points. Thus we might have corners or centers of holes in a part sorting application, or intersections and high curvature land/water boundary points in a land use application.

Figure 11.11 illustrates the overall model-matching paradigm. Figure 11.11a) is a boundary model of an airplane part. Feature points that may be used in matching are indicated with small black circles. Figure 11.11b) is an image of the real airplane part in approximately the same orientation as the model. Figure 11.11c) is a second image of the real part rotated about 45 degrees. Figure 11.11d) is a third image of the real part in which the camera angle causes a large amount of skewing in the resultant image. The methods described in this section are meant to determine if a given image, such as those of Figures 11.11b), 11.11c), and 11.11d) contains an instance of an object model such as that of Figure 11.11a) and to determine the *pose* (position and orientation) of the object with respect to the camera.

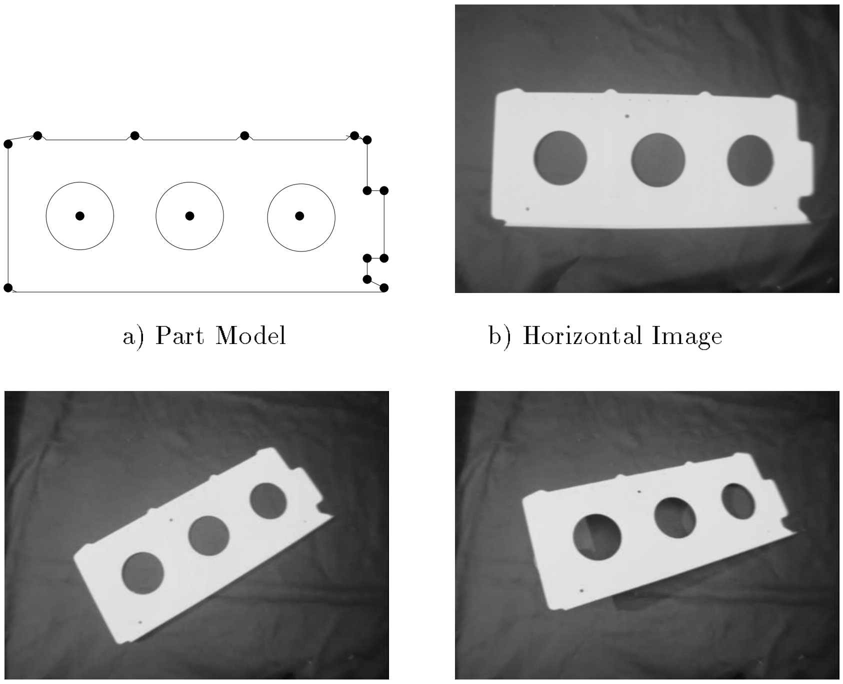
Local Feature Focus Method

The local-feature-focus method uses local features of an object and their 2D spatial relationships to recognize the object. In advance, a set of object models is constructed, one for each object to be recognized. Each object model contains a set of *focus features*, which are major features of the object that should be easily detected if they are not occluded by some other object. For each focus feature, a set of its nearby features is included in the model. The nearby features can be used to verify that the correct focus feature has been found and to help determine the pose of the object.

In the matching phase, feature extraction is performed on an image of one or more objects. The matching algorithm looks first for focus features. When it finds a focus feature belonging to a given model, it looks for a cluster of image features near the focus feature that match as many as possible of the required nearby features for that focus feature in that model. Once such a cluster has been found and the correspondences between this small set of image features and object model features have been determined, the algorithm

```
======Best 2D Affine Fit Program======
Matching control point pairs are:
288 210  31 160    232 288  95 205    195 372 161 229    269 314 112 159
203 424 199 209    230 336 130 196    284 401 180 124    327 428 198  69
284 299 100 146    337 231  45 101    369 223  38  64

The Transformation Matrix is:

[ -0.0414 ,  0.773 , -119
  -1.120  , -0.213 ,  526  ]

 Residuals (in pixels) for 22 equations are as follows:

 0.18  -0.68  -1.22  0.47  -0.77  0.06  0.34  -0.51  1.09  0.04  0.96
 1.51  -1.04  -0.81  0.05   0.27  0.13 -1.12   0.39 -1.04 -0.12  1.81

======Fitting Program Complete======
```

Figure 11.10: Images of same scene and best affine mapping from the left image into the right image using 11 control points. [x,y] coordinates for the left image with x increasing downward and y increasing to the right; [u,v] coordinates for the right image with u increasing downward and v toward the right. The 11 clusters of coordinates directly below the images are the matching control points x,y,u,v.



a) Part Model b) Horizontal Image

c) Rotated Image d) Rotated and Skewed Image

Figure 11.11: A 2D model and 3 matching images of an airplane part.

hypothesizes that this object is in the image and uses a verification technique to decide if the hypothesis is correct.

The verification procedure must determine whether there is enough evidence in the image that the hypothesized object is in the scene. For polyhedral objects, the boundary of the object is often used as suitable evidence. The set of feature correspondences is used to determine a possible affine transformation from the model points to the image points. This transformation is then used to transform each line segment of the boundary into the image space. The transformed line segments should approximately line up with image line segments wherever the object is unoccluded. Due to noise in the image and errors in feature extraction and matching, it is unlikely that the transformed line segments will exactly align with image line segments, but a rectangular area about each transformed line segment can be searched for evidence of possibly matching image segments. If sufficient evidence is found, that model segment is marked as verified. If enough model segments are verified, the object is declared to be in the image at the location specified by the computed transformation.

The local-feature-focus algorithm for matching a given model F to an image is given below. The model has a set $\{F_1, F_2, \ldots, F_M\}$ of focus features. For each focus feature F_m, there is a set $S(F_m)$ of nearby features that can help to verify this focus feature. The image has a set $\{G_1, G_2, \ldots, G_I\}$ of detected image features. For each image feature G_i, there is a set of nearby image features $S(G_i)$.

Find the transformation from model features to image features
using the local-feature-focus approach.
$G_i, i = 1, I$ is the set of the detected image features.
$F_m, m = 1, M$ is the set of focus features of the model.
$S(f)$ is the set of nearby features for any feature f.

procedure local_feature_focus(G,F);
{
for each focus feature F_m
  for each image feature G_i of the same type as F_m
    {
      Find the maximal subgraph S_m of $S(F_m)$ that
        matches a subgraph S_i of $S(G_i)$;
      Compute the transformation T that maps the points of
        each feature of S_m to the corresponding feature of S_i;
      Apply T to the boundary segments of the model;
      if enough of the transformed boundary segments find
        evidence in the image **then return**(T);
    } ;
}

Algorithm 1: Local-Feature-Focus Method

Figure 11.12 illustrates the local-feature-focus method with two models, E and F, and an image. The detected features are circular holes and sharp corners. Local feature F1 of model F has been hypothesized to correspond to feature G1 in the image. Nearby features F2, F3, and F4 of the model have been found to correspond well to nearby features G2, G3, and G4, respectively, of the image. The verification step will show that model F is indeed in the image. Considering the other model E, feature E1 and the set of nearby features E2, E3, and E4 have been hypothesized to correspond to features G5, G6, G7, and G8 in the image. However, when verification is performed, the boundary of model E will not line up well with the image segments, and this hypothesis will be discarded.

Pose Clustering

We have seen that an alignment between model and image features using an RST transformation can be obtained from two matching control points. The solution can be obtained using Equations 11.6 once two control points have been matched between image and model. Obtaining the matching control points automatically may not be easy due to ambiguous matching possibilities. The pose clustering approach computes an RST alignment for all possible control point pairs and then checks for a cluster of similar parameter sets. If indeed there are many matching feature points between model and image, then a cluster should

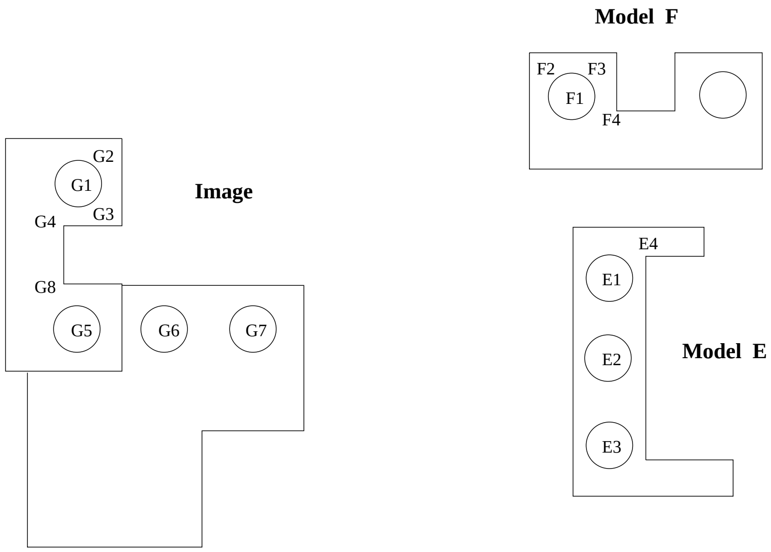


Figure 11.12: The Local Feature Focus Method

exist in the parameter space. A pose-clustering algorithm is sketched below.

