Chapter 3

Editorial Introduction

Paradoxically, some of the so-called “high-level” operators for image understanding are actually concerned with apparently simple concepts. Their apparent simplicity can be deceiving. Consider, for example, the relationship “is to the left of”. This is difficult to express in a computing language such as Java, C or Basic, although it is easy to define it in Prolog. Other abstract relationships of this type are “inside”, “next to”, “between”, etc., and can also be represented conveniently in this language. When we are discussing natural products, abstract relationships between symbolic objects become important, because they allow us to escape from the confinements imposed by always working with precise numeric (e.g. geometric position) data. People do this all the time in everyday conversation. For example, we may note that a person has a mole on the left cheek. We do not necessarily want to spend a long time discussing what a mole is, what it looks like, or exactly where the limits of that person’s left cheek are. Suppose that somebody says “The mole on Mary’s left cheek is getting bigger.” There are far more serious issues to consider than being pre-occupied with irrelevancies that would only obscure the sinister nature of this statement. There are no precise rules for identifying features like moles, eyes on potatoes, bird-pecks on apples, etc., so we may need to devise a vision system that can learn to do so. Anatomical features of an animal or plant are also ill-defined, and hence are often difficult to recognise. In situations like these, where precise terms cannot always be applied, we must resort to using techniques borrowed from Fuzzy Logic, Artificial Intelligence or Pattern Recognition

Chapter 3

Intelligent Image Processing

B.G. Batchelor

If it's green, we reject it.
If it's too ripe, we reject it.
If it's bruised, we reject it.
If it's diseased, we reject it.
If it's dirty, we reject it.
If it's just right, we squash it.

Advertisement, McDonald's Restaurants, 1992

3.1 Why We Need Intelligence

An introduction to the basic concepts and techniques of image processing is provided in the previous chapter. That alone would be adequate for a book describing engineering applications of Machine Vision but the inspection of natural products requires the use of techniques that are more flexible and intelligent. In this chapter, we shall therefore concentrate on additional methods needed to enhance the basic picture manipulation and measurement functions just described. We shall discuss computational techniques, borrowed from Pattern Recognition and Artificial Intelligence, that are appropriate for inspecting highly variable products. These two subjects, together with Computer Vision, all have an important contribution to make to Machine Vision when it is applied to natural product inspection. However, we must emphasise that, while we are happy to use "borrowed" techniques, we do not accept the ethos of any of these science-based subjects; our roots are very firmly established in engineering. In other words, pragmatism and systems-related issues outweigh theoretical ones that have little or no practical significance.

3.2 Pattern Recognition

Nowadays, *Pattern Recognition* is often mistakenly equated solely with the study of *Neural Networks*. In fact, this provides a limited view, since Pattern Recognition sometimes employs other types of decision-making and analysis procedures. For

example, the Pattern Recognition methods that we describe for colour recognition could not legitimately be termed Neural Networks. The simple (traditional) model of a Pattern Recognition system is explained in Figure 3.1 and will be refined later.

3.2.1 Similarity and Distance

In Pattern Recognition, it is common practice to describe a pattern (e.g., an image, or feature within an image, an acoustic signal, a medical patient, the state of an engine, or machine tool) in terms of a fixed number of measurements, taken in parallel (Figure 3.1). This might be appropriate as a way of representing texture, colour, object shape, or some ill-defined feature in an image. Of course, we cannot always conveniently and efficiently describe image features in this way. For this reason, this approach is relatively uncommon in most existing industrial Machine Vision systems. However, this data format is widely used in Pattern Recognition systems for applications as widely varied as insurance-risk assessment, investment planning, differential diagnosis in medicine, electro-encephalography, analysing seismic events, speech recognition, fault detection in machine tools, biological classification, etc. It can also be used to recognise varieties of seeds, colours of fruit, shapes of leaves, etc.
Let us denote a set of N parallel measurements representing a pattern P by the N-dimensional vector

$$\mathbf{X}_P = (X_{P,1}, X_{P,2}, X_{P,3}, \ldots, X_{P,N}) \tag{3.1}$$

(Since this a vector, rather than a set, the order in which the elements $X_{P,1}, X_{P,2},\ldots$ are written is critical.) It is reasonable to assume that, if two patterns, say P and Q are subjectively similar, then their corresponding vectors $\mathbf{X}_P$ and $\mathbf{X}_Q$ will also be similar. Of course, this presupposes that we have managed to find parameters that adequately describe the salient features of the pattern P. The *dissimilarity* between the vectors $\mathbf{X}_P$ and $\mathbf{X}_Q$ can be measured by one of the following distance metrics:

