9
Features

In this chapter we explore some techniques for the representation and extraction of image features. Typically, the aim is to process the image in such a way that the image, or properties of it, can be adequately represented and extracted in a compact form amenable to subsequent recognition and classification. Representations are typically of two basic kinds: internal (which relates to the pixels comprising a region) or external (which relates to the boundary of a region). We will discuss shape-based representations (external), features based on texture and statistical moments (internal) and end the chapter with a detailed look at principal component analysis (PCA), a statistical method which has been widely applied to image processing problems.

9.1 Landmarks and shape vectors

A very basic representation of a shape simply requires the identification of a number N of points along the boundary of the features. In two dimensions, this is simply

$$\mathbf{x} = [x_1 \quad y_1 \quad x_2 \quad \cdots \quad x_N \quad y_N]$$

Two basic points about the shape vector representation should be immediately stressed:

- In practice, we are restricted to a finite number of points. Therefore, we need to ensure that a sufficient number of points are chosen to represent the shape to the accuracy we require.

- Selection of the coordinate pairs which constitute the shape vector must be done in a way that ensures (as far as possible) good agreement or *correspondence*. By correspondence, we mean that the selected coordinates constituting the boundary/shape must be chosen in such a way as to ensure close agreement across a group of observers. If this were not satisfied, then different observers would define completely different shape vectors for what is actually the same shape. Correspondence is achieved by defining criteria for identifying appropriate and well-defined key points along the boundaries. Such key points are called *landmarks*.

Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab Chris Solomon and Toby Breckon
 ISBN: 978-0-470-84472-4

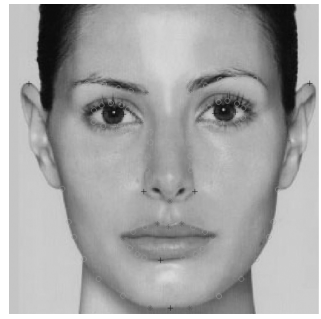


Figure 9.1 Anatomical landmarks (indicated in red) are located at points which can be easily identified visually. Mathematical landmarks (black crosses) are identified at points of zero gradient and maximum corner content. The pseudo-landmarks are indicated by green circles (*See colour plate section for colour version*)

Landmarks essentially divide into three categories:

- *Mathematical landmarks* These correspond to points located on an object according to some mathematical or geometric property (e.g. according to local gradient, curvature or some extremum value). 'Interest' detectors, such as those devised by Harris and Haralick (discussed in Chapter 10) combined with thresholding constitute a good example of a mathematical criterion for the calculation of landmarks.

- *Anatomical or true landmarks* These are points assigned by an expert which are identifiable in (and correspond between) different members of a class of objects. Examples of anatomical landmarks are the corners of the mouth and eyes in human faces.[1]

- *Pseudo-landmarks* These are points which are neither mathematically nor anatomically completely well defined, though they may certainly be intuitively meaningful and/or require some skilled judgment on the part of an operator. An example of a pseudo-landmark might be the 'centre of the cheek bones' in a human face, because the visual information is not definite enough to allow unerring placement. Despite their 'second-rank' status, pseudo-landmarks are often very important in practice. This is because true landmarks are often quite sparse and accurate geometric transformations (discussed in the geometry chapter) usually require correspondence between two similar (but subtly different) objects at a large and *well-distributed* set of points over the region of interest. In many cases, pseudo-landmarks are constructed as points between mathematical or anatomical landmarks. Figure 9.1 depicts an image of a human face in which examples of all three kinds of landmark have been placed.

Landmarks[2] can also be defined in an analogous way for 3-D shapes or (in principle) even in abstract higher dimensional 'objects'.

[1] 'Anatomical' landmarks can be defined for any pattern class of objects. The term is not confined to strictly biological organisms.

[2] Labelling images with anatomical landmarks can become very tedious and is subject to a certain degree of uncontrollable error. Accordingly, one of the big challenges of current image processing/computer vision research is to develop methods which can *automate* the procedure of finding landmarks.

Table 9.1 Some common, single-parameter descriptors for approximate shape in 2-D

Measure	Definition	Circle	Square	Rectangle as $b/a \to \infty$
Form factor	$\frac{4\pi \times \text{Area}}{\text{Perimeter}^2}$	1	$\pi/4$	$\to 0$
Roundness	$\frac{4 \times \text{Area}}{\pi \times \text{MaxDiameter}^2}$	1	$2/\pi$	$\to 0$
Aspect ratio	$\frac{\text{MaxDiameter}}{\text{MinDiameter}}$	1	1	$\to \infty$
Solidity	$\frac{\text{Area}}{\text{ConvexArea}}$	1	1	$\to 0$
Extent	$\frac{\text{TotalArea}}{\text{Area Bounding Rectangle}}$	$\pi/4$	1	indeterminate
Compactness	$\frac{\sqrt{(4 \times \text{Area})/\pi}}{\text{MaxDiameter}}$	1	$\sqrt{2/\pi}$	$\to 0$
Convexity	$\frac{\text{Convex Perimeter}}{\text{Perimeter}}$	1	1	1

9.2 Single-parameter shape descriptors

In certain applications, we do not require a shape analysis/recognition technique to provide a precise, regenerative representation of the shape. The aim is simply to characterize a shape as succinctly as possible in order that it can be differentiated from other shapes and classified accordingly. Ideally, a small number of descriptors or even a single parameter can be sufficient to achieve this task. Differentiating between a boomerang and an apple, a cigar and a cigarette can intuitively be achieved by quite simple methods, a fact reflected in our everyday language ('long and curved', 'roughly spherical', 'long, straight and thin', etc).

Table 9.1 gives a number of common, single-parameter measures of approximate shape which can be employed as shape features in basic tasks of discrimination and classification. Note that many of these measures can be derived from knowledge of the perimeter length, the extreme x and y values and the total area enclosed by the boundary – quantities which can be calculated easily in practice. All such measures of shape are a dimensionless combination of size parameters which, in many cases, exploit the variation in perimeter length to surface area as shape changes. Meaningful reference values for such measures are usually provided by the circle or square. Table 9.1 gives these measures for three basic shapes: the circle, the square and the limiting case of a rectangle of length a and side b when $b/a \to \infty$.

Figure 9.2 (produced using the Matlab® code in Example 9.1) shows an image on the left containing a number of metallic objects: a key, a drill bit and a collection of coins. The coins are actually divisible into two basic shapes: those which are circular (1 and 2 pence coins) and those which are heptagonal (the 20 pence and 50 pence coins). The centre image in Figure 9.2 shows the processed binary objects after thresholding and basic morphological processing

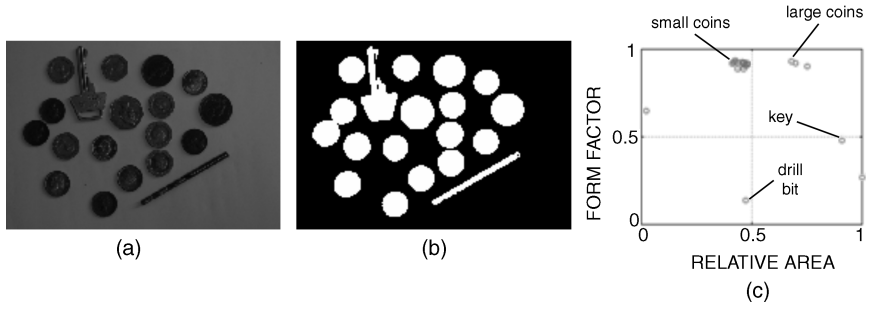


Figure 9.2 From left to right: (a) original image; (b) binary image after thresholding and morphological processing; (c) the normalized area and the form factor of each object in (b) are plotted in a 2-D feature space

and the image on the right plots their associated areas and form factors. The key and drill bit are easily distinguished from the coins by their form-factor values. However, the imperfect result from thresholding means that the form factor is unable to distinguish between the heptagonal coins and circular coins. These are, however, easily separated by area.

Example 9.1

Matlab code	**What is happening?**
A=imread('coins_and_keys.png'); subplot(1,2,1), imshow(A);	%Read in image and display
bw=∼im2bw(rgb2gray(A),0.35); bw=imfill(bw,'holes');	%Threshold and fill in holes
bw=imopen(bw,ones(5)); subplot(1,2,2), imshow(bw,[0 1]);	%Morphological opening
[L,num]=bwlabel(bw);	%Create labelled image
s=regionprops(L,'area','perimeter');	%Calculate region properties
for i=1:num	%Object's area and perimeter
x(i)=s(i).Area;	
y(i)=s(i).Perimeter;	
form(i)=4.*pi.*x(i)./(y(i).^2);	%Calculate form factor
end	
figure; plot(x./max(x),form,'ro');	%Plot area against form factor

Comments

In this example, intensity thresholding produces a binary image containing a variety of objects of different shapes. The ***regionprops*** function is then used to calculate the area and perimeter of each of the objects in the binary object. ***regionprops*** is a powerful in-built function called which can calculate a number of basic and useful shape-related measures. It requires as input a so-called *labelled image* (a matrix in which all pixels belonging to object 1 have value 1, all pixels belonging to object 2 have value 2 and so on). This is generated through use of the Matlab function ***bwlabel***. The form factor is then calculated and plotted against the object area.