9 DEFINITION *Let T be a spatial transformation aligning model M to an object O in image I. The pose of object O is its location and orientation as defined by the parameters α of T.*

Using all possible pairs of feature points would provide too much redundancy. An application in matching aerial images to maps can use intersection points detected on road networks or at the corners of regions such as fields. The degree of the intersection gives it a type to be used in matching; for example, common intersections have type 'L','Y','T', 'Arrow', and 'X'. Assume that we use only the pairs of combined type LX or TY. Figure 11.14 shows an example with 5 model pairs and 4 image pairs. Although there are $4 \times 5 = 20$

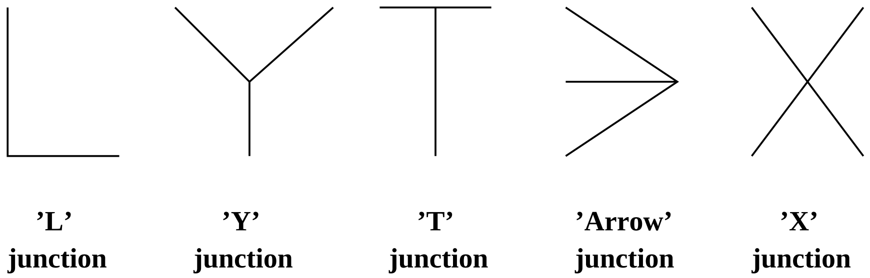


Figure 11.13: Common line-segment junctions used in matching.

Find the transformation from model features to image features
using pose clustering.
$P_i, i = 1, D$ is the set of detected image features.
$L_j, j = 1, M$ is the set of stored model features.

```
procedure pose_clustering(P,L);
{
for each pair of image feature points (P_i, P_j)
   for each pair of model feature points (L_m, L_n) of same type
      {
         compute parameters α of RST mapping
            pair (L_m, L_n) onto (P_i, P_j);
         contribute α to the cluster space;
      } ;
examine space of all candidates α for clusters;
verify every large cluster by mapping all
   model feature points and checking the image;
return(verified {α_k });
}
```

Algorithm 2: Pose-Clustering Algorithm

possible pairings, only 10 of them have matching types for both points. The transformations computed from each of those are given in Table 11.3. The 10 transformations computed have scattered and inconsistent parameter sets, except for 3 of them indicated by '* in the last column of the table. These 3 parameter sets form a cluster whose average parameters are $\theta = 0.68, s = 2.01, u_0 = 233, v_0 = -41$. While one would like less variance than this for correct matches, this variance is typical due to slight errors in point feature location and small nonlinear distortions in the imaging process. If the parameters of the RST mapping are inaccurate, they can be used to verify matching points which can then be used as control points to find a nonlinear mapping or affine mapping (with more parameters) that is more accurate in matching control points.

Pose clustering can work using low level features; however, both accuracy and efficiency are improved when features can be filtered by type. Clustering can be performed by a simple $O(n^2)$ algorithm: for each parameter set α, count the number of other parameter sets α_i that are close to it using some permissible distance. This requires $n - 1$ distance computations for each of the n parameter sets in cluster space. A faster, but less flexible, alternative is to use binning. Binning has been the traditional approach reported in the literature and was discussed in Chapter 10 with respect to the Hough transform. Each parameter set produced is contributed to a bin in parameter space, after which all bins are examined for significant counts. A cluster can be lost when a set of similar α_i spread over neighboring bins.

The clustering approach has been used to detect the presence of a particular model of airplane from an aerial image, as shown in Figure 11.15. Edge and curvature features are extracted from the image using the methods of Chapters 5 and 10. Various overlapping

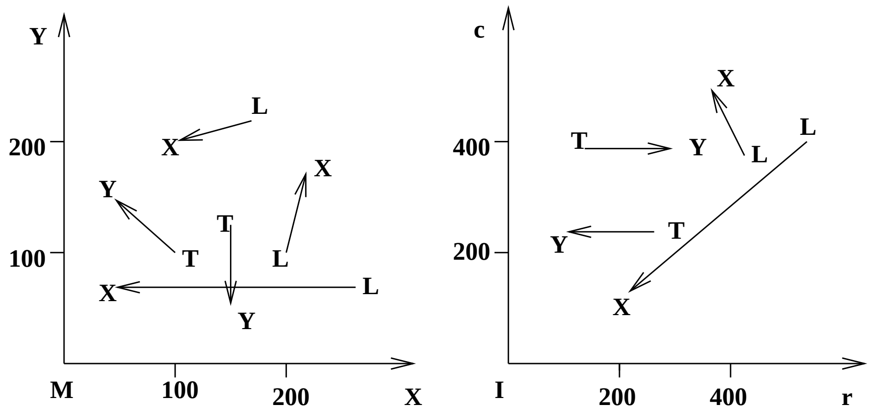
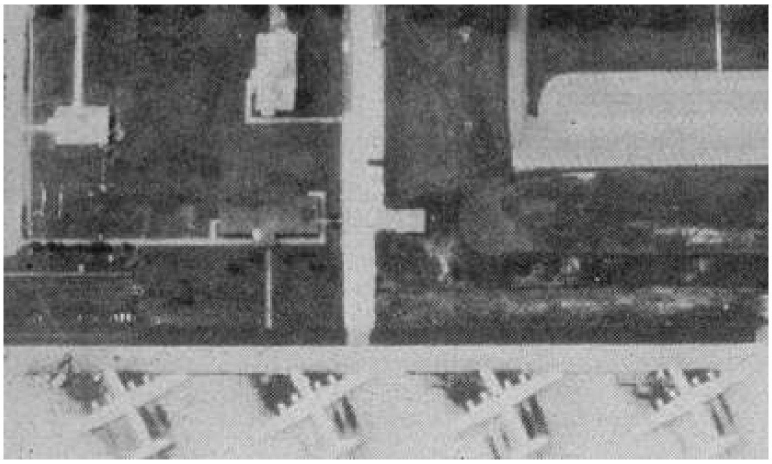


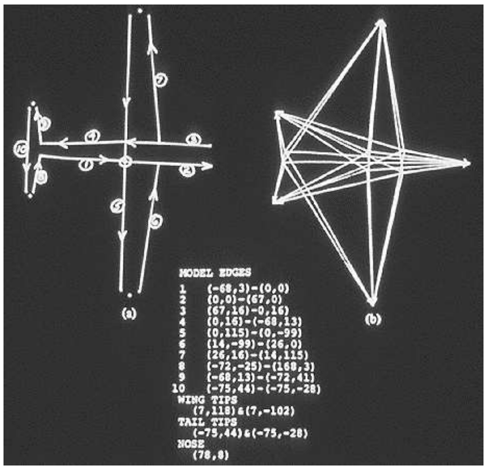
Figure 11.14: Example pose detection problem with 5 model feature point pairs and 4 image feature point pairs.

Table 11.3: Cluster space formed from 10 pose computations from Figure 11.14

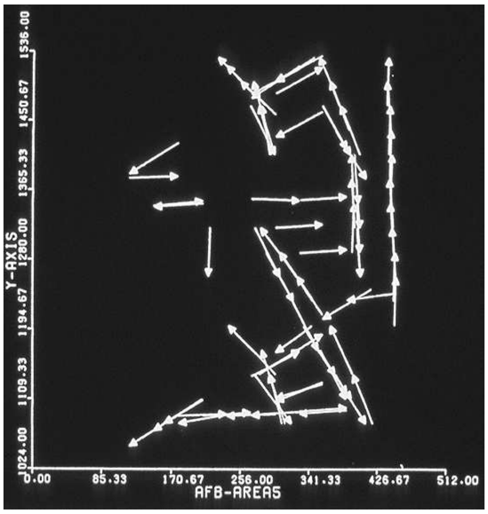
Model Pair	Image Pair	θ	s	u_0	v_0	
L(170,220),X(100,200)	L(545,400),X(200,120)	0.403	6.10	118	-1240	
L(170,220),X(100,200)	L(420,370),X(360,500)	5.14	2.05	-97	514	
T(100,100),Y(40,150)	T(260,240),Y(100,245)	0.663	2.05	225	-48	*
T(100,100),Y(40,150)	T(140,380),Y(300,380)	3.87	2.05	166	669	
L(200,100),X(220,170)	L(545,400),X(200,120)	2.53	6.10	1895	200	
L(200,100),X(220,170)	L(420,370),X(360,500)	0.711	1.97	250	-36	*
L(260, 70),X(40, 70)	L(545,400),X(200,120)	0.682	2.02	226	-41	*
L(260, 70),X(40, 70)	L(420,370),X(360,500)	5.14	0.651	308	505	
T(150,125),Y(150, 50)	T(260,240),Y(100,245)	4.68	2.13	3	568	
T(150,125),Y(150, 50)	T(140,380),Y(300,380)	1.57	2.13	407	60	



(a) original airfield image



(b) model of object



(c) detections matching model

Figure 11.15: Pose-clustering applied to detection of a particular airplane. (a) Aerial image of an airfield; (b) object model in terms of real edges and abstract edges subtending one corner and one curve tip point; (c) image window containing detections that match many model parts via the same transformation.

windows of these features are matched against the model shown in part (b) of the figure. Part (c) of the figure shows the edges detected in one of these windows where many of the features aligned with the model features using the same transformation parameters.

Geometric Hashing

Both the local-feature-focus method and the pose clustering algorithm were designed to match a single model to an image. If several different object models were possible, then these methods would try each model, one at a time. This makes them less suited for problems in which a large number of different objects can appear. Geometric hashing was designed to work with a large database of models. It trades a large amount of offline preprocessing and a large amount of space for a potentially fast online object recognition and pose determination.

Suppose we are given

1. a large database of models
2. an unknown object whose features are extracted from an image and which is known to be an affine transformation of one of the models.

and we wish to determine which model it is and what transformation was applied.