Euclidean distance

$$D_e(\mathbf{X}_P, \mathbf{X}_Q) = \sqrt{\left[\sum (X_{P,i} - X_{Q,i})^2\right]} \tag{3.2}$$

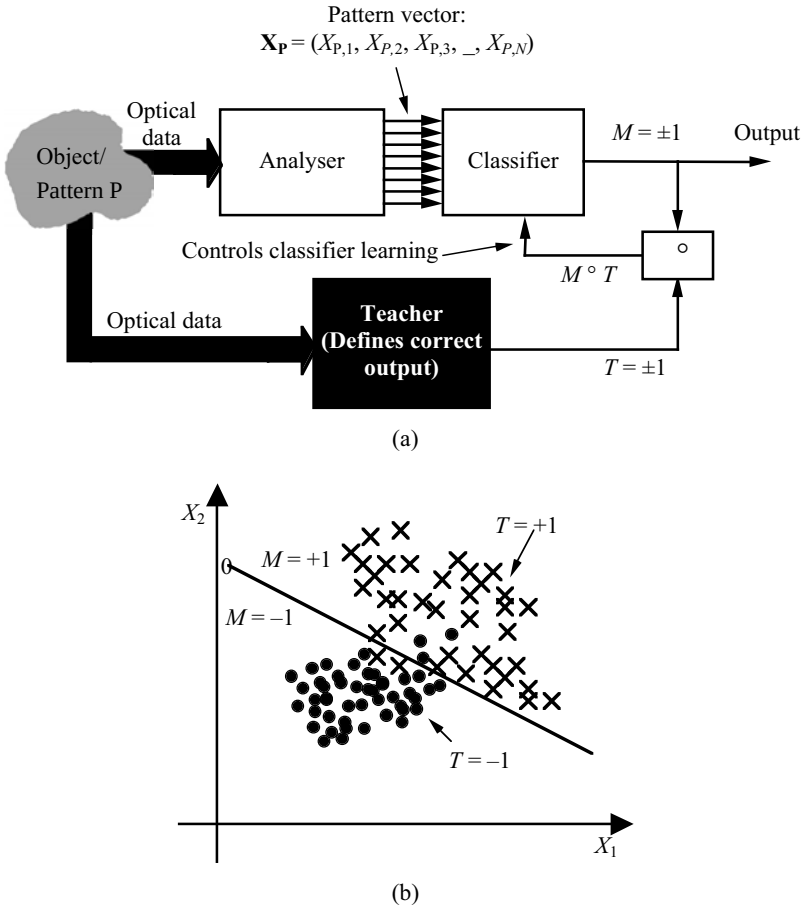


Figure 3.1. Traditional model of a Pattern Recognition system: (a) organisation of a system employing error-correction learning: (b) measurement space. In this simple case $N = 2$ and the machine decision (M) is computed by deciding whether a point falls above or below a straight line.

Square distance

$$D_s(\mathbf{X_P}, \mathbf{X_Q}) = \boldsymbol{MAX}(|(X_{P,i} - X_{Q,i})| \tag{3.3}$$

Manhatten (or City Block) distance

$$D_m(\mathbf{X_P}, \mathbf{X_Q}) = \sum|(X_{P,i} - X_{Q,i})| \tag{3.4}$$

Minkowski r-distance

$$D_r(\mathbf{X}_P, \mathbf{X}_Q) = \left[\sum (X_{P,i} - X_{Q,i})^2\right]^{1/r} \tag{3.5}$$

The parameter r is a positive integer. It is of interest to note that the Minkowski r-distance is able to model the other three metrics; it is identically equal to the Manhatten distance when $r = 1$; the Euclidean distance when $r = 2$; and approaches the Square distance as r tends to infinity.

There are three essential criteria for a distance metric that apply to all of the above formulae and any linear combination of them:

$$D(\mathbf{A}, \mathbf{A}) = 0 \tag{3.6}$$

$$D(\mathbf{X}_P, \mathbf{X}_Q) \geq 0 \tag{3.7}$$

$$D(\mathbf{A}, \mathbf{C}) \leq D(\mathbf{A}, \mathbf{B}) + D(\mathbf{B}, \mathbf{C}) \tag{3.8}$$

Which of these four distance measurements is best? In many cases, it does not matter which one we choose. Other factors, such as ease of implementation then become more important. The Euclidean distance is the most familiar to us, as it is the one that we can derive in two- or three-dimensional space, using a tape measure. It also has certain theoretical advantages over the others, in some situations [1]. The City Block distance is important when we drive or walk through New York City, where the streets are laid out on a grid. This and the Square distance were originally introduced into Pattern Recognition research in the 1960s, principally to avoid the difficulties that then existed in calculating the Euclidean distance. Nowadays, it is usually best to use the Euclidean distance to avoid the theoretical dangers associated with the City Block and Square distances.[1] Notice that, in many situations, it is not necessary to perform the square-root operation implicit in the Euclidean distance, $D_e(X,Y)$.