This example also uses the morphological functions ***imfill*** (to fill in the holes in the objects) and ***imopen*** (to remove any small isolated objects), both of which result from imperfect thresholding. These and other functions are discussed more fully in Chapter 8.

9.3 Signatures and the radial fourier expansion

A *signature* is the representation of a 2-D boundary as a 1-D function. Typically, this is achieved by calculating the centroid of the given shape (as defined by the boundary coordinates) and plotting the distance from the centroid to the boundary r as a function of polar angle θ. The signature then of a *closed* boundary is a periodic function, repeating itself on an angular scale of 2π. Thus, one simple and neat way to encode and describe a closed boundary to arbitrary accuracy is through a 1-D (radial) Fourier expansion. The signature can be expressed in real or complex form as follows:

$$\begin{aligned} r(\theta) &= \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(n\theta) + \sum_{n=1}^{\infty} b_n \sin(n\theta) \quad \text{(real form)} \\ r(\theta) &= \sum_{n=-\infty}^{\infty} c_n \exp(\mathrm{i}n\theta) \qquad\qquad\qquad\quad \text{(complex form)} \end{aligned} \tag{9.1}$$

and the shape can be parametrically encoded by the real Fourier expansion coefficients $\{a_n, b_n\}$ or by the complex coefficients $\{c_n\}$.[3] These coefficients can easily be calculated through use of the orthogonality relations for Fourier series (see Chapter 5). The real coefficients for a radial signature are given by

$$\begin{aligned} a_n &= \frac{1}{\pi} \int_{-\pi}^{\pi} r(\theta) \cos(n\theta)\, \mathrm{d}\theta \\ b_n &= \frac{1}{\pi} \int_{-\pi}^{\pi} r(\theta) \sin(n\theta)\, \mathrm{d}\theta \end{aligned} \tag{9.2}$$

whereas the complex coefficients $\{c_n\}$ are given by

$$c_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} r(\theta) \exp(-\mathrm{i}n\theta)\, \mathrm{d}\theta \tag{9.3}$$

Typically, a good approximation to the shape can be encoded using a relatively small number of parameters, and more terms can be included if higher accuracy is required. Figure 9.3 (produced using the Matlab code in Example 9.2) shows how a relatively complex boundary (the Iberian Peninsula, comprising Spain and Portugal) can be increasingly well approximated by a relatively modest number of terms in the Fourier expansion.

Example 9.2

Matlab code	**What is happening?**
A=imread('spain.png'); iberia=logical(A);	%Read in image convert to binary
ibbig=imdilate(iberia,ones(3));	%Calculate boundary pixels
bound=ibbig-iberia;	
[i,j]=find(bound>0); xcent=mean(j);	%Calculate centroid
ycent=mean(i);	
hold on; plot(xcent,ycent,'ro');	

[3] Note that for a digital boundary specified by N points, the order of the expansion n cannot exceed N.

```
subplot(4,2,1),imagesc(bound); axis image;
axis on; colormap(gray);
                                                    %Plot perimeter
[th,r]=cart2pol(j-xcent,i-ycent);                   %Convert to polar coordinates
subplot(4,2,2); plot(th,r,'k.'); grid on; axis off; %Plot signature

                                                    %Calculate Fourier series boundary
N=0;L=2.*pi;f=r;M=length(f);                         %N = 0 – DC term only
[a,b,dc]=fseries_1D(f,L,N);                         %Calculate expansion coeffs
fapprox=fbuild_1D(a,b,dc,M,L);                      %Build function using the coeffs
[x,y]=pol2cart(th,fapprox);
x=x+xcent; x=x-min(x)+1; y=y+ycent;                 %Convert back to Cartesian coordinates
y=y-min(y)+1;
prm=zeros(round(max(y)),round(max(x)));
i=sub2ind(size(prm),round(y),round(x)); prm(i)=1;
subplot(4,2,3); imagesc(prm); axis image;
axis ij; axis on; colormap(gray);
                                                    %Display 2-D boundary
subplot(4,2,4); plot(th,fapprox,'k.'); axis off;    %Display corresponding signature
s=sprintf('Approximation %d terms',N); title(s);

N=5;L=2.*pi;f=r;M=length(f);                         %Repeat for N = 5 terms
[a,b,dc]=fseries_1D(f,L,N);                         %Calculate expansion coeffs
fapprox=fbuild_1D(a,b,dc,M,L);                      %Build function using the coeffs
[x,y]=pol2cart(th,fapprox);
x=x+xcent; x=x-min(x)+1; y=y+ycent;
y=y-min(y)+1;
prm=zeros(round(max(y)),round(max(x)));
i=sub2ind(size(prm),round(y),round(x)); prm(i)=1;
subplot(4,2,5); imagesc(prm); axis image;
axis ij; axis off; colormap(gray);
                                                    %Display 2-D boundary
subplot(4,2,6); plot(th,fapprox,'k.'); axis off;    %Display corresponding signature
s=sprintf('Approximation %d terms',N); title(s);

N=15;L=2.*pi;f=r;M=length(f);                        %Repeat for N = 15 terms
[a,b,dc]=fseries_1D(f,L,N);                         %Calculate expansion coeffs
fapprox=fbuild_1D(a,b,dc,M,L);                      %Build function using the coeffs
[x,y]=pol2cart(th,fapprox);                         %Convert back to Cartesian coordinates
x=x+xcent; x=x-min(x)+1; y=y+ycent; y=y-min(y)+1;
prm=zeros(round(max(y))+10,round(max(x))+10);
i=sub2ind(size(prm),round(y),round(x)); prm(i)=1;
subplot(4,2,7); imagesc(prm); axis image;
axis ij; axis off; colormap(gray);
                                                    %Display 2-D boundary
subplot(4,2,8); plot(th,fapprox,'k.'); axis off;    %Display corresponding signature
s=sprintf('Approximation %d terms',N); title(s);
```

Comments

Calculates the Fourier expansion of the radial signature of the Iberian landmass using DC only, 5 and 15 terms. This uses two user-defined Matlab functions ***fbuild_1D*** and ***fseries_1D*** available on the book's website http://www.fundipbook.com/materials/.

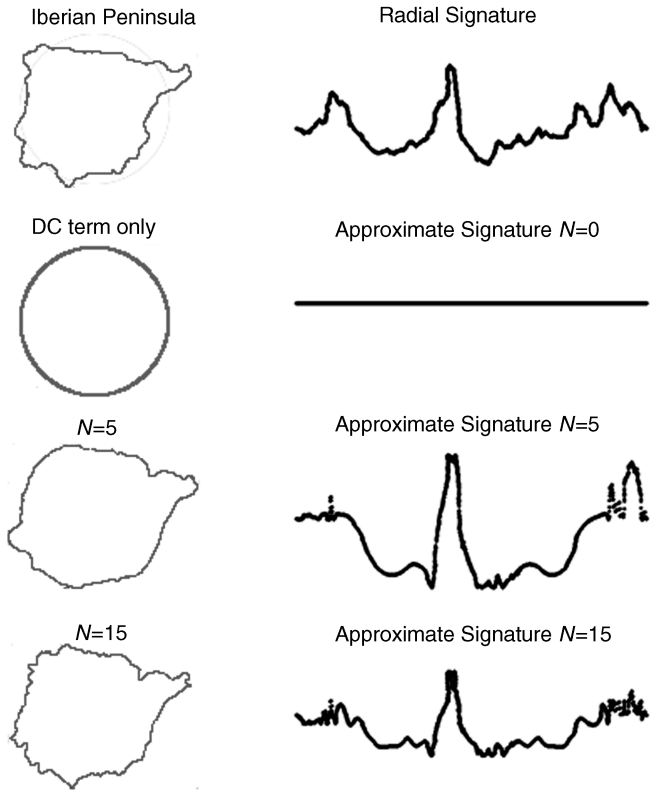


Figure 9.3 An example of the use of the radial Fourier expansion technique to approximate the shape of the Iberian Peninsula using $N = 0$, 5 and 15 terms

The use of radial Fourier expansions can, however, become problematic on complicated boundary shapes, particularly those in which the boundary 'meanders back' on itself (see Figure 9.4). The signature function $r(\theta)$ may not be single valued, there being two or more possible radial values for a given value of θ. In such cases, the choice of which value of $r(\theta)$ to select is somewhat arbitrary and the importance of these unavoidable ambiguities will depend on the specific application. In general, however, strongly meandering boundaries, such as the second example depicted in Figure 9.4, are not suitable for radial Fourier expansion.