Consider a model M to be an ordered set of feature points. Any subset of three non-collinear points $E = \{e_{00},\ e_{01},\ e_{10}\}$ of M can be used to form an affine basis set, which defines a coordinate system on M, as shown in Figure 11.16a). Once the coordinate system is chosen, any point $x \in M$ can be represented in affine coordinates (ξ, η) where

$$x = \xi(e_{10} - e_{00}) + \eta(e_{01} - e_{00}) + e_{00}$$

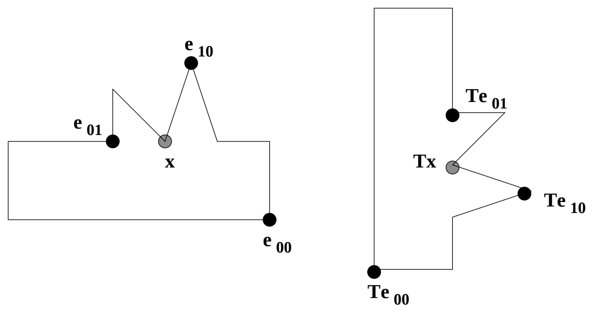
Furthermore, if we apply an affine transform T to point x, we get

$$Tx = \xi(Te_{10} - Te_{00}) + \eta(Te_{01} - Te_{00}) + Te_{00}$$

Thus Tx has the same affine coordinates (ξ, η) with respect to $(Te_{00}, Te_{01}, Te_{10})$ as x has with respect to (e_{00}, e_{01}, e_{10}). This is illustrated in Figure 11.16b).

Offline Preprocessing: The offline preprocessing step creates a hash table containing all of the models in the database. The hash table is set up so that the pair of affine coordinates (ξ, η) indexes a bin of the hash table that stores a list of model-basis pairs (M, E) where some point x of model M has affine coordinates (ξ, η) with respect to basis E. The offline preprocessing algorithm is given below.

Online Recognition: The hash table created in the preprocessing step is used in the online recognition step. The recognition step also uses an accumulator array A indexed by model-basis pairs. The bin for each (M, E) is initialized to zero and used to vote for the hypothesis that there is a transformation T that places (M, E) in the image. Computation of the actual transformations is done only for those model-basis pairs that achieve a high number of votes and is part of the verification step that follows the voting. The online



a) original object b) transformed object

Figure 11.16: The affine transformation of a point with respect to an affine basis set.

recognition and pose estimation algorithm is given below.

Suppose that there are s models of approximately n points each. Then the preprocessing step has complexity $O(sn^4)$ which comes from processing s models, $O(n^3)$ triples per model, and $O(n)$ other points per model. In the matching, the amount of work done depends somewhat on how well the feature points can be found in the image, how many of them are occluded, and how many false or extra feature points are detected. In the best case, the first triple selected consists of three real feature points all from the same model, this model gets a large number of votes, the verification procedure succeeds, and the task is done. In this best case, assuming the average list in the hash table is of a small constant size and that hashing time is approximately constant, the complexity of the matching step is approximately $O(n)$. In the worst case, for instance when the model is not in the database at all, every triple is tried, and the complexity is $O(n^4)$. In practice, although it would be rare to try all bases, it is also rare for only one basis to succeed. A number of different things can go wrong:

1. feature point coordinates have some error
2. missing and extra feature points
3. occlusion, multiple objects
4. unstable bases
5. weird affine transforms on a subset of the points

In particular, the algorithm can "halucinate" a transformation based on a subset of the points that passes the point verification tests, but gives the wrong answer. Figure 11.17 illustrates this point. Pose clustering and focus feature methods are also susceptible to this same phenomenon.

Set up the hash table for matching to a database of models
using geometric hashing.
D is the database of models.
H is an initially empty hash table.

```
procedure GH_Preprocessing(D,H);
{
for each model M
  {
  Extract the feature point set F_M of M;
  for each noncollinear triple E of points from F_M
    for each other point x of F_M
      {
      Calculate (ξ, η) for x with respect to E;
      Store (M, E) in hash table H at index (ξ, η);
      } ;
  } ;
}
```

Algorithm 3: Geometric Hashing Offline Preprocessing

11.6 2D Object Recognition via Relational Matching

We have previously described three useful methods for matching observed image points to model points: these were local-feature-focus, pose clustering, and geometric hashing. In this section, we examine three simple general paradigms for object recognition within the context given in this chapter. All three paradigms view recognition as a mapping from model structures to image structures: a *consistent labeling* of image features is sought in terms of model features, recognition is equivalent to mapping a sufficient number of features from a single model to the observed image features. The three paradigms differ in how the mapping is developed.

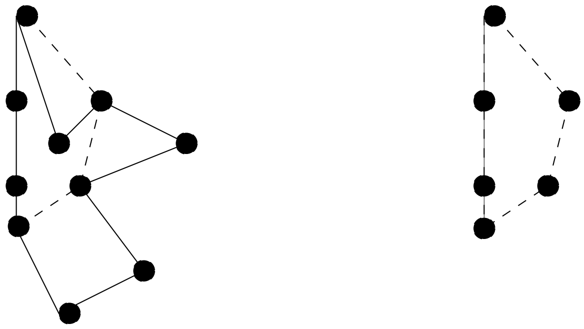
Four concepts important to the matching paradigms are *parts, labels, assignments, and relations.*

- A **part** is an object or structure in the scene such as region segment, edge segment, hole, corner or blob.
- A **label** is a symbol assigned to a part to identify/recognize it at some level.
- An **assignment** is a mapping from parts to labels. If P_1 is a region segment and L_1 is the lake symbol and L_2 the field symbol, an assignment may include the pair (P_1, L_2) or perhaps $(P_1, \{L_1, L_2\})$ to indicate remaining ambiguity. A pairing (P_1, NIL) indicates that P_1 has no interpretation in the current label set. An interpretation of the scene is just the set of all pairs making up an assignment.
- A **relation** is the formal mathematical notion. Relations will be discovered and computed among scene objects and will be stored for model objects. For example, $R4(P_1, P_2)$ might indicate that region P_1 is *adjacent to* region P_2

Use the hash table to find the correct model and transformation
that maps image features to model features.
H is the hash table created by the preprocessing step.
A is an accumulator array indexed by (M, E) pairs.
I is the image being analyzed.

```
procedure GH_Recognition(H,A,I);
{
Initialize accumulator array A to all zeroes;
Extract feature points from image I;
for each basis triple F
  {
  for each other point v
    {
    Calculate (ξ, η) for v with respect to F;
    Retrieve the list L of model-basis pairs from the
      hash table H at index (ξ, η);
    for each pair (M, E) of L
      A[M, E] = A[M, E] + 1;
    } ;
  Find the peaks in accumulator array A;
  for each peak (M, E)
    {
    Calculate T such at F = TE;
    if enough of the transformed model points of M find
      evidence on the image then return(T);
    } ;
  } ;
}
```

Algorithm 4: Geometric Hashing Online Recognition



a) image points b) halucinated object

Figure 11.17: The geometric hashing algorithm can halucinate that a given model is present in an image. In this example, 60% of the feature points (left) led to the verified hypothesis of an object (right) that was not actually present in the image.

Given these four concepts, we can define a consistent labeling.

10 Definition *Given a set of parts P, a set of labels for those parts L, a relation R_P over P, and a second relation R_L over L, a consistent labeling f is an assignment of labels to parts that satisfies:*

If $(p_i, p_{i'}) \in R_P$, then $(f(p_i), f(p_{i'})) \in R_L$.

For example, suppose we are trying to find a match between two images: for each image we have a set of extracted line segments and a *connection* relation that indicates which pairs of line segments are connected. Let P be the set of line segments and R_P be the set of pairs of connecting line segments from the first image, $R_P \subseteq P \times P$. Similarly, let L be the set of line segments and R_L be the set of pairs of connecting line segments from the second image, $R_L \subseteq L \times L$. Figure 11.18 illustrates two sample images and the sets P, R_P, L, and R_L. Note that both R_P and R_L are symmetric relations; if (Si, Sj) belongs to such a relation, then so does (Sj, Si). In our examples, we list only tuples (Si, Sj) where $i < j$, but the mirror image tuple (Sj, Si) is implicitly present.

A consistent labeling for this problem is the mapping f given below:

f(S1)	=	Sj	f(S7)	=	Sg
f(S2)	=	Sa	f(S8)	=	Sl
f(S3)	=	Sb	f(S9)	=	Sd
f(S4)	=	Sn	f(S10)	=	Sf
f(S5)	=	Si	f(S11)	=	Sh
f(S6)	=	Sk			

For another example, we return to the recognition of the kleep object shown in Figure 11.8 and defined in the associated tables. Our matching paradigms will use the distance

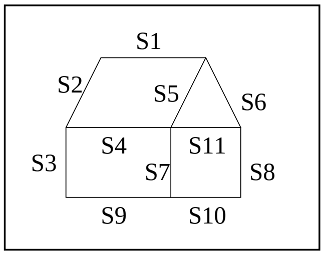


Image 1

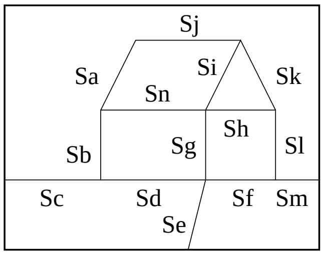


Image 2

P = {S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11}.

L = {Sa,Sb,Sc,Sd,Se,Sf,Sg,Sh,Si,Sj,Sk,Sl,Sm}.

R_P = { (S1,S2), (S1,S5), (S1,S6), (S2,S3), (S2,S4), (S3,S4), (S3,S9), (S4,S5), (S4,S7), (S4,S11), (S5,S6), (S5,S7), (S5,S11), (S6,S8), (S6,S11), (S7,S9), (S7,S10), (S7,S11), (S8,S10), (S8,S11), (S9,S10) }.