3.2.2 Compactness Hypothesis

The so-called *Compactness Hypothesis* relates similarity between patterns P and Q to the inverse of the distance between their corresponding vectors, $\mathbf{X}_P$ and $\mathbf{X}_Q$ [1]. Thus, in order to identify (classify) an unknown pattern $\mathbf{X}$, we might reasonably compare it to a set of stored vectors, $S = \{\mathbf{X}^1, \mathbf{X}^2, \mathbf{X}^3, \ldots\}$, representing patterns $\{\mathbf{P}^1, \mathbf{P}^2, \mathbf{P}^3, \ldots\}$, which belong to classes $\{\mathbf{C}^1, \mathbf{C}^2, \mathbf{C}^3, \ldots\}$. A vector $\mathbf{X}$ obtained from a pattern that we wish to classify is then identified with the pattern class associated with the closest neighbour in S. That is, we find i such that $D(\mathbf{X},\mathbf{X}^i)$ is a minimum and then associate $\mathbf{X}$ with class $\mathbf{C}^i$. Such a classifier is called a *Maximum Similarity* or, more commonly, a *Nearest Neighbour Classifier* (Figure 3.2) [1].

[1] When the City Block, Square and Minkowski ($r \neq 2$) distances are used within a Nearest Neighbour Classifier, the decision surface is unstable.

(This is not a Neural Network in the strict sense but could be implemented by one.) The so-called *Compound Classifier* is closely related to the Nearest Neighbour Classifier and makes decisions by measuring distances from a (small) set of stored points. A wide variety of other methods for performing pattern classification, including *Neural Networks*, has been devised (Figure 3.3) [2]. They differ in the way that they combine the elements of the pattern vector. Conceptually, the Nearest Neighbour and Compound Classifiers are among the simplest. They are just as versatile as Neural Networks. Indeed, the Compound Classifier is more natural for certain types of application (Figures 3.4 and 3.5) [1].

3.2.3 Pattern Recognition Models

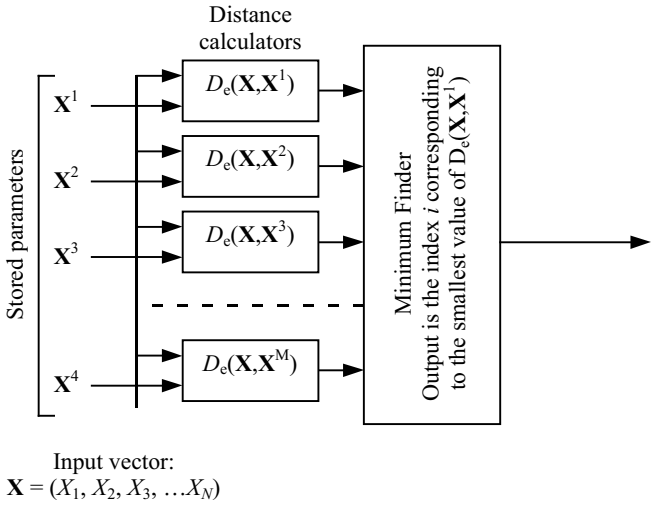
In this section, we shall see that the traditional model for Pattern Recognition is not always appropriate for the types of applications that interest us. Figure 3.5 illustrates various situations in which the original paradigm, expressed in Figure 3.1, is unsatisfactory.