As discussed in Chapter 7 on geometry, a robust shape descriptor should be invariant to translation, scaling and rotation. The Fourier descriptors calculated according to Equations (9.2) and (9.3) are certainly translation invariant. This follows because the radial distance in the signature is calculated with respect to an origin defined by the centroid coordinates of the boundary.

Turning to the question of scale invariance, we note the linear nature of the expansion given by Equation (9.1). Multiplication of the signature by an arbitrary scale factor is reflected in the same scale factor multiplying each of the individual Fourier coefficients. A form of scale invariance can thus be achieved most simply by dividing the signature by its maximum value (thus fixing its maximum value as one). Finally, rotational invariance can

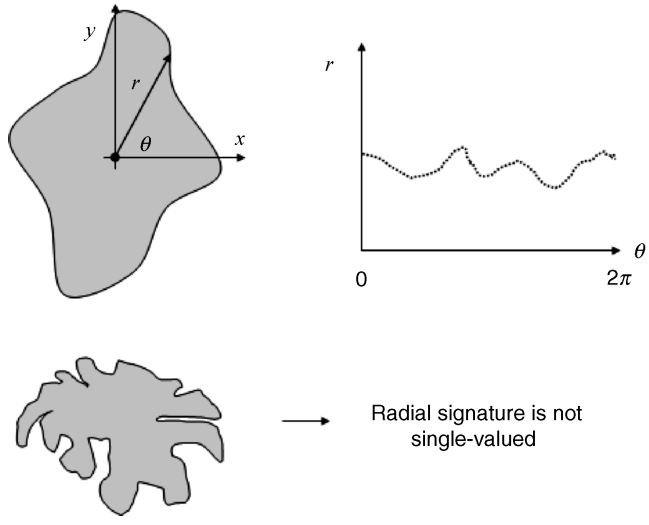


Figure 9.4 The signature is given by the distance from the centroid to the boundary as a function of polar angle. For complex, boundaries (second example) it will not, in general, be single-valued

also be achieved relatively simply as follows. First, note that, for a radial signature, rotational invariance is the same as start-point invariance. In other words, we need the same set of descriptors for a shape irrespective of what point on the boundary we choose to define as zero angle. Consider, then, a radial signature $r(\theta)$ described by a set of complex Fourier coefficients $\{c_n\}$ according to Equation (9.3). Rotation of this shape by some positive, arbitrary angle θ' results in a radial signature $r(\theta - \theta')$. The complex Fourier coefficients for this function, which we distinguish from the unrotated version as $\{c'_n\}$, can be calculated using Equation (9.3) as

$$c'_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} r(\theta - \theta') \exp(-\mathrm{i}n\theta)\, \mathrm{d}\theta \tag{9.4}$$

Making the change of variable $\theta'' = \theta - \theta'$ and substituting in Equation (9.4) gives

$$\begin{aligned} c'_n &= \exp(-\mathrm{i}n\theta') \frac{1}{2\pi} \int_{-\pi}^{\pi} r(\theta'') \exp(-\mathrm{i}n\theta'')\, \mathrm{d}\theta'' \\ &= \exp(-\mathrm{i}n\theta') c_n \end{aligned} \tag{9.5}$$

and we see that the coefficients of the rotated shape differ only by a multiplicative phase factor $\exp(-\mathrm{i}n\theta')$. A rotation-invariant set of descriptors can thus easily be obtained by taking the modulus squared of the coefficients: $\{|c_n|^2 = |c'_n|^2\}$. Such a parameterization can form an effective way of comparing and distinguishing between different shapes for purposes of recognition and classification. The shape itself, however, must clearly be regenerated using the original complex values $\{c'_n\}$.

9.4 Statistical moments as region descriptors

The concept of moments forms an important part of elementary probability theory. If we have a probability density function $p(x)$ which describes the distribution of the random variable x, then the nth moment is defined as

$$m_n = \int_{-\infty}^{\infty} x^n p(x) \mathrm{d}x \tag{9.6}$$

The zeroth moment $m_0 = \int_{-\infty}^{\infty} p(x) \mathrm{d}x$ gives the total area under the function $p(x)$, and is always equal to unity if $p(x)$ is a true probability density function. The first moment, $\mu = m_1 = \int_{-\infty}^{\infty} xp(x)\, \mathrm{d}x$, corresponds to the mean value of the random variable.

The *central* moments of the density function describe the variation about the mean and are defined as

$$\mathrm{M}_n = \int_{-\infty}^{\infty} (x-\mu)^n p(x)\, \mathrm{d}x \tag{9.7}$$

The most common central moment, $\mathrm{M}_2 = \int_{-\infty}^{\infty} (x-\mu)^2 p(x) \mathrm{d}x$, is the well-known *variance* and forms the most basic measure of how 'spread out' the density function is. Higher order moments can yield other information on the shape of the density function, such as the skewness (the tendency to shoot further on one side of the mean than the other). An important theorem of probability theory states that knowledge of all the moments uniquely determines the density function. Thus, we can understand that the moments collectively encode information on the shape of the density function.

Moments extend naturally to 2-D (and higher dimensional) functions. Thus, the pqth moment of a 2-D density function $p(x, y)$ is given by

$$m_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q p(x, y) \mathrm{d}x \mathrm{d}y \tag{9.8}$$

We calculate moments of images, however, by replacing the density function with the intensity image $I(x, y)$ or, in some cases, a corresponding binary image $b(x, y)$. The intensity image is thus viewed as an (unnormalized) probability density which provides the likelihood of a particular intensity occurring at location $I(x, y)$.

Let us also note at this juncture that if the image in question is binary then the moments directly encode information about the *shape*.

Thus, in a completely analogous way to the 1-D case, we can define the $(p-q)$th central moment of our 2-D shape $I(x, y)$ as

$$M_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x-\mu_x)^p (y-\mu_y)^q I(x, y)\, \mathrm{d}x\, \mathrm{d}y \tag{9.9}$$

Since the central moments are measured with respect to the centroid of the shape, it follows that they are necessarily translation invariant. In general, however, we require shape descriptors which will not change when the shape is scaled and/or rotated, i.e. that are also scale and rotation invariant. It is beyond the scope of our immediate discussion to offer a proof, but it is possible to show that *normalized central moments* possess scale invariance.

The $(p-q)$th normalized central moment is defined as

$$\eta_{pq} = \frac{M_{pq}}{M_{00}^{\beta}} \qquad \text{where } \beta = \frac{p+q}{2} + 1 \text{ and } p+q \geq 2 \tag{9.10}$$

From these normalized central moments, it is possible to calculate seven derived quantities attributed to Hu (also referred to as moments) which are invariant to translation, scale and rotation:

$$\begin{aligned}
\Lambda_1 &= \eta_{20} + \eta_{02} \\
\Lambda_2 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
\Lambda_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
\Lambda_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{30})^2 \\
\Lambda_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} - \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{03} + \eta_{21}) \\
&\quad \times [3(\eta_{30} + \eta_{12})^2 - (\eta_{03} + \eta_{21})^2] \\
\Lambda_6 &= (\eta_{20} - \eta_{02})[(\eta_{12} + \eta_{30})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{21} + \eta_{03})(\eta_{12} + \eta_{30}) \\
\Lambda_7 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{03} + \eta_{21})^2] + (3\eta_{21} - \eta_{30})(\eta_{21} + \eta_{03}) \\
&\quad \times [3(\eta_{30} + \eta_{12})^2 - (\eta_{03} + \eta_{21})^2]
\end{aligned} \tag{9.11}$$

Figure 9.5 (produced using the Matlab code in Example 9.3) shows an object extracted from an image along with identically shaped objects which have been translated, scaled and rotated with respect to the first. The values of the first three invariant moments are calculated according to Equation (9.11) for all three shapes in code example 9.3. As is evident, they are equal to within the computational precision allowed by a discrete, digital implementation of the equations in Equation (9.11).