R_L = { (Sa,Sb), (Sa,Sj), (Sa,Sn), (Sb,Sc), (Sb,Sd), (Sb,Sn), (Sc,Sd), (Sd,Se), (Sd,Sf), (Sd,Sg), (Se,Sf), (Se,Sg), (Sf,Sg), (Sf,Sl), (Sf,Sm), (Sg,Sh), (Sg,Si), (Sg,Sn), (Sh,Si), (Sh,Sk), (Sh,Sl), (Sh,Sn), (Si,Sj), (Si,Sk), (Si,Sn), (Sj,Sk), (Sk,Sl), (Sl,Sm) }.

Figure 11.18: Example of a Consistent Labeling Problem

Exercise 13 Consistent Labeling Problem

Show that the labeling f given above is a consistent labeling. Because the relations are symmetric, the following modified constraint must be satisfied:

If $(p_i, p_{i'}) \in R_P$, then $(f(p_i), f(p_{i'})) \in R_L$ or $(f(p_{i'}), f(p_i)) \in R_L$.

"relation" defined for any two points: each pair of holes is related by the distance between them. Distance is invariant to rotations and translations but not scale change. Abusing notation, we write $12(A, B)$ and $12(B, C)$ to indicate that points A and B are distance 12 apart in the model, similarly for points B and C. $12(C, D)$ does NOT hold, however, as we see from the distance tables. To allow for some distortion or detection error, we might allow that $12(C, D)$ is true even when the distance between C and D is actually $12 + / - \Delta$ for some small amount Δ.

The Interpretation Tree

11 Definition *An interpretation tree (IT) is a tree that represents all possible assignments of labels to parts. Every path in the tree is terminated either because it represents a complete consistent assignment, or because the partial assignment it represents fails some relation.*

A partial interpretation tree for the image data of Figure 11.8 is shown in Figure 11.19. The tree has three levels, each to assign a label to one of the three holes H_1, H_2, H_3 ob-

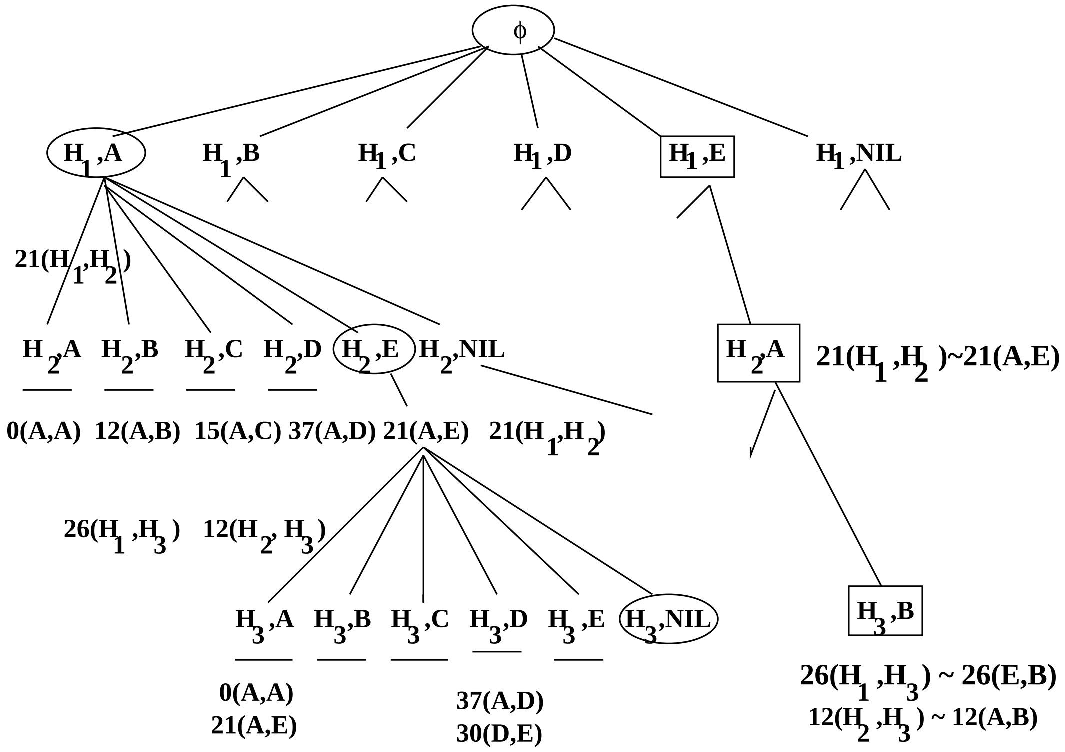


Figure 11.19: Partial interpretation tree search for a consistent labeling of the kleep parts in Figure 11.8 (right).

served in the image. No inconsistencies occur at the first level since there are no distance constraints to check. However, most label possibilities at level 2 can be immediately terminated using one distance check. For example, the partial assignment $\{(H_1, A), (H_2, A)\}$ is inconsistent because the relation $21(H_1, H_2)$ is violated by $0(A, A)$. Many paths are not shown due to lack of space. The path of labels denoted by the boxes yields a complete and consistent assignment. The path of labels denoted by ellipses is also consistent; however, it contains one NIL label and thus has fewer constraints to check. This assignment has the first two pairs of the complete (boxed) assignment reversed in labels and the single distance check is consistent. Multiple paths of an IT can succeed due to symmetry. Although the IT potentially contains an exponential number of paths, it has been shown that most paths will terminate by level 3 due to the relational constraints. Use of the label NIL allows for detection of artifacts or the presence of features from another object in the scene.

The IT can easily be developed using a recursive backtracking process that develops paths in a depth-first fashion. At any instantiation of the procedure, the parameter f, which is initially NIL, contains the consistent partial assignment. Whenever a new labeling of a part is consistent with the partial assignment, the algorithm goes deeper in the tree by hypothesizing another label for an unlabeled part; if an inconsistency is detected, then the algorithm backs up to make an alternate choice. As coded, the algorithm returns the first completed path, which may include NIL labels if that label is explicitly included in L.

An improvement would be to return the completed path with the most nonNIL pairs, or perhaps all completed paths.

The recursive interpretation tree search algorithm is given below. It is defined to be general and to handle arbitary N-ary relations, R_P and R_L, rather than just binary relations. R_P and R_L can be single relations, such as the connection relation in our first example, or they can be unions of a number of different relations, such as connection, parallel, and distance.

Find the mapping from model features to image features that
satisfies the model relationships by a tree search.
P is the set of detected image features.
L is the set of stored model features.
R_P is the relationship over the image features.
R_L is the relationship over the model features.
f is the consistent labeling to be returned, initially NIL.

```
procedure Interpretation_Tree_Search(P, L, R_P, R_L, f);
{
p := first(P);
for each l in L
  {
  f' = f ∪ {(p, l)}; /* add part-label to interpretation */
  OK = true;
  for each N-tuple (p_1, ..., p_N) in R_P containing component p
    and whose other components are all in domain(f)
    /* check on relations */
    if (f(p_1), ..., f(p_N)) is not in R_L then
      {
      OK: = false;
      break;
      }
  if OK then
    {
    P' = rest(P);
    if isempty(P') then output(f');
    else Interpretation_Tree_Search(P', L, R_P, R_L, f');
    }
  }
}
```

Algorithm 5: Interpretation Tree Search

Discrete Relaxation

Relaxation uses only local constraints rather than all the constraints available; for example, all constraints from matches on a single path of the **IT**. After N iterations, local constraints from one part neighborhood can propagate across the object to another part on a path N edges distant. Although the constraints used in one iteration are weaker than those available using the **IT** search, they can be applied in parallel, thus making faster and simpler processing possible.

Initially, a part can be labeled by any label permitted by its type; suppose we assign it a set of all these possible labels. Discrete relaxation examines the relations between a particular part and all others, and by doing so, reduces the possible labels for that particular part. In the related problem of character recognition, if it is known that the following letter cannot be "U", then we can conclude that the current letter cannot be "Q". In yet a different domain, if it is known that an image region is not water, then an object in it is not a ship. Discrete relaxation was popularized by David Waltz, who used it to constrain the labels assigned to edges of line drawings. (Waltz filtering is discussed in the text by Winston.) Waltz used an algorithm with some sequential character; here we present a parallel approach.

Each part P_i is initially assigned the entire **set** of possible labels L_j according to its type. Then, all relations are checked to see if some labels are impossible: inconsistent labels are removed from the set. The label sets for each part can be processed in parallel through passes. If, after any pass some labels have been filtered out of some sets, then another pass is executed; if no labels have changed, then the filtering is completed. There may be no interpretations left possible, or there may be several. The following example should be instructive. To keep it simple, we assume that there are no extra features detected that are not actual parts of the model; as before, we assume that some features may have been missed.

We now match the data in Tables 11.1 and 11.2. The filtering process begins with all 5 labels possible for each of the 3 holes H_1, H_2, H_3. To add interest and to be more practical, *we will allow a tolerance of* ± 1 *on distance matches.* Table 11.4 shows the 3 label sets at some point midway through the first pass. Each cell of the table gives the reason why a label must be deleted or why it survives. A is deleted from the label set for H_1 because the relation $26(H_1, H_3)$ cannot be explained by any label for H_3. The label A survives for H_2 because there is label $E \in L(H_1)$ to explain the relation $21(H_2, H_1)$ and label $B \in L(H_3)$ to explain relation $12(H_2, H_3)$. The label C survives for H_2, because $d(H_2, H_1) = 21 \approx 22 = d(C, D)$.