Traditional Model

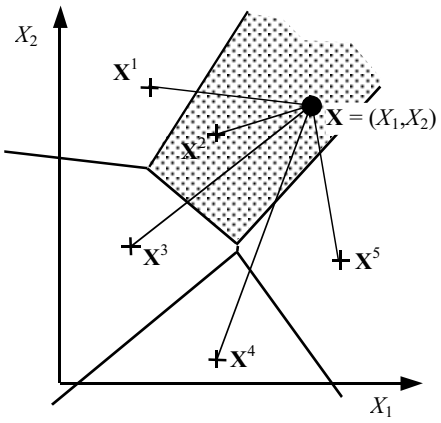
A classifier is simply a mathematical formula, for calculating a decision (M) given a vector of parallel measurements: $\mathbf{X} = (X_1, X_2, \ldots, X_N)$. In most applications, we do not know what function to use to compute M and we must derive an estimate by self-adaptive learning, based on feedback (Figure 3.1). Traditionally, the parameters of a classifier are adjusted if the decision (M) that it has just computed in response to an input $\mathbf{X} = (X_1, X_2, \ldots, X_N)$ differs from that of an abstract entity, called the *Teacher*. The Teacher is the ultimate authority that defines the correct classification (T) about $\mathbf{X}$ and might typically be

- a farmer showing the classifier a set of "ripe" and "unripe" items of fruit for harvesting;
- a human inspector watching a production line;
- a committee of people examining a batch of randomly chosen products from a manufacturing process;
- an analyst performing a "post-mortem" on a set of products that have been returned by customers;
- a post-production test machine.

The sole function of the Teacher is to act as the expert authority, helping us design a classifier by adjusting its internal parameters. Eventually, the Teacher must be removed, since it (or he or she) is too expensive, too slow, or does not have the capacity to be used all of the time. For example, an expert human inspector cannot work reliably for more than a few hours of each day. Nearly all work on Neural Networks and other self-adaptive classifiers has been based on this traditional model of Pattern Recognition, in which the Teacher labels samples of at least two pattern classes (Figure 3.5a).

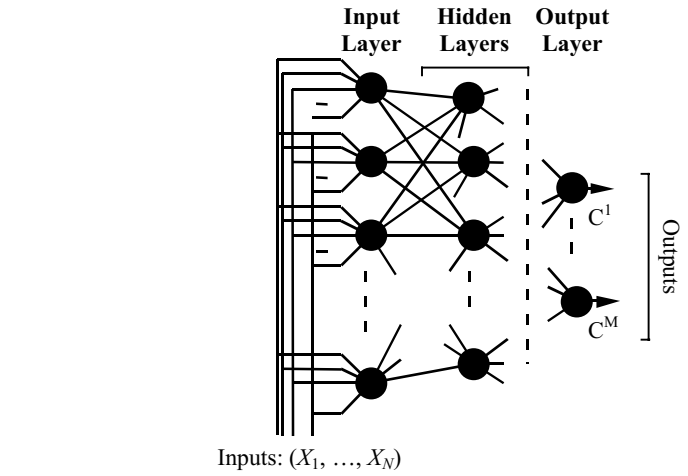


(a)

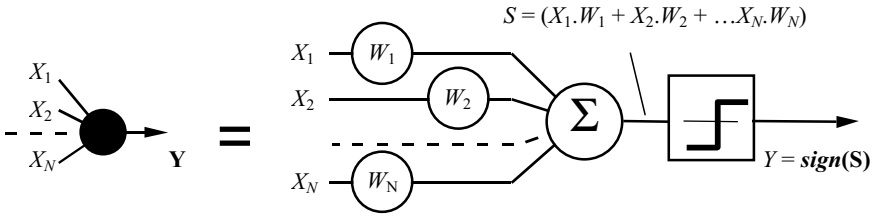


(b)

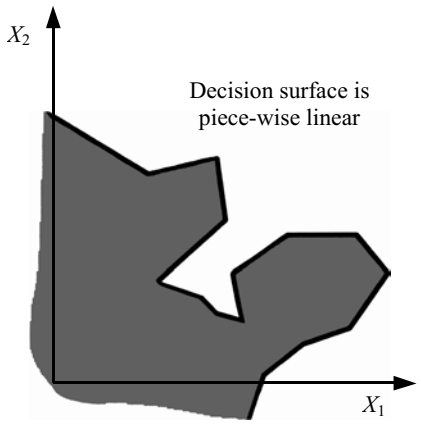
Figure 3.2. Nearest Neighbour Classifier: (a) implementing the algorithm; (b) the pattern represented by the vector $\mathbf{X} = (X_1,X_2)$ is identified with that class associated with the closest stored point: $\mathbf{X}^1$, $\mathbf{X}^2$, $\mathbf{X}^3$, … (i.e., $\mathbf{C}^2$). All points in the shaded area are closer to $\mathbf{X}^2$ than they are to any of the other stored points: $\mathbf{X}^1$, $\mathbf{X}^3$, $\mathbf{X}^4$ or $\mathbf{X}^5$.