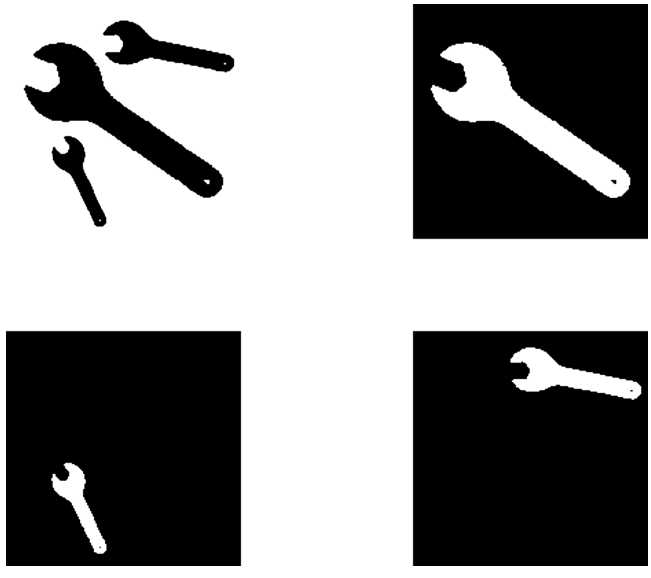


Figure 9.5 Example objects for Hu moment calculation in Example 9.3

Example 9.3

Matlab code	What is happening?
A=rgb2gray(imread('spanners.png'));	%Read in image, convert to grey
bwin=∼im2bw(A,0.5);	%Threshold and display
[L, num]=bwlabel(bwin);	%Create labelled image
subplot (2,2,1), imshow(A);	%Display input image
for i=1:num	%Loop through each labelled object
I=zeros(size(A));	%array for ith object
ind=find(L==i); I(ind)=1;	%Find pixels belonging to ith object and set=1
subplot(2, 2, i+1), imshow(I);	%Display identified object
	%I=double(bw)./(sum(sum(bw)));
[rows,cols]=size(I); x=1:cols;y=1:rows;	%get indices
[X,Y]=meshgrid(x,y);	%Set up grid for calculation
	%calculate required ordinary moments
M_00=sum(sum(I));	
M_10=sum(sum(X.*I)); M_01=sum(sum(Y.*I));	
xav=M_10./M_00; yav=M_01./M_00;	
X=X-xav; Y=Y-yav;	%mean subtract the X and Y coordinates
hold on; plot(M_10,M_01,'ko'); drawnow	
	%calculate required central moments
M_11=sum(sum(X.*Y.*I));	
M_20=sum(sum(X.^2.*I)); M_02=sum(sum(Y.^2.*I));	
M_21=sum(sum(X.^2.*Y.*I)); M_12=sum(sum(X.*Y.^2.*I));	
M_30=sum(sum(X.^3.*I)); M_03=sum(sum(Y.^3.*I));	
	%calculate normalised central moments
eta_11=M_11./M_00.^2;	
eta_20=M_20./M_00.^2;	
eta_02=M_02./M_00.^2;	
eta_21=M_21./M_00.^(5./2);	
eta_12=M_12./M_00.^(5./2);	
eta_30=M_30./M_00.^(5./2);	
eta_03=M_02./M_00.^(5./2);	
	%calculate Hu moments
Hu_1=eta_20 + eta_02;	
Hu_2=(eta_20 - eta_02).^2 + (2.*eta_11).^2;	
Hu_3=(eta_30 - 3.*eta_12).^2 + (3.*eta_21 - eta_03).^2;	
s=sprintf('Object number is %d', i)	
s=sprintf('Hu invariant moments are %f %f %f',Hu_1,Hu_2,Hu_3)	
pause;	
end	

Comments

This example takes an image containing three similar objects at different scale, location and rotation and calculates the first three Hu invariant moments.

9.5 Texture features based on statistical measures

In the simplest terms, texture is loosely used to describe the 'roughness' of something. Accordingly, texture measures the attempt to capture characteristics of the intensity fluctuations between groups of neighbouring pixels, something to which the human eye is very sensitive. Note that texture measures based on statistical measures must generally be defined with respect to a certain neighbourhood Ω which defines the local region over which the calculation is to be made. The simplest such measures are the range defined as

$$\Re_\Omega = \{\max(I(x,y)) - \min(I(x,y))\}_\Omega \tag{9.12}$$

and the local variance defined as

$$\mathrm{Var}_\Omega = \{\langle I^2(x,y)\rangle - \langle I(x,y)\rangle^2\}_\Omega \tag{9.13}$$

where the angle brackets denote averaging over the neighbourhood Ω. By first applying a degree of smoothing and varying the size of Ω, we can attempt to extract multiscale measures of texture.

Figure 9.6 (produced using the Matlab code in Example 9.4) shows an image with two contrasting textures and the results of applying the local range, entropy and variance operators.

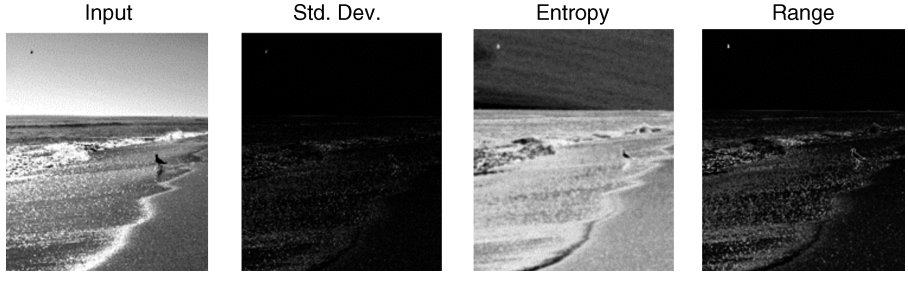


Figure 9.6 Some basic local texture operators: The response to the image is given by applying filters extending over a certain neighbourhood region of the target pixel. Proceeding from left to right: The original input image is followed by the output images resulting from a local operators calculating the standard deviation, entropy and range

Example 9.4

Matlab code	**What is happening?**
A=imread('sunandsea.jpg') ;	%Read image
I=rgb2gray(A);	%Convert to grey-scale
J = stdfilt(I);	%Apply local standard deviaton filter
subplot(1,4,1), imshow(I);	
subplot(1,4,2),imshow(J,[]);	%Display original and processed
J = entropyfilt(I,ones(15));	%Apply entropy filter over 15x15 neighbourhood
subplot(1,4,3),imshow(J,[]);	%Display processed result
J = rangefilt(I,ones(5));	%Apply range filter over 5x5 neighbourhood
subplot(1,4,4),imshow(J,[]);	%Display processed result

Comments
This file shows the effect of three different texture filters based on calculating the variance (standard devn), range and entropy over a locally defined neighbourhood

9.6 Principal component analysis

The reader is probably familiar with the common saying that goes something along the lines of 'Why use a hundred words when ten will do?' This idea of expressing information in its most succinct and compact form embodies very accurately the central idea behind Principal Component Analysis (PCA) a very important and powerful statistical technique. Some of the more significant aspects of digital imaging in which it has found useful application include the classification of photographic film, remote sensing and data compression, and automated facial recognition and facial synthesis, an application we will explore later in this chapter. It is also commonly used as a *low-level* image processing tool for certain tasks, such as determination of the orientation of basic shapes. Mathematically, PCA is closely related to eigenvector/eigenvalue analysis and, as we shall see, can be conveniently understood in terms of the geometry of vectors in high-dimensional spaces.

All the applications to which PCA is put have one very important thing in common: they consider a sequence of images (or speaking more generally, data) which are *correlated*. Whenever we have many examples of data/feature vectors which exhibit a significant degree of correlation with each other, we may consider applying PCA as a means of closely approximating the feature vectors, but using considerably fewer parameters to describe them than the original data. This is termed *dimensionality reduction* and can significantly aid interpretation and analysis of both the sample data and new examples of feature vectors drawn from the same distribution. Performing PCA often allows simple and effective classification and discrimination procedures to be devised which would otherwise be difficult. In certain instances, PCA is also useful as a convenient means to synthesize new data examples obeying the same statistical distribution as an original training sample.

9.7 Principal component analysis: an illustrative example

To grasp the essence of PCA, suppose we have a large sample of M children and that we wish to assess their physical development. For each child, we record exhaustive information on N variables, such as age, height, weight, waist measurement, arm length, neck circumference, finger lengths, etc. It is readily apparent that each of the variables considered here is *not independent* of the others. An older child can generally be expected to be taller, a taller child will generally weigh more, a heavier child will have a larger waist measurement and so on. In other words, we expect that the value of a given variable in the set is, to some degree, *predictive* of the values of the other variables. This example illustrates the primary and essential requirement for PCA to be useful: there *must be some degree of correlation* between the data vectors. In fact, the more strongly correlated the original data, the more effective PCA is likely to be.

To provide some concrete but simple data to work with, consider that we have just $M = 2$ measurements on the height h and the weight w of a sample group of $N = 12$ people as given in Table 9.2. Figure 9.7 plots the mean-subtracted data in the 2-D feature space (w, h) and it is evident that the data shows considerable variance over both

Table 9.2 A sample of 2-D feature vectors (height and weight)

Weight (kg)	65	75	53	54	61	88	70	78	52	95	70	72
Height (cm)	170	176	154	167	171	184	182	190	166	168	176	175

variables and that they are correlated. The sample covariance matrix for these two variables is calculated as:

$$C_x = \frac{1}{N-1}(\mathbf{x} - \bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}})^{\mathrm{T}} \tag{9.14}$$

where $\mathbf{x} = (h \quad w)^{\mathrm{T}}$ and $\bar{\mathbf{x}} = (\bar{h} \quad \bar{w})^{\mathrm{T}}$ is the sample mean and confirms this (see Table 9.3).[4]

The basic aim of PCA is to effect a rotation of the coordinate system and thereby express the data in terms of a new set of variables or equivalently axes which are uncorrelated. How are these found? The first principal axis is chosen to satisfy the following criterion:

> The axis passing through the data points which *maximizes the sum of the squared lengths of the perpendicular projection* of the data points onto that axis is the *principal axis*.