At the end of the first pass, as Table 11.5 shows, there are only two labels possible for H_2, only label E remains for H_1, and only label B remains for H_3. At the end of pass 1 the reduced label sets are made available for the parallel processing of pass 2, where each label set is further filtered in asynchronous parallel order.

Exercise 14

Give detailed justification for each of the labels being in or out of each of the label sets after pass 1 as shown in Table 11.5.

Pass 2 deletes label C from $L(H_2)$ because the relation $21(H_1, H_2)$ can no longer be explained by using D as a label for H_1. After pass 3, additional passes cannot change any label set so the process has converged. In this case, the label sets are all singletons repre-

Table 11.4: Midway through first pass of relaxation labeling

	A	B	C	D	E
H_1	no $N \ni$ $d(A,N) = 26$	no $N \ni$ $d(B,N) = 21$	no $N \ni$ $d(C,N) = 26$	no $N \ni$ $d(D,N) = 26$	$21(H_1,H_2)$ $A \in L(H_2)$ $26(H_1,H_3)$ $B \in L(H_3)$
H_2	$21(H_2,H_1)$ $E \in L(H_1)$ $12(H_2,H_3)$ $B \in L(H_3)$	no $N \ni$ $d(B,N) = 21$	$21(H_2,H_1)$ $D \in L(H_1)$ $12(H_2,H_3)$ $B \in L(H_3)$		
H_3	no $N \ni$ $d(A,N) = 26$	$12(H_3,H_2)$ $A \in L(H_2)$ $26(H_3,H_1)$ $E \in L(H_1)$			

Table 11.5: After completion of the first pass of relaxation labeling

	A	B	C	D	E
H_1	no	no	no	no	possible
H_2	possible	no	possible	no	no
H_3	no	possible	no	no	no

senting a single assignment and interpretation. A high-level sketch of the algorithm is given below. Although a simple and potentially fast procedure, relaxation labeling sometimes leaves more ambiguity in the interpretation than does IT search because constraints are only applied pairwise. Relaxation labeling can be applied as preprocessing for IT search: it can substantially reduce the branching factor of the tree search.

Continuous Relaxation

In exact consistent labeling procedures, such as tree search and discrete relaxation, a label l for a part p is either possible or impossible at any stage of the process. As soon as a part–label pair (p, l) is found to be incompatible with some already instantiated pair, the label l is marked as illegal for part p. This property of calling a label either possible or impossible in the preceding algorithms makes them *discrete* algorithms. In contrast, we can associate with each part–label pair (p, l) a real number representing the probability or

Table 11.6: After completion of the second pass of relaxation labeling

	A	B	C	D	E
H_1	no	no	no	no	possible
H_2	possible	no	no	no	no
H_3	no	possible	no	no	no

Table 11.7: After completion of the third pass of relaxation labeling

	A	B	C	D	E
H_1	no	no	no	no	possible
H_2	possible	no	no	no	no
H_3	no	possible	no	no	no

Remove incompatible labels from the possible labels for a set of detected image features.
$P_i, i = 1, D$ is the set of detected image features.
$S(P_i), i = 1, D$ is the set of initially compatible labels.
R is a relationship over which compatibility is determined.

```
procedure Relaxation_Labeling(P, S, R);
{
repeat
   for each (P_i, S(P_i))
   {
      for each label L_k ∈ S(P_i)
         for each relation R(P_i, P_j) over the image parts
            if ∃ L_m ∈ S(P_j) with R(L_k, L_m) in model
               then keep L_k in S(P_i)
               else delete L_k from S(P_i)
   }
until no change in any set S(P_i)
return(S);
}
```

Algorithm 6: Discrete Relaxation Labeling certainty that part p can be assigned label l. In this case the corresponding algorithms are called *continuous*. In this section we will look at a labeling algorithm called *continuous relaxation* for symmetric binary relations.

A continuous relaxation labeling problem is a 6–tuple $CLRP = (P, L, R_P, R_L, PR, C)$. As before, P is a set of parts, L is a set of labels for those parts, R_P is a relationship over the parts, and R_L is a relationship over the labels. L is usually given as the union over all parts i of L_i, the set of allowable labels for part i. Suppose that $|P| = n$. Then PR is a set of n functions $PR = \{pr_1 \ldots, pr_n\}$ where $pr_i(l)$ is the *a priori* probability that label l is valid for part i. C is a set of n^2 compatibility coefficients $C = \{c_{ij}\}, i = 1, \ldots, n; j = 1, \ldots, n$. c_{ij} can be thought of as the influence that part j has on the labels of part i. Thus, if we view the constraint relation R_P as a graph, we can view c_{ij} as a weight on the edge between part i and part j.

Instead of using R_P and R_L directly, we combine them to form a set R of n^2 functions $R = \{r_{ij}\}, i = 1, \ldots, n; j = 1, \ldots, n$, where $r_{ij}(l, l')$ is the compatibility of label l for part i with label l' for part j. In the discrete case, $r_{ij}(l, l')$ would be 1, meaning $((i, l), (j, l'))$ is allowed or 0, meaning that combination is incompatible. In the continuous case, $r_{ij}(l, l')$ can be any value between 0 and 1, indicating how compatible the relationship between parts i and j is with the relationship between labels l and l'. This information can come from R_P and R_L, which may be themselves simple, binary relations or may be attributed binary relations, where the attribute associated with a pair of parts (or pair of labels) represents the likelihood that the required relationship holds between them. The solution of a continuous relaxation labeling problem, like that of a consistent labeling problem, is a mapping $f : P \rightarrow L$ that assigns a label to each unit. Unlike the discrete case, there is no external definition stating what conditions such a mapping f must satisfy. Instead, the definition of f is implicit in the procedure that produces it. This procedure is known as *continuous relaxation*.

As discrete relaxation algorithms iterate to remove possible labels from the label set L_i of a part i, continuous relaxation iterates to update the probabilities associated with each part–label pair. The initial probabilities are defined by the set PR of functions defining *a priori* probabilities. The algorithm starts with these initial probabilities at step 0. Thus we define the probabilities at step 0 by

$$pr_i^0(l) = pr_i(l) \tag{11.18}$$

for each part i and label l. At each iteration k of the relaxation, a new set of probabilities $\{pr_i^k(l)\}$ is computed from the previous set and the compatibility information. In order to define $pr_i^k(l)$ we first introduce a piece of it, $q_i^k(l)$ defined by

$$q_i^k(l) = \sum_{\{j \mid (i,j) \in R_P\}} c_{ij} [\sum_{l' \in L_j} r_{ij}(l, l') pr_j^k(l')] \tag{11.19}$$

The function $q_i^k(l)$ represents the influence that the current probabilities associated with labels of other parts constrained by part i have on the label of part i. With this formulation, the formula for updating the pr_i^k's can be written as

$$pr_i^{k+1}(l) = \frac{pr_i^k(l)(1 + q_i^k(l))}{\sum_{l' \in L_i} pr_i^k(l')(1 + q_i^k(l'))} \tag{11.20}$$

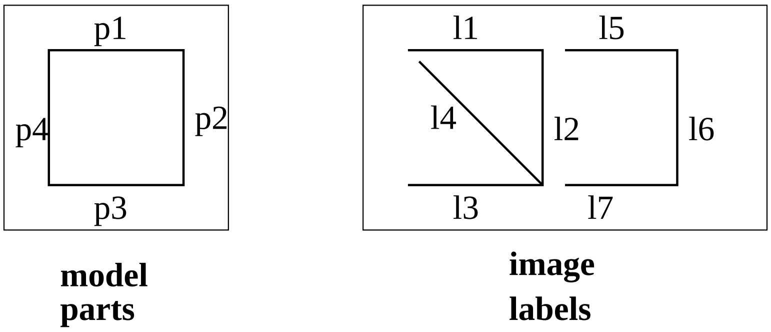


Figure 11.20: A model and image for continuous relaxation.

The numerator of the expression allows us to add to the current probability $pr_i^k(l)$ a term that is the product $pr_i^k(l)q_i^k(l)$ of the current probability and the opinions of other related parts, based on the current probabilities of their own possible labels. The denominator normalizes the expression by summing over all possible labels for part i.

Exercise 15 Continous Relaxation

Figure 11.20 shows a model and an image, each composed of line segments. Two line segments are said to be in the relationship *closadj* if their endpoints either coincide or are close to each other. a) Construct the attributed relation R_P over the parts of the model defined by $R_P = \{(p_i, p_j, d) | p_i \ closadj \ p_j\}$ and the attributed relation R_L over the labels of the image defined by $R_L = \{(l_i, l_j) | l_i \ closadj \ l_j\}$. b) Define the compatibility coefficients by $c_{ij} = 1 \ if \ (p_i, p_j) \in R_P \ else \ 0$. Use R_P and R_L together to define R in a manner of your choosing. Let $pr_i(l_j)$ be given as 1 if p_i has the same orientation as l_j, 0 if they are perpendicular, and .5 if one is diagonal and the other is horizontal or vertical. Define pr for the parts of the model and labels of the image. c) Apply several iterations of continuous relaxation to find a probable labeling from the model parts to the image labels.

Relational Distance Matching

A fully-consistent labeling is unrealistic in many real applications. Due to feature extraction errors, noise, and occlusion of one object by another, an image may have missing and extra parts, and required relationships may not hold. Continous relaxation may be used in these cases, but it is not guaranteed to find the best solution. In problems where finding an optimal solution is important, we can perform a search to find the *best* mapping f from P to L, in the sense that it preserves the most relationships and/or minimizes the number of NIL labels. The concept of *relational distance* as originally defined by Haralick and Shapiro [1981] allows us to define the best mapping in the general case where there may be any number of relations with possibly different dimensions. To do this we first need the concept of a *relational description* of an image or object.