(a)

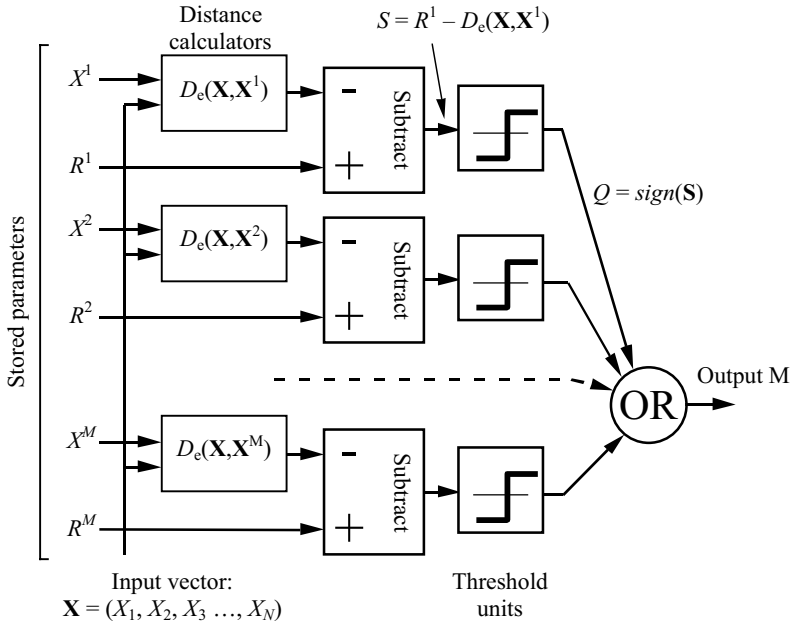


(b)

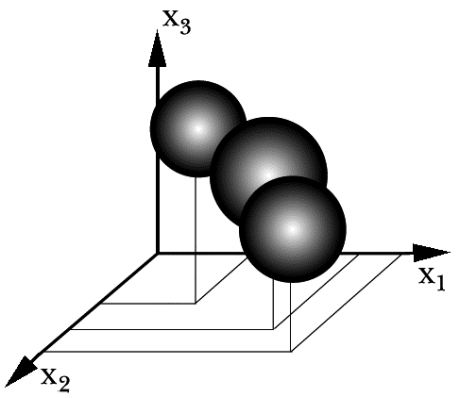


(c)

Figure 3.3. Neural Network: (a) impementation of the algorithm; (b) function of a single isolated "neuron"; (c) decision surface consists of a number of straight lines (2d), planes (3d) or hyperplanes (many dimensions). Notice that both the Nearest Neighbour Classifier and Neural Networks produce piece-wise linear decision surfaces.



(a)



(b)

Figure 3.4. Compound Classifier: (a) internal organisation; (b) decision surface. In 3d, the decision surface consists of a number of spheres, which may or may not overlap. In 2d, they are circles and are hyperspheres in multi-dimensional space.