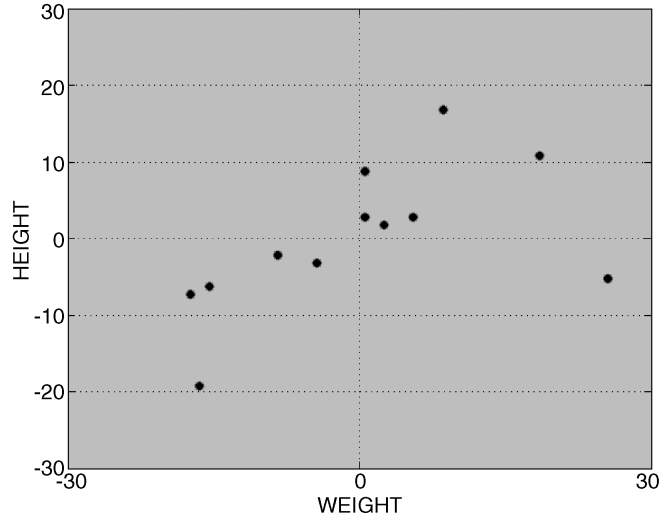


Figure 9.7 Distribution of weight and height values within a 2-D feature space. The mean value of each variable has been subtracted from the data

[4] Note that the correlation matrix is effectively a normalized version of the covariance and given by $\rho_{ij} = C_x(i,j)/\sigma_{ij}$, where $\sigma_{ij} = \langle (x_i - \bar{x}_i)(x_j - \bar{x}_j) \rangle$.

Table 9.3 Summary statistics for data in Table 9.2

	Covariance matrix			Correlation matrix		
	Weight	Height	Mean		Weight	Height
Weight	90.57	73.43	$\overline{w} = 69.42$	Weight	1	0.57
Height	73.43	182.99	$\overline{h} = 173.25$	Height	0.57	1

Thus, the principal axis is oriented so as to maximize the overall variance of the data with respect to it (a variance-maximizing transform). We note in passing that an alternative but entirely equivalent criterion for a principal axis is that it *minimizes* the sum of the squared errors (differences) between the actual data points and their *perpendicular projections* onto the said straight line.[5] This concept is illustrated in Figure 9.8.

Let us suppose that the first principal axis has been found. To calculate the second principal axis we proceed as follows:

- Calculate the projections of the data points onto the first principal axis.
- Subtract these projected values from the original data. The modified set of data points is termed the *residual* data.
- The second principal axis is then calculated to satisfy an identical criterion to the first (i.e. the variance of the residual data is maximized along this axis).

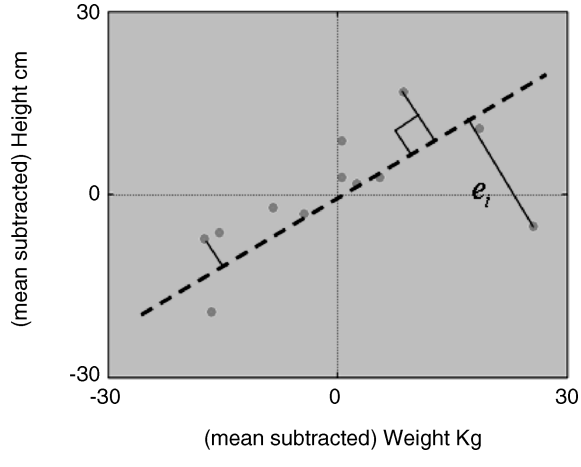


Figure 9.8 The principal axis minimizes the sum of the squared differences between the data and their orthogonal projections onto that axis. Equivalently, this maximizes the variance of the data along the principal axis

[5] Note also that this is *distinct* from fitting a straight line by regression. In regression, x is an independent variable and y the dependent variable and we seek to minimize the sum of the squared errors between the actual y values and their predicted values. In PCA, x and y are on an equal footing.

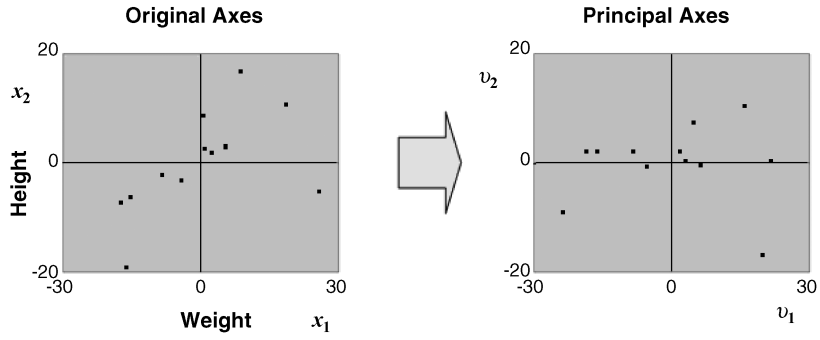


Figure 9.9 The weight–height data of Example 9.5 referred to the original and principal axis coordinate systems. Expressing the data with respect to the principal axes has increased the variance of the data along one axis and (necessarily) reduced it with respect to the other

For a 2-D space, we must stop here (there are no more dimensions left). Figure 9.9 shows our example height–weight data referred to both the original axes and to the principal axes. However, for an N-dimensional feature space (i.e. when N variables exist), this procedure is simply repeated N times to derive N principal axes each of which successively maximizes the variance of the residual data along the given principal axis.

Hopefully, the concept of PCA is clear, but how are these principal axes *actually calculated* ? In the next two sections we will present the basic theory. There are in fact two subtly different viewpoints and thus two different approaches to the calculation of principal components. Both produce the same result, but, as we will see, each offers a slightly different viewpoint on PCA. As this subtlety is of direct relevance to the calculation of principal-component images in image processing, we will present both.

9.8 Theory of principal component analysis: version 1

Assume we have N observations on M variables ($N \geq M$). Let us denote the mean-subtracted observations on the ith variable by x_i ($i = 1, 2, \ldots, N$). The aim is to define a set of new, *uncorrelated* variables y_i which are some appropriate linear combination of the x_i. Thus:

$$y_i = \sum_{j=1}^{M} a_{ij} x_j \tag{9.14}$$

with the coefficients a_{ij} to be determined. The set of M linear equations expressed by Equation (9.14) can be written compactly in matrix form as

$$\vec{y} = \mathbf{R}\vec{x} \tag{9.15}$$

where $\vec{y} = [y_1, y_2 \cdots y_M]^{\mathrm{T}}$, $\vec{x} = [x_1, x_2 \cdots x_M]^{\mathrm{T}}$ and $\mathbf{R}$ is a matrix of unknown coefficients. Note that the variables x_i have had their sample means subtracted and are zero-mean variables, so that $\langle \vec{y} \rangle = \mathbf{R}\langle \vec{x} \rangle = 0$. By definition, the covariance matrix $C_{\vec{y}}$ on the new variables represented by $\vec{y}$ is then

$$C_{\vec{y}} = \langle \vec{y}\vec{y}^{\mathrm{T}} \rangle = \mathbf{R}\langle \vec{x}\vec{x}^{\mathrm{T}} \rangle \mathbf{R}^{\mathrm{T}} = \mathbf{R} C_{\vec{x}} \mathbf{R}^{\mathrm{T}} \tag{9.16}$$

where the angle brackets denote ensemble averaging over the observations on the data vector. We demand that the new variables be uncorrelated. This is equivalent to requiring that the covariance matrix $C_{\vec{y}}$ on the new variables is *diagonal*. Accordingly, we must form the covariance matrix $C_{\vec{x}} = \langle \vec{x}\vec{x}^{\mathrm{T}} \rangle$ on our original data and find a diagonalizing matrix $\mathbf{R}$ such that

$$C_{\vec{y}} = \langle \vec{y}\vec{y}^{\mathrm{T}} \rangle = \mathbf{R} C_{\vec{x}} \mathbf{R}^{\mathrm{T}} = \Lambda \text{ a diagonal matrix} \tag{9.17}$$

Fortunately, finding such a matrix is easy. Equation (9.16) describes the classic eigenvalue/eigenvector problem,[6] which can be solved by standard numerical procedures. The columns of $\mathbf{R}$ are the set of orthogonal eigenvectors which achieve the diagonalization of $C_{\vec{x}}$ and the diagonal elements of Λ give the corresponding eigenvalues. Computationally, the solution steps are as follows:

- Form the sample covariance matrix $C_{\vec{x}} = \langle \vec{x}\vec{x}^{\mathrm{T}} \rangle$.
- Provide this as the input matrix to an eigenvector/eigenvalue decomposition routine which will return the eigenvectors and eigenvalues in the matrices $\mathbf{R}$ and Λ respectively.
- The new variables (axes) are then readily calculable using Equation (9.15).