12 DEFINITION *A relational description D_P is a sequence of relations $D_X = \{R_1, \ldots, R_I\}$ where for each $i = 1, \ldots, I$, there exists a positive integer n_i with $R_i \subseteq P^{n_i}$ for some set P. P is a set of the parts of the entity being described and the relations R_i indicate various relationships among the parts.*

A relational description is a data structure that may be used to describe two–dimensional shape models, three-dimensional object models, regions on an image, and so on.

Let $D_A = \{R_1, \ldots, R_I\}$ be a relational description with part set A and $D_B = \{S_1, \ldots, S_I\}$ be a relational description with part set B. We will assume that $\mid A \mid = \mid B \mid$; if this is not the case, we will add enough dummy parts to the smaller set to make it the case. The assumption is made in order to guarantee that the relational distance is a metric.

Let f be any one-one, onto mapping from A to B. For any $R \subseteq A^N$, N a positive integer, the *composition* $R \circ f$ of relation R with function f is given by

$$R \circ f = \{(b_1, \ldots, b_N) \in B^N \mid there\ exists (a_1, \ldots, a_N) \in R$$

$$with\ f(a_n) = b_n, n = 1, \ldots, N\}$$

This composition operator takes N–tuples of R and maps them, component by component, into N–tuples of B^N.

The function f maps parts from set A to parts from set B. The *structural error* of f for the ith pair of corresponding relations (R_i and S_i) in D_A and D_B is given by

$$E_S^i(f) = \mid R_i \circ f - S_i \mid + \mid S_i \circ f^{-1} - R_i \mid .$$

The structural error indicates how many tuples in R_i are not mapped by f to tuples in S_i and how many tuples in S_i are not mapped by f^{-1} to tuples in R_i. The structural error is expressed with respect to only one pair of corresponding relations.

The *total error* of f with respect to D_A and D_B is the sum of the structural errors for each pair of corresponding relations. That is,

$$E(f) = \sum_{i=1}^{I} E_S^i(f).$$

The total error gives a quantitative idea of the difference between the two relational descriptions D_A and D_B with respect to the mapping f.

The *relational distance* $GD(D_A, D_B)$ between D_A and D_B is then given by

$$GD(D_A, D_B) = \min_{\substack{f: A \xrightarrow[\text{onto}]{1-1} B}} E(f).$$

That is, the relational distance is the minimal total error obtained for any one–one, onto mapping f from A to B. We call a mapping f that minimizes total error a *best mapping* from D_A to D_B. If there is more than one best mapping, additional information that is outside the pure relational paradigm can be used to select the preferred mapping. More than one

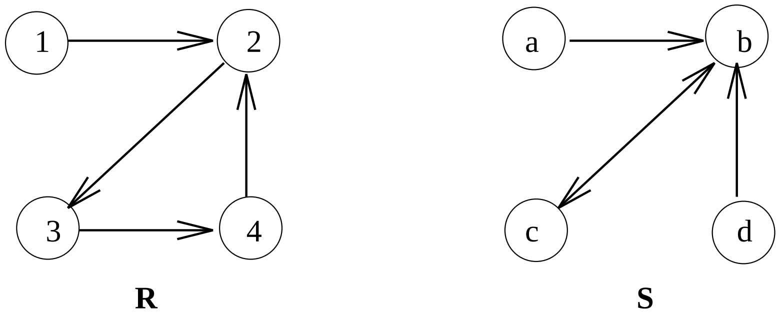


Figure 11.21: Two digraphs whose relational distance is 3.

best mapping will occur when the relational descriptions involve certain kinds of symmetries.

We illustrate the relational distance with several examples. Figure 11.21 shows two digraphs, each having four nodes. A best mapping from $A = \{1, 2, 3, 4\}$ to $B = \{a, b, c, d\}$ is $\{f(1) = a, f(2) = b, f(3) = c, f(4) = d\}$. For this mapping we have

$$\begin{aligned}
|R \circ f - S| &= |\{(1,2)(2,3)(3,4)(4,2)\} \circ f - \{(a,b)(b,c)(c,b)(d,b)\}| \\
&= |\{(a,b)(b,c)(c,d)(d,b)\} - \{(a,b)(b,c)(c,b)(d,b)\}| \\
&= |\{(c,d)\}| \\
&= 1 \\
|S \circ f^{-1} - R| &= |\{(a,b)(b,c)(c,b)(d,b)\} \circ f^{-1} - \{(1,2)(2,3)(3,4)(4,2)\}| \\
&= |\{(1,2)(2,3)(3,2)(4,2)\} - \{(1,2)(2,3)(3,4)(4,2)\}| \\
&= |\{(3,2)\}| \\
&= 1 \\
E(f) &= |R \circ f - S| + |S \circ f^{-1} - R| \\
&= \quad 1 \quad + \quad 1 \\
&= 2
\end{aligned}$$

Since f is a best mapping, the relational distance is also 2.

Figure 11.22 gives a set of object models M_1, M_2, M_3, and M_4 whose primitives are image regions. Two relations are shown in the figure: the connection relation and the parallel relation. Both are binary relations over the set of primitives. Consider the first two models, M_1 and M_2. The best mapping f maps primitive 1 to $1'$, 2 to $2'$, and 3 to $3'$. Under this mapping the connection relations are isomorphic. The parallel relationship $(2, 3)$ in model M_1 does not hold between $2'$ and $3'$ in model M_2. Thus the relational distance between M_1 and M_2 is exactly 1. Now consider models M_1 and M_3. The best mapping maps 1 to $1''$, 2 to $2''$, 3 to $3''$, and a dummy primitive to $4''$. Under this mapping, the parallel relations are

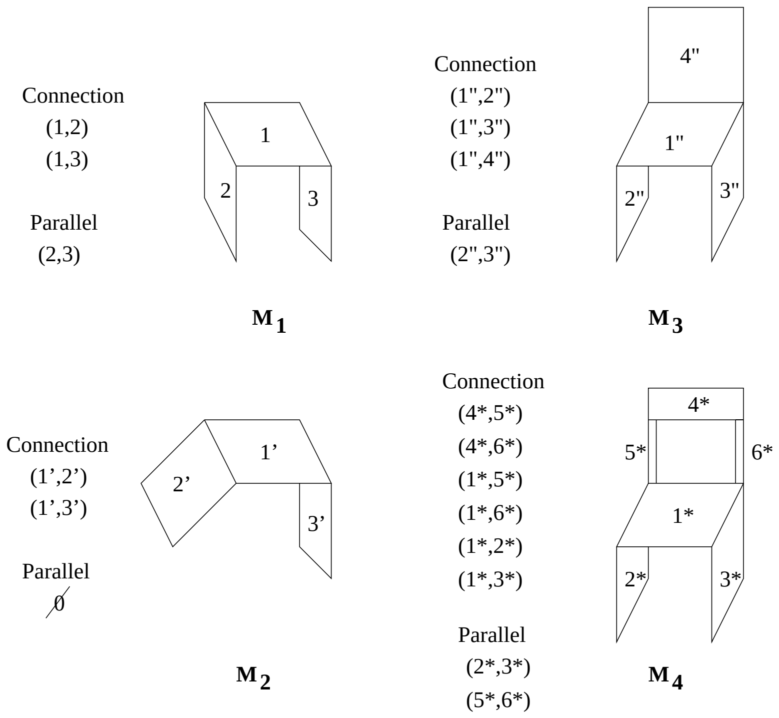


Figure 11.22: Four object models. The relational distance of model M_1 to M_2 and M_1 to M_3 is 1. The relational distance of model M_3 to M_4 is 6.

now isomorphic but there is one more connection in M_3 than in M_2. Again the relational distance is exactly 1.

Finally consider models M_3 and M_4. The best mapping maps $1''$ to 1^*, $2''$ to 2^*, $3''$ to 3^*, $4''$ to 4^*, 5_d to 5^*, and 6_d to 6^*. (5_d and 6_d are dummy primitives.) For this mapping we have

$$\begin{aligned}
|R_1 \circ f - S_1| &= |\{(1'',2'')(1'',3'')(1'',4'')\} \circ f - \\
&\qquad \{(4^*,5^*)(4^*,6^*)(1^*,5^*)(1^*,6^*)(1^*,2^*)(1^*,3^*)\}| \\
&= |\{(1^*,2^*)(1^*,3^*)(1^*,4^*)\} - \\
&\qquad \{(4^*,5^*)(4^*,6^*)(1^*,5^*)(1^*,6^*)(1^*,2^*)(1^*,3^*)\}| \\
&= |\{(1^*,4^*)\}| \\
&= 1 \\
|S_1 \circ f^{-1} R_1| &= |\{(4^*,5^*)(4^*,6^*)(1^*,5^*)(1^*,6^*)(1^*,2^*)(1^*,3^*)\} \circ f^{-1} - \\
&\qquad \{(1'',2'')(1'',3'')(1'',4'')\}| \\
&= |\{(4'',5_d)(4',6_d)(1'',5_d)(1'',6_d)(1'',2'')(1'',3'')\} - \\
&\qquad \{1'',2'')(1'',3'')(1'',4'')\}| \\
&= |\{(4'',5_d)(4'',6_d)(1'',5_d)(1'',6_d)\}| \\
&= 4 \\
|R_2 \circ f - S_2| &= |\{(2'',3'')\} \circ f - \{2^*,3^*)(5^*,6^*)\}| \\
&= |\{(2^*,3^*)\} - \{(2^*,3^*)(5^*,6^*)\}| \\
&= |\emptyset| \\
&= 0 \\
|S_2 \circ f^{-1} - R_2| &= |\{(2^*,3^*)(5^*,6^*)\} \circ f^{-1} - \{(2'',3'')\}| \\
&= |\{(2'',3'')(5_d,6_d)\} - \{(2'',3'')\}| \\
&= |\{(5_d,6_d)\}| \\
&= 1 \\
E_S^1(f) &= 1 + 4 = 5 \\
E_S^2(f) &= 0 + 1 = 1 \\
E(f) &= 6
\end{aligned}$$

Exercise 16 Relational Distance Treesearch

Modify the algorithm for Interpretation Tree Search to find the relational distance between two structural descriptions and to determine the best mapping in the process of doing so.