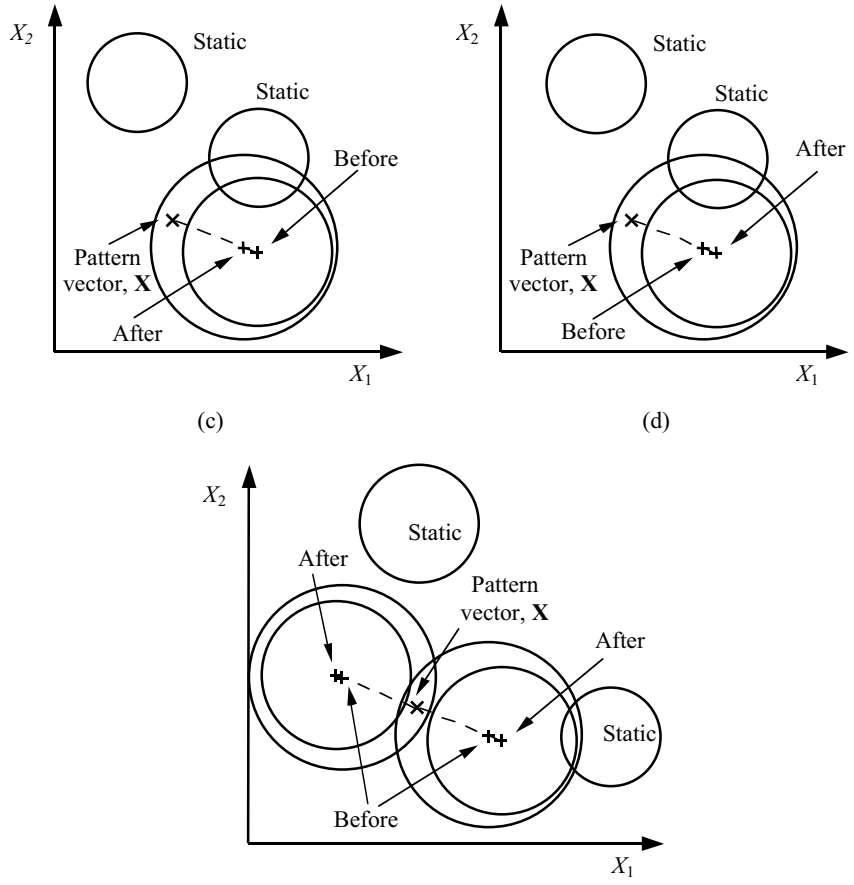


Figure 3.4 (continued) (c) learning rule for the situation in which the pattern vector **X** is outside all circles: the nearest circle is enlarged and is moved towards **X** (all other circles are unchanged). Here, and in the two other situations illustrated, the changes made to the circles are very small indeed; they have been greatly enlarged here for clarity; (d) learning rule for the situation in which the pattern vector **X** is inside just one circle: that circle is reduced in size and is moved away from **X** (all other circles are unchanged); (e) learning rule for the situation in which the pattern vector **X** is inside more than one circle: all circles enclosing **X** are reduced in size and are moved away from **X** (all other circles are unchanged).