From this viewpoint, PCA has effected a rotation of the original coordinate system to define new variables (or, equivalently, new axes) which are linear combinations of the original variables. We have calculated the *principal axes*.

Note that the data itself can be expressed in terms of the new axes by exploiting the orthonormality of the eigenvector matrix (i.e. it consists of unit length, mutually orthogonal vectors such that $\mathbf{R}\mathbf{R}^{\mathrm{T}} = \mathbf{R}^{\mathrm{T}}\mathbf{R} = \mathbf{I}$). Multiplying Equation (9.15) from the left by $\mathbf{R}^{\mathrm{T}}$, we thus have an expression for the data in the new system:

$$\vec{x} = \mathbf{R}^{\mathrm{T}}\vec{y} \tag{9.18}$$

9.9 Theory of principal component analysis: version 2

In this second approach to the PCA calculation, we take an alternative viewpoint. We consider each sample of N observations on each of the M variables to form our basic data/

[6] More commonly written as $\mathbf{R}C_{\vec{x}} = \Lambda\mathbf{R}$ with Λ a diagonal matrix

observation vector. The M data vectors $\{\vec{\mathbf{x}}_i\}$ thus exist in an N-dimensional space and each 'points to a particular location' in this space. The more strongly correlated the variables are, it follows that the more similar will be the pointing directions of the data vectors. We propose new vectors $\vec{\mathbf{v}}_j$ which are linear combinations of the original vectors $\{\vec{\mathbf{x}}_i\}$. Thus, in tableau form, we may write

$$\begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ \vec{\mathbf{v}}_1 & \vec{\mathbf{v}}_2 & \cdots & \vec{\mathbf{v}}_M \\ \downarrow & \downarrow & & \downarrow \end{bmatrix} = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ \vec{\mathbf{x}}_1 & \vec{\mathbf{x}}_2 & \cdots & \vec{\mathbf{x}}_M \\ \downarrow & \downarrow & & \downarrow \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1M} \\ a_{21} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{M1} & a_{M2} & & a_{MM} \end{bmatrix} \quad \text{or} \quad \mathbf{V} = \mathbf{XR} \tag{9.19}$$

and demand that the $\vec{\mathbf{v}}_j$ are mutually orthogonal. The task is to find the matrix of coefficients $\mathbf{R}$ which will accomplish this task and, therefore, we require that the new vectors satisfy

$$\mathbf{V}^{\mathrm{T}}\mathbf{V} = [\mathbf{XR}]^{\mathrm{T}}[\mathbf{XR}] = \Lambda \text{ a diagonal matrix}$$

Multiplying both sides by the factor $1/(N-1)$ and rearranging we have

$$\frac{1}{N-1}\mathbf{V}^{\mathrm{T}}\mathbf{V} = \mathbf{R}^{\mathrm{T}}\frac{[\mathbf{X}^{\mathrm{T}}\mathbf{X}]}{N-1}\mathbf{R} = \Lambda \text{ a diagonal matrix} \tag{9.20}$$

Note that the insertion on both sides of the factor $1/(N-1)$ is not strictly required but is undertaken as it enables us to identify terms on the left- and right-hand sides which are sample covariance matrices on the data, i.e. $\mathbf{C_X} = (N-1)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{X}$. Thus, we require a matrix $\mathbf{R}$ such that

$$\mathbf{R}^{\mathrm{T}}\mathbf{C_X}\mathbf{R} = \frac{1}{N-1}\mathbf{V}^{\mathrm{T}}\mathbf{V} = C_V = \Lambda \text{ a diagonal matrix} \tag{9.21}$$

and we again arrive at a solution which requires the eigenvector/eigenvalue decomposition of the data covariance matrix $\mathbf{C_X} = (N-1)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{X}$.[7] Computationally, things are the same as in the first case:

- Form the data covariance matrix $\mathbf{C_X} = (N-1)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{X}$.
- Perform an eigenvector/eigenvalue decomposition of $\mathbf{C_X}$, which will return the eigenvectors and eigenvalues in the matrices $\mathbf{R}$ and Λ respectively.
- Calculate the new data vectors in matrix $\mathbf{V}$ using Equation (9.18). Note that the columns of matrix $\mathbf{V}$ thus obtained are the *principal components*.

[7] That this covariance is formally equivalent to that version given earlier is readily demonstrated by writing out the terms explicitly.

The data itself can be expressed in terms of the principal components by noting that the eigenvector matrix is *orthonormal* (i.e. $\mathbf{R}\mathbf{R}^{\mathrm{T}} = \mathbf{R}^{\mathrm{T}}\mathbf{R} = \mathbf{I}$). Multiplying Equation (9.19) from the left by $\mathbf{R}^{\mathrm{T}}$ then gives the original data as $\mathbf{X} = \mathbf{V}\mathbf{R}^{\mathrm{T}}$.

9.10 Principal axes and principal components

There is a subtle difference between the two alternative derivations we have offered. In the first case we considered our variables to define the feature space and derived a set of principal axes. The dimensionality of our feature space was determined by the number of variables and the diagonalizing matrix of eigenvectors $\mathbf{R}$ is used in Equation (9.13) to produce a new set of axes in that 2-D space. Our second formulation leads to essentially the same procedure (i.e. to diagonalize the covariance matrix), but it nonetheless admits an alternative viewpoint. In this instance, we view the observations on each of the variables to define our (higher dimensional) feature vectors. The role of the diagonalizing matrix of eigenvectors $\mathbf{R}$ here is thus to multiply and transform these higher dimensional vectors to produce a new orthogonal (principal) set. Thus, the dimensionality of the space here is determined by *the number of observations* made on each of the variables.

Both points of view are legitimate. We prefer to reserve the term principal components for the vectors which constitute the columns of the matrix $\mathbf{V}$ calculated according to our *second* approach. Adopting this convention, the principal components are in fact formally equivalent to the projections of the original data onto their corresponding principal axes.

9.11 Summary of properties of principal component analysis

It is worth summarizing the key points about the principal axes and principal components.

- The principal axes are *mutually orthogonal*. This follows if we consider how they are calculated. After the first principal axis has been calculated, each successive component is calculated on the residual data (i.e. after subtraction of the projection onto the previously calculated axis). It follows, therefore, that the next component can have no component in the direction of the previous axis and must, therefore, be orthogonal to it.

- The data expressed in the new (principal) axes system is uncorrelated. The new variables do not co-vary and, thus, their covariance matrix has zero entries in the off-diagonal terms.

- The principal axes successively *maximize* the variance in the data with respect to themselves.

- The eigenvalues in the diagonal matrix Λ directly specify the total amount of variance in the data associated with each principal component.

Example 9.5 calculates the principal axes and principal components for the height and weight measurements provided earlier. Figure 9.10 (in part shown in Figure 9.9) summarizes the results.

Example 9.5

Matlab code	**What is happening?**
W=[65 75 53 54 61 88 70 78 52 95 70 72]'	%1. Form data vector on weight
H=[170 176 154 167 171 184 182 190 166 168 176 175]'	%Form data vector on height
XM=[mean(W).*ones(length(W),1) mean(H).*ones(length(H),1)]	%matrix with mean values replicated
X=[W H]-XM;	%Form mean-subtracted data matrix
Cx=cov(X)	%Calculate covariance on data
[R,LAMBDA,Q]=svd(Cx)	%Get eigenvalues LAMBDA and %eigenvectors R
V=X*R;	%Calculate principal components
subplot(1,2,1), plot(X(:,1),X(:,2),'ko'); grid on;	%2. Display data on original axes
subplot(1,2,2), plot(V(:,1),V(:,2),'ro'); grid on;	%Display PCs as data in rotated space
XR=XM + V*R'	%3. Reconstruct data in terms of PCs
XR-[W H]	%Confirm reconstruction (diff = 0)
V'*V./(length(W)-1)	%4. Confirm covariance terms on %New axes = LAMBDA

Comments

Matlab functions: ***cov***, ***mean***, ***svd***.

This example (1) calculates the principal components of the data, (2) displays the data in the new (principal) axes, (3) confirms that the original data can be 'reconstructed' in terms of the principal components and (4) confirms that the variance of the principal components is equal to the eigenvalues contained in matrix LAMBDA.