Exercise 17 One-Way Relational Distance

The above definition of relational distance uses a two-way mapping error, which is useful when comparing two objects that stand alone. When matching a model to an image, we often want to use only a one-way mapping error, checking how many relationships of the model are in the image, but not vice versa. Define a modified one-way relational distance that can be used for model-image matching.

Exercise 18 NIL mappings in Relational Distance

The above definition of relational distance does not handle NIL labels explicitly. Instead, if part j has a NIL label, then any relationship (i, j) will cause an error, since $(f(i), NIL)$ will not be present. Define a modified relational distance that counts NIL labels as errors only once and does not penalize again for missing relationships caused by NIL labels.

Exercise 19 Attributed Relational Distance

The above definition also does not handle attributed relations in which each tuple, in addition to a sequence of parts, contains one or more attributes of the relation. For example, a connection relation for line segments might have as an attribute the angle between the connecting segments. Formally, an attributed n-relation R over part set P and attribute set A is a set $R \subseteq P_n \times A_m$ for some nonnegative integer m that specifies the number of attributes in the relation. Define a modified relational distance in terms of attributed relations.

Relational Indexing

Sometimes a tree search even with relaxation filtering is too slow, especially when an image is to be compared to a large database of models. For structural descriptions in terms of labeled relations, it is possible to approximate the relational distance with a simpler voting scheme. Intuitively, suppose we observe two coincentric circles and two 90 degree corners connected by an edge. We would like to quickly find all models that have these structures and match them in more detail. To achieve this, we can build an index that allows us to look up the models given the partial graph structure. Given two coincentric circles, we look up all models containing these related features and give each of those models one vote. Then, we look up all models having connected 90 degree corners: any models repeating from the first test will now have two votes. These lookups can be done rapidly provided that an index is built offline before recognition by extracting significant binary relations from each model and recording each in a lookup table.

Let $DB = \{M_1, M_2, \ldots, M_t\}$ be a database of t object models. Each object model M_t consists of a set of attributed parts P_T plus a labeled relation R_T. For simplicity of explanation, we will assume that each part has a single label, rather than a vector of attributes and that the relation is a binary relation, also with a single label attached to each tuple. In this case, a model is represented by a set of *two-graphs* each of which is a graph with two nodes and two directed edges. Each node represents a part, and each edge represents a directed binary relationship. The value in the node is the *label* of the part, rather than a unique identifier. Similarly, the value in an edge is the *label* of the relationship. For example, one node could represent an ellipse and another could represent a pair of parallel lines. The edge from the parallel lines node to the ellipse node could represent the relationship "encloses", while the edge in the opposite direction represents the relationship "is enclosed by".

Relational indexing requires a preprocessing step in which a large hash table is set up. The hash table is indexed by a string representation of a two-graph. When it is completed, one can look up any two-graph in the table and quickly retrieve a list of all models containing that particular two-graph. In our example, all models containing an ellipse between two parallel line segments can be retrieved. During recognition of an object from an image, the

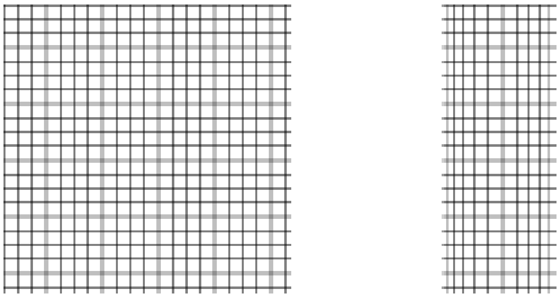


Figure 11.23: (Left) Regular grid of lines; (right) grid warped by wrapping it around a cylinder.

Figure 11.24: (Left) Image of center of US$20 bill; (center) image of Andrew Jackson wrapped around a cylinder of circumference 640 pixels; (right) same as center except circumference is 400 pixels.

features are extracted and all the two-graphs representing the image are computed. A set of accumulators, one for each model in the database are all set to zero. Then each two-graph in the image is used to index the hash table, retrieve the list of associated models, and vote for each one. The discrete version of the algorithm adds one to the vote; a probabilistic algorithm would add a probability value instead. After all two-graphs have voted, the models with the largest numbers of votes are candidates for verification.

11.7 Nonlinear Warping

Nonlinear warping functions are also important. We may need to rectify nonlinear distortion in an image; for example, radial distortion from a fisheye lens. Or, we may want to distort an image in creative ways. Figure 11.23 shows a nonlinear warp which maps a regular grid onto a cylinder. The effect is the same as if we wrapped a flat image around a cylinder and then viewed the cylinder from afar. This same warp applied to a twenty dollar bill is shown in Figure 11.24. Intuitively, we need to choose some image axis corresponding to the center of a cylinder and then use a formula which models how the pixels of the input image will "compress" off the cylinder axis in the output image. Figure 11.24 shows two warps; the rightmost one uses a cylinder of smaller radius than the center one.

Figure 11.25 shows how to derive the cylindrical warp. We choose an axis for the warp

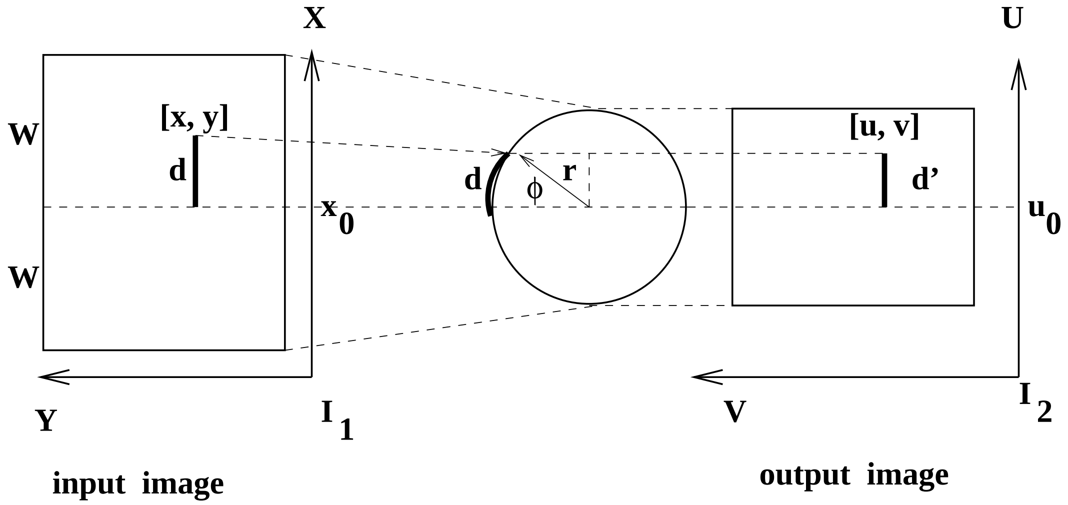


Figure 11.25: The output image at the right is created by "wrapping" the input image at the left around a cylinder (center): distance **d** in the input image becomes distance **d'** on output.

(determined by x_0) and a width W. W corresponds to one quarter of the circumference of the cylinder. Any length d in the input image is "wrapped around the cylinder" and then projected to the output image. Actually, d corresponds to the length $x - x_0$ where x_0 is the x-coordinate of the axis of the cylinder. The y coordinate of the input image point is preserved by the warp, so we have $v = y$. From the figure, the following equations are seen to hold. First, $W = (\pi/2)r$ since W accounts for one quarter of the circumference. d accounts for a fraction of that: $d/W = \phi/(\pi/2)$ and $\sin\phi = d'/r$. Combining these yields $d = x - x_0 = (2W/\pi)\arcsin((\pi/2W)(u - u_0))$. Of course, $d' = u - u_0 = u - x0$.

We now have a formula for computing input image coordinates $[x, y]$ from output coordinates $[u, v]$ and the warp parameters x_0 and W. This seems to be backwards; why not take each pixel from the input image and transform it to the output image? Were we to do this, there would be no guarantee that the output image would have each pixel set. For a digital image, we would like to generate each and every pixel of output exactly once and obtain the pixel value from the input image as the following algorithm shows. Moreover, using this approach it is easily possible for the output image to have more or fewer pixels than the input image. The concept is that in generating the output image, we *map back* into the input image and *sample* it.

Perform a cylindrical warp operation.
$^1I[x, y]$ is the input image.
x_0 is the axis specification.
W is the width.
$^2I[u, v]$ is the output image.

```
procedure Cylindrical_Warp(¹I[x, y])
{
r = 2W/π;
for u:=0, Nrows-1
   for v:=0, Ncols-1
   {
      ²I[u, v] = 0; // set as background
      if (|u − u₀| ≤ r)
         {
            x = x₀ + r arcsin((u − x₀)/r) ;
            y = v;
            ²I[u, v]  =  ¹I[round(x), round(y)];
         }
   }
return(²I[u, v]);
}
```

Algorithm 7: Cylindrical Warp of Image Region

Exercise 20

(a) Determine the transformation that maps a circular region of an image onto a hemisphere and then projects the hemisphere onto an image. The circular region of the original image is defined by a center (x_c, y_c) and radius r_0. (b) Develop a computer program to carry out the mapping.