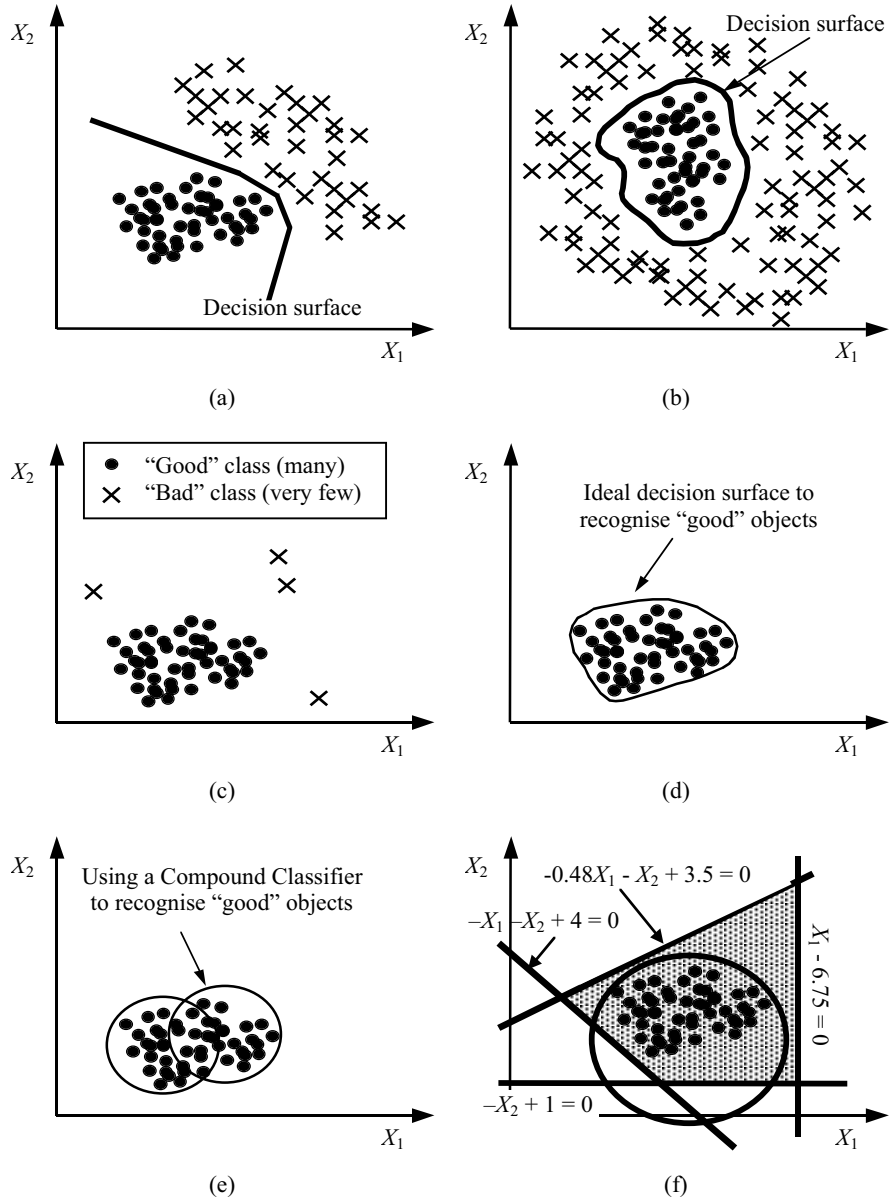


Figure 3.5. Pattern recognition models: (a) traditional model; (b) one pattern class ("good") lies in a compact region, while the "faulty" class surrounds it. (Typically occurs when homeostatic control breaks down. The Compound Classifier is better than Neural Networks in this situation.); (c) there are many samples of the "good", class and very few "faulty" ones; (d) a closed boundary is used to detect anomalies; (e) in practice, a Compound Classifier might be used, although the learning rules differ from those explained in Figure 3.4; (f) combining Pattern Recognition, Compound Classifier, in this case, and rule-based decision methods.

Learning on Single Class

To illustrate the shortcomings of the traditional model, we shall describe two situations in which we should like to design a classifier but cannot obtain a representative set of samples from more than one class.

First, consider an automated bakery. The manufacturing system *never* produces large quantities of "bad" loaves. (We shall ignore the problem of nomenclature in which bakers and production engineers never admit to making "bad" objects but will confess that they occasionally make "unsaleable" items. This and other euphemisms for "bad", "defective", or "faulty" are common-place and should not distract us.) A baking line is an expensive facility and any significant downtime leads to a large financial loss. As a result, the baker will immediately adjust the system, so that it quickly returns to making "good" loaves again; very few "bad" loaves are ever made. The reader might well question whether a sufficient number of "bad" products could be made as part of a research project. For reasons of cost this is most unlikely and is probably impossible in practice anyway. To understand why, let us suppose that there are N individual control parameters on a baking line. Then, a minimum of 2^N settings must be made to the manufacturing system, which may take several hours to settle after some controls are adjusted. (We need to set each control high/low individually.) Some parameters cannot even be altered at all. For example, the amount of moisture in a 10-tonne batch of flour cannot be altered at the whim of an engineer. The effect is that we can obtain very large quantities of the "good" class of products but almost none of the "bad" class, certainly not enough to match the huge appetite of learning systems for training data (Figure 3.5c).

Second, consider the task of inspecting apples. We cannot order a farmer to grow either "good" or "bad" fruit. Even if they could do so, farmers would not be willing to grow diseased or malformed fruit deliberately. Most of the harvest produce will be "good". Once again, we cannot obtain a truly representative set of "bad" apples. The situation is akin to that in medicine; there is a very large number of ways that we can be ill and we are unwilling/unable to be ill to order!

In both of these cases, we can reasonably expect to receive a large set of "good" produce to train a classifier but very few samples of "bad" ones. Pattern Recognition techniques have been developed for this type of situation. They fall into several categories: Unsupervised Learning, Probability Density Function Estimation and Learning on a Single Class (Figure 3.5e) [2].

Hybrid Model

Human beings have the ability to combine self-adaptive learning with rule-based reasoning. To demonstrate this, consider the problem of meeting a stranger, at a pre-arranged point. Prior to the first meeting, the person we are about to meet is described by a mutual acquaintance; recognition then relies on decision rules. At the first meeting, we learn a lot about our new friend's appearance and mannerisms, so that on subsequent occasions, we recognise him/her, by recalling this learned information. By then we may even have forgotten the initial recognition rules, given by our mutual friend. In any case, those rules have been augmented by learning.