Note that we have used the Matlab function ***svd*** (singular value decomposition) to calculate the eigenvectors and eigenvalues rather than ***eig***. The reason for this is that ***svd*** orders the eigenvalues (and their associated eigenvectors) *strictly according to size* from largest to smallest (***eig*** does not). This is a desirable and natural ordering when calculating principal components which are intended to successively describe the maximum possible amount of variance in the data.

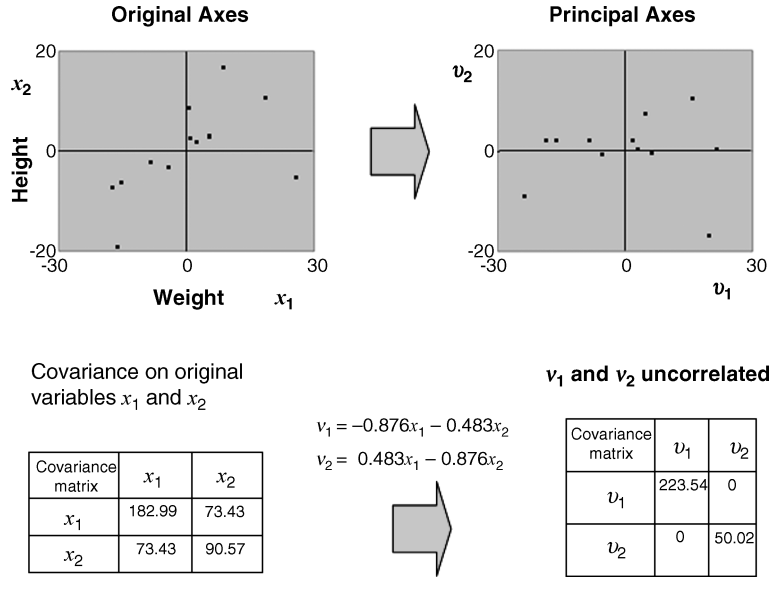


Figure 9.10 The weight–height data referred to the original axes x_1 and x_2 and to the principal axes v_1 and v_2

Example 9.6 shows the basic use of PCA in the calculation of the principal axes of a 2-D shape (the silhouette of an aeroplane). Identification of the principal axes, as shown in Figure 9.11 of a 2-D or 3-D shape, can sometimes assist in applying processing operations such that they are effectively invariant to image rotation.

Example 9.6

Matlab code	**What is happening?**
A=imread('aeroplane_silhouette.png');	%Read in image and convert to grey scale
bw=~im2bw(A,0.5);	%Threshold and invert
subplot(1,2,1), imshow(bw,[]);	%Display image
[y,x]=find(bw>0.5);	%Get coordinates of non-zero pixels
centroid=mean([x y]);	%Get (centroid) of data
hold on; plot(centroid(1),centroid(2),'rd');	%Plot shape centroid
C=cov([x y]);	%Calculate covariance of coordinates
[U,S]=eig(C)	%Find principal axes and eigenvalues
	%Plot the principal axes
m=U(2,1)./U(1,1);	
const=centroid(2)-m.*centroid(1);	
xl=50:450; yl=m.*xl + const	
subplot(1,2,2), imshow(bw,[]); h=line(xl,yl);	%Display image and axes
set(h,'Color',[1 0 0],'LineWidth',2.0)	
m2=U(2,2)./U(1,2);	
const=centroid(2)-m2.*centroid(1);	
x2=50:450; y2=m2.*x2 + const	

```
h=line(x2,y2); set(h,'Color',[1 0 0],
    'LineWidth',2.0)
```

Comments

This example calculates the principal axes of the foreground pixels in the binary image of the aeroplane silhouette.

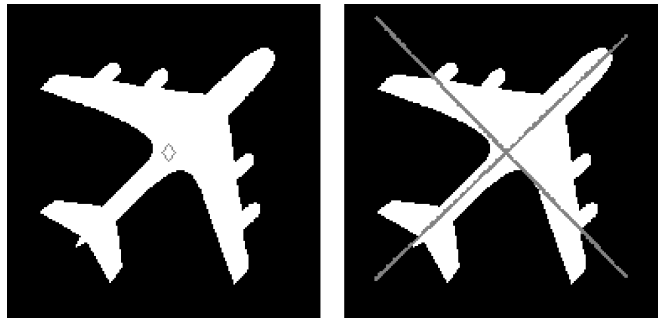


Figure 9.11 Calculation of the principal axes of a 2-D object (see Example 9.6 for calculation)

9.12 Dimensionality reduction: the purpose of principal component analysis

Depending on the precise context, there may be several closely related reasons for doing PCA, but the aim central to most is *dimensionality reduction*. By identifying a dominant subset of orthogonal directions in our feature space we can effectively discard those components which contribute little variance to our sample and use only the significant components to approximate accurately and describe our data in a much more compact way. To take our 2-D height–weight problem as an example, the basic question we are really asking is: 'Do we really need two variables to describe a person's basic physique or will just one variable (what one might call a 'size index') do a reasonable job? We can answer this question by calculating the principal components. In this particular example, a single index (the value of the first principal component) accounts for 84 % of the variation in the data. If we are willing to disregard the remaining 16 % of the variance we would conclude that we do, indeed, need just one variable. It should be readily apparent that the more closely our height–weight data approximates to a straight line, the better we can describe our data using just one variable. Ultimately, if the data lay exactly on a straight line, the first principal axis then accounts for 100 % of the variance in the data.

Our height–weight example is, perhaps, a slightly uninspiring example, but it hopefully makes the point. The *real power* of PCA becomes apparent in multidimensional problems, where there may be tens, hundreds or even thousands of variables. Such, indeed, is often the case when we are considering digital images. Here, a collection of primitive image features, landmarks or even the pixel values themselves are treated as variables which vary

over a sample of correlated images. We will shortly present such an example in Section 9.15. In the next section, we first make an important note on how PCA on image ensembles is carried out. This is important if we are to keep the computational problem to a manageable size.

9.13 Principal components analysis on an ensemble of digital images

We can generally carry out PCA in two senses:

(1) by taking our statistical average over the ensemble of vectors;

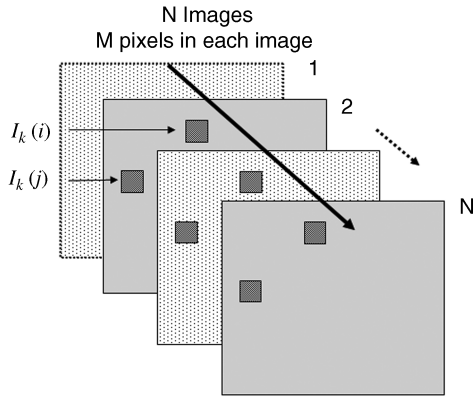
(2) by taking our statistical average over the elements of the vectors themselves.

This point has rarely been made explicit in existing image-processing textbooks. Consider Figure 9.12, in which we depict a stack of M images each of which contains N pixels. In one possible approach to PCA the covariance is calculated on a *pixel-to-pixel* basis. This is to say that the ijth element of our covariance matrix is calculated by examining the values of the ith and jth pixels in each image, $I(i, j)$, and averaging over the whole sample of images. Unfortunately, such an approach will lead to the calculation of a covariance matrix of quite unmanageable size (consider that even a very modest image size of 256^2 would then require calculation and diagonalization of a covariance matrix of $256^2 \times 256^2 \sim 4295 \times 10^6$ elements). However, the second approach, in which the ensemble averaging takes place over the pixels of the images, generally results in a much more tractable problem – the resulting covariance matrix has a square dimension equal to the number of images in the ensemble and can be diagonalized in a straightforward fashion.[8]

9.14 Representation of out-of-sample examples using principal component analysis

In the context of pattern recognition and classification applications, we speak of *training data* (the data examples used to calculate the principal components) and *test data* (new data examples from the same basic pattern class that we are interested in modelling). So far, we have only explicitly discussed the capacity of PCA to produce a reduced-dimension representation of training data. An even more important aspect of PCA is the ability to model a certain class of object so that new (*out-of-sample*) examples of data can be accurately and efficiently represented by the calculated components. For example, if we calculate the principal components of a sample of human faces, a reasonable hope

[8] The derivation and calculation of principal components for image ensembles is covered in full detail on the book's website at http://www.fundipbook.com/materials/.