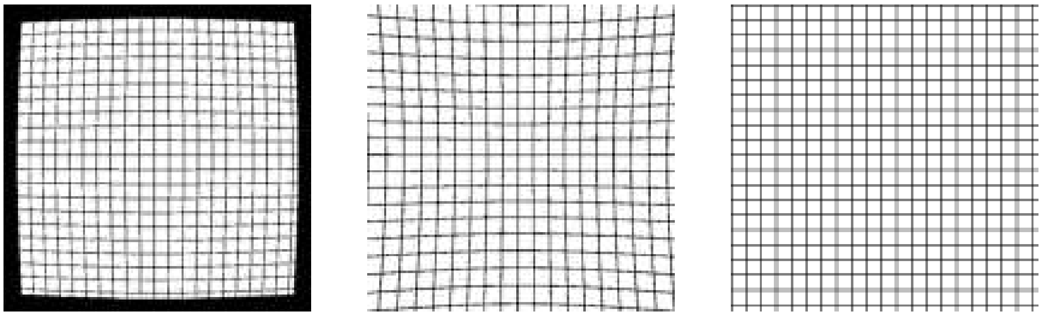


Figure 11.26: Two types of radial distortion, barrel (left) and pincushion (center) which can be removed by warping to produce a rectified image (right).

Rectifying Radial Distortion

Radial distortion is present in most lenses: it might go unnoticed by human interpreters, but sometimes can produce large errors in photometric measurements if not corrected. Physical arguments deduce that radial distortion in the location of an imaged point is proportional to the distance from that point to the optical axis. Figure 11.26 shows two common cases of radial distortion along with the desirable rectified image. If we assume that the optical axis passes near the image center, then the distortion can be corrected by displacing all image points either toward or away from the center by a displacement proportional to the square of the distance from the center. This is not a linear transformation since the displacement varies across the image. Sometimes, more even powers of the radial distance are used in the correction, as the mathematical model below shows. Let $[x_c, y_c]$ be the image center which we are assuming is also where the optical axis passes through the image. The corrections for the image points are as follows, assuming we need the first two even powers of the radial distance to compute the radial distortion. The best values for the constants c_2 and c_4 can be found by empirical study of the radial displacements of known control points or by formally using least-squares fitting in a calibration process.

$$\begin{aligned} R &= \sqrt{((x - x_c)^2 + (y - y_c)^2)} \\ D_r &= (c_2 R^2 + c_4 R^4) \\ x &= x_c \ + \ (x - x_c) D_r \\ y &= y_c \ + \ (y - y_c) D_r \end{aligned} \tag{11.21}$$

Polynomial Mappings

Many small global distortion factors can be rectified using polynomial mappings of maximum degree two in two variables as defined in Equation 11.22. Twelve different coefficients must be estimated in order to adapt to the different geometric factors. To estimate these coefficients, we need the coordinates of at least six control points before and after mapping; however, many more points are used in practice. (Each such control point yields two equations.) Note that if only the first three terms are used in Equation 11.22 the mapping is an affine mapping.

$$\begin{aligned} u &= a_{00} + a_{10}x + a_{01}y + a_{11}xy + a_{20}x^2 + a_{02}y^2 \\ v &= b_{00} + b_{10}x + b_{01}y + b_{11}xy + b_{20}x^2 + b_{02}y^2 \end{aligned} \tag{11.22}$$

Exercise 21

Show that radial distortion in Equation 11.21 with $c_4 = 0$ can be modeled exactly by a polynomial mapping of the form shown in Equation 11.22.

11.8 Summary

Multiple concepts have been discussed in this chapter under the theme of 2D matching. One major theme was 2D mapping using transformations. These could be used as simple image processing operations which could extract or sample a region from an image, register two images together in the same coordinate system, or remove or creatively add distortion to 2D images. Algebraic machinery was developed for these transformations and various methods and applications were discussed. This development is continued in Chapter 13 as it relates to mapping points in 3D scenes and 3D models. The second major theme of this chapter was the interpretation of 2D images through correspondences with 2D models. A general paradigm is *recognition-by-alignment*: the image is interpreted by discovering a model and an RST transformation such that the transformation maps known model structures onto image structures. Several different algorithmic approaches were presented, including pose-clustering, interpretation tree search, and the feature focus method. Discrete relaxation and relational matching were also presented: these two methods can be applied in very general contexts even though they were introduced here within the context of constrained geometric relationships. Relational matching is potentially more robust than rigid alignment when the relations themselves are more robust than those depending on metric properties. Image distortions caused by lens distortions, slanted viewing axes, quantification effects, etc, can cause metric relations to fail; however, topological relationships such as cotermination, connectivity, adjacency, and insideness are usually invariant to such distortions. A successful match using topological relationships on image/model parts might then be used to find a large number of matching points, which can then be used to find a mapping function with many parameters that can adapt to the metric distortions. The methods of this chapter are directly applicable to many real world applications. Subsequent chapters extend the methods to 3D.

11.9 References

The paper by Van Wie and Stein (1977) discusses a system to automatically bring satellite images into registration with a map. An approximate mapping is known using the time at which the image is taken. This mapping is used to do a refined search for control points using templates: the control points are then used to obtain a refined mapping. The book by Wolberg gives a complete account of image warping including careful treatment of sampling the input image and lessening aliasing by smoothing. The treatment of 2D matching via pose clustering was drawn from Stockman et al (1982), which contained the airplane detection example provided. A more general treatment handling the 3D case is given in Stockman

1987. The paper by Grimson and Lozano-Perez demonstrates how distance constraints can be used in matching model points to observed data points. Least squares fitting is treated very well in other references and is a topic of much depth. Least squares techniques are in common use for the purpose of estimating transformation parameters in the presence of noise by using many more than the minimal number of control points. The book by Wolberg (1990) treats several least squares methods within the context of warping while the book by Daniel and Wood (1971) treats the general fitting problem. In some problems, it is not possible to find a good geometric transformation that can be applied globally to the entire image. In such cases, the image can be partitioned into a number of regions, each with its own control points, and separate warps can be applied to each region. The warps from neighboring regions must smoothly agree along the boundary between them. The paper by Gostasby (1988) presents a flexible way of doing this.

Theory and algorithms for the consistent labeling problems can be found in the papers by Haralick and Shapiro (1979,1980). Both discrete and continuous relaxation are defined in the paper by Rosenfeld, Hummel, and Zucker (1976), and continuous relaxation is further analyzed in the paper by Hummel and Zucker (1983). Methods for matching using structural descriptions have been derived from Shapiro (1981); relational indexing using 2-graphs can be found in Costa and Shapiro (1995). Use of invariant attributes of structural parts for indexing into models can be found in Chen and Stockman (1996).

1. J.L. Chen and G. Stockman, *Indexing to 3D Model Aspects using 2D Contour Features*, Proc. Int. Conf. Computer Vision and pattern Recognition (CVPR), San Francisco, CA June 18-20, 1996. Expanded paper to appear in the journal CVIU.

2. M. Clowes (1971) *On seeing things*, Artifical Intelligence, Vol. 2, pp79-116.

3. M. S. Costa and L. G. Shapiro (1995), *Scene Analysis Using Appearance-Based Models and Relational Indexing*, *IEEE Symposium on Computer Vision*, November, 1995, pp. 103-108.

4. C. Daniel and F. Wood (1971) *Fitting Equations to Data* John Wiley and Sons, Inc., New York.

5. A.Goshtasby (1988) *Image Registration by Local Approximation Methods*, Image and Vision Computing, Vol. 6, No. 4 (Nov 1988) pp255-261.

6. W. Grimson and T. Lozano-Perez (1984) *Model-Based Recognition and Localization From Sparse Range or Tactile Data*, Int. Journal of Robotics Research, Vol. 3, No. 3 pp3-35.

7. R. Haralick and L. Shapiro (1979) *The consistent labeling problem I*, IEEE Trans. PAMI-1, pp173-184.

8. R. Haralick and L. Shapiro (1980) *The consistent labeling problem II*, IEEE Trans. PAMI-2, pp193-203.

9. R. Hummel and S. Zucker (1983) *On the Foundations of Relaxation Labeling Processes*, IEEE Trans. PAMI-5, pp267-287.

10. Y. Lamden and H. Wolfson (1988) *Geometric Hashing: A General and Efficient Model-Based Recognition Scheme*, Proc. 2nd Int. Conf. on Computer Vision, Tarpon Springs, FL. (Nov 1988) pp238-249.

11. D. Rogers and J. Adams (**) **Mathematical Elements for Computer Graphics, 2nd Ed.**. McGraw-Hill.

12. A. Rosenfeld, R. Hummel, and S. Zucker (1976) *Scene Labeling by Relaxation Operators*, IEEE Trans. Systems, Man, and Cybernetics, SMC-6, pp. 420-453.

13. G. Stockman, S. Kopstein and S. Benett (1982) *Matching Images to Models for Registration and Object Detection via Clustering*, IEEE Trans. on PAMI, Vol. PAMI-4, No. 3 (May 1982) pp 229-241.

14. G. Stockman (1987) *Object Recognition and Localization via Pose Clustering*, Computer Vision, Graphics and Image Processing, Vol. 40 (1987) pp 361-387.

15. P Van Wie and M. Stein (1977) *A LANDSAT digital image rectification system*, IEEE Trans. Geosci. Electron., Vol. GE-15, July, 1977.

16. P.Winston (1977) **Artificial Intelligence** Addison-Wesley.

17. G. Wolberg (1990) **Digital Image Warping**, IEEE Computer Society Press, Los Alamitos, CA.