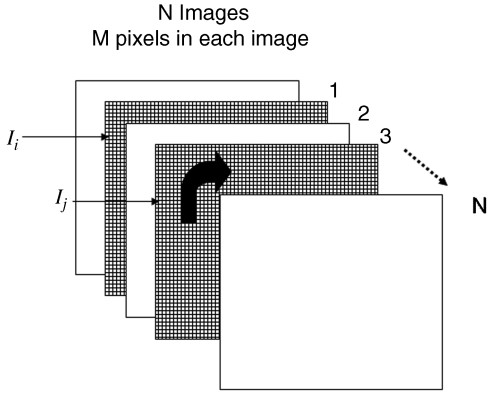


- Averaging takes place over the total number of images
- Covariance dimension = number of pixels squared

$\bar{I}_k$ denotes the average value of the kth image

$I_k(q)$ is the value of the q^{th} pixel in the k^{th} image

$$\text{Covariance } C_{ij} = \frac{1}{N-1}\sum_{n=1}^{N}\left(I_i(n)-\bar{I}_i\right)\left(I_j(n)-\bar{I}_j\right)$$



- Averaging takes place over the all the pixels in each image pair
- Covariance dimension = number of images squared

$\bar{I}_k$ denotes the average value of the kth image

$I_k(m)$ is the value of the m^{th} pixel in the k^{th} image

$$\text{Covariance } C_{ij} = \frac{1}{M-1}\sum_{m=1}^{M}\left(I_i(m)-\bar{I}_i\right)\left(I_j(m)-\bar{I}_j\right)$$

Figure 9.12 Two ways of carrying out PCA on image ensembles. Only the second approach in which the statistical averaging takes place over the image pixels is computationally viable.

is that we might be able to represent *new* unseen faces using these principal components as a basis.

Representing an out-of-sample data vector in terms of a set of precalculated principal components is easily achieved by projection of the data onto the principal component axes. Consider an arbitrary vector $\boldsymbol{I}$ having mean pixel value $\bar{\boldsymbol{I}}$. We wish to express this as a linear combination of the principal components:

$$\boldsymbol{I} - \bar{\boldsymbol{I}} = \sum_{k=1}^{N} a_k \boldsymbol{P_k} \tag{9.22}$$

To do this we can simply exploit the orthogonality of the $\{P_k\}$. Multiplying from the left by $\boldsymbol{P}_i^{\mathrm{T}}$, since $\vec{\boldsymbol{P}}_i^{\mathrm{T}}\vec{\boldsymbol{P}}_k = \lambda_k\delta_{ki}$, we have

$$\boldsymbol{P}_i^{\mathrm{T}}(\boldsymbol{I}-\bar{\boldsymbol{I}}) = \sum_{k=1}^{N} a_k \boldsymbol{P}_i^T \boldsymbol{P}_k = \sum_{k=1}^{N} a_k \lambda_k \delta_{ki} \tag{9.23}$$

And we thus obtain the required expansion coefficients as

$$\hat{a}_i = \frac{\boldsymbol{P}_i^{\mathrm{T}}(\boldsymbol{I}-\bar{\boldsymbol{I}})}{\lambda_i} \tag{9.24}$$

and the vector $\boldsymbol{I}$ is estimated as

$$\hat{\boldsymbol{I}} = \bar{\boldsymbol{I}} + \sum_{k=1}^{N} \hat{a}_k \boldsymbol{P}_k \tag{9.25}$$

Note that, in a really well-trained model, it should be possible to represent new test data just as compactly and accurately as the training examples. This level of perfection is not often achieved. However, a good model will allow a close approximation of all possible examples of the given pattern class to be built using this basis.

9.15 Key example: eigenfaces and the human face

It is self-evident that the basic placement, size and shape of human facial features are similar. Therefore, we can expect that an ensemble of suitably scaled and registered images of the human face will exhibit fairly strong correlation. Figure 9.13 shows six such images from a total sample of 290. Each image consisted of 21 054 grey-scale pixel values.

Figure 9.13 A sample of human faces scaled and registered such that each can be described by the same number of pixels (21 054)

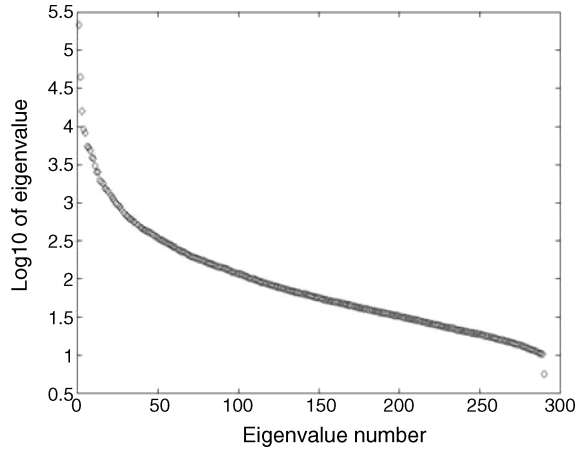


Figure 9.14 The eigenvalue spectrum (on a log scale) after calculation of the principal components of the sample of 290 faces, examples of which are shown in Figure 9.15

When we carry out a PCA on this ensemble of faces, we find that the eigenvalue spectrum dies off quite rapidly. This suggests considerable potential for compression.

The first six facial principal components are displayed in Figure 9.14.

The potential use of PCA for face encoding and recognition has now been studied quite extensively. The essence of the technique is that the face of any given individual may be synthesized by adding or subtracting a multiple of each principal component. In general, a linear combination of as few as 50–100 'eigenfaces' is sufficient for human and machine recognition of the synthesized face. This small code (the weights for the principal components) can be readily stored as a very compact set of parameters occupying $\sim$50 bytes. Figure 9.16 shows the increasing accuracy of the reconstruction as more principal

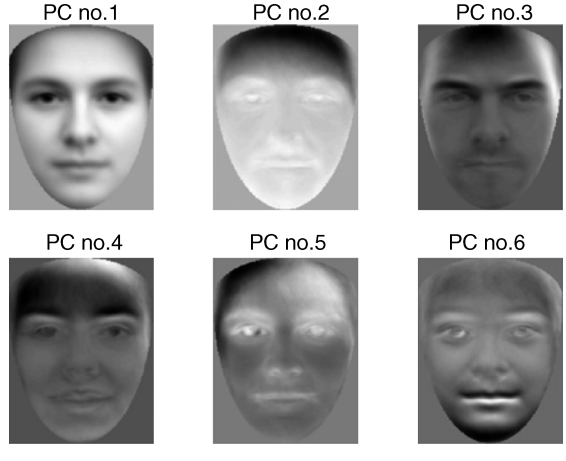


Figure 9.15 The first six facial principal components from a sample of 290 faces. The first principal component is the average face. Note the strong male appearance coded by principal component no. 3

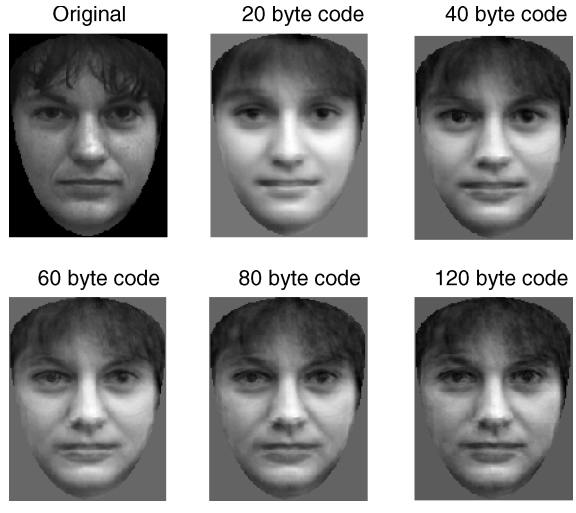


Figure 9.16 Application of PCA to the compact encoding of out-of-sample human faces

Figure 9.17 Example of an EFIT-V. The facial likeness (right) was created by an evolutionary search to find the dominant principal components in the real face (left) based on an eyewitness's memory. The image detail on the right has been slightly enhanced by use of a computer art package. (Image courtesy of Visionmetric Ltd, www.visionmetric.com)

components are used to approximate the face. Note that the original subject displayed was *not included* in the ensemble used to form the covariance matrix.

The use of PCA modelling of human faces has recently been extended to a new generation of facial composite systems (commonly known as PhotoFIT systems)[9] . These systems are used by police forces in an attempt to create a recognisable likeness of a suspect in a crime based on an eyewitnesss memory. It is hoped that subsequent exposure of the image to the public might then result in positive identification. The basic idea behind these systems is that the face can be synthesized by guiding the witness to evolve an appropriate combination of shape and texture principal components. As the witness is working from memory, this forms a fundamental limitation on the accuracy of the constructed composite. Nonetheless, the global nature of the principal components which carry information about the whole face appears to offer real advantages. Likenesses created in this way can be remarkably good and are often achieved much more quickly than by traditional police methods. Figure 9.17 shows one such example in which a photo of the suspect is compared to the likeness created using a PCA-based, evolutionary system. The image displayed shows the likeness after a small degree of artistic enhancement which is commonplace in this application.

For further examples and exercises see http://www.fundipbook.com

[9] See for example the EFIT-V system www.visionmetric.com.