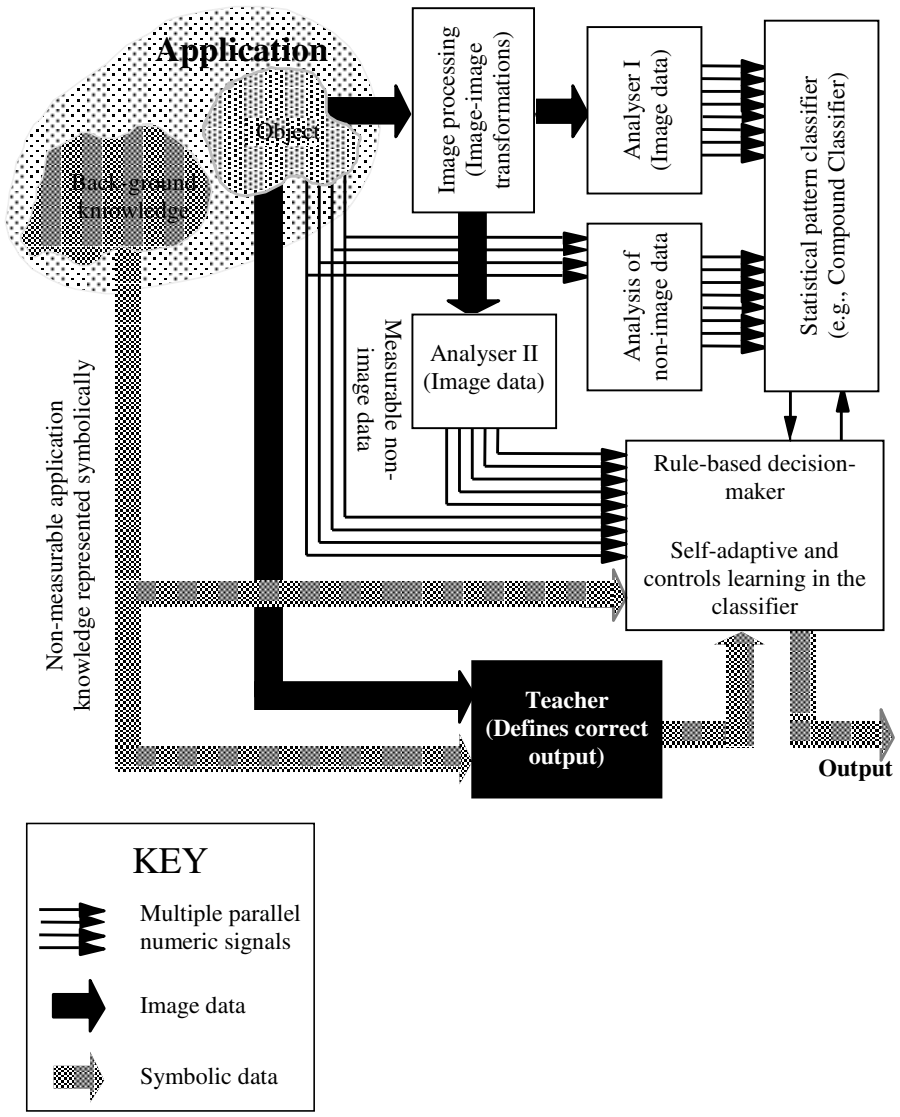


Figure 3.6. Combining Pattern Recognition and rule-based decision making. Notice the 2-way data link between the decision-maker and the classifier. The classifier can provide inputs to the rule-based system and the latter can control the training of the pattern classifier.

To understand how this can help us, let us consider the bakery and fruit-inspection applications again. We might reasonably expect to receive a large set of "good" loaves or apples *and* a set of rules that human inspectors *think* that they employ to grade them. Introspection is often far from accurate as a means of deriving decision rules but it may well provide the best criteria we can obtain in practice. The existence of a large training set (i.e., a collection of physical objects, such as loaves or apples) suggests the use of a self-adaptive learning system, while

the rules just mentioned prompt us to contemplate using an Expert System (Figure 3.5f). With this in mind and for other reasons, a hybrid system is proposed for tasks that present both types of data. Figure 3.6 shows how multi-variate classification and rule-based reasoning might be combined. This has several interesting features: it allows a rule-based system to control a self-adaptive learning process. The former must possess meta-rules specifically for this purpose it allows an Expert System to base its decisions upon sub-decisions derived from the classifier. The Expert System can incorporate non-measurable application knowledge, specified in symbolic terms, by a human being, with both image and non-image data, derived mechanically from product samples. The hybrid system outlined in Figure 3.6 offers a possible solution to some of the problems caused by the special nature of the data that we have available. For far too long, researchers have relied on the traditional Pattern Recognition model (Figures 3.1 and 3.5a and b) without seriously questioning whether it is appropriate. In conceptual terms, the hybrid system has much to commend it. However, this is a speculative suggestion for future work and programming such a system may prove to be problematical.

3.3 Rule-based Systems

The hybrid system just described involving both Pattern Recognition and Rule-based decision-making sub-systems has an obvious appeal but it is simply an idea, put forward to help us analyse images from complex highly variable scenes. It remains untested. However, rule-based systems are already being used, for example, in examining solder joints [3]. In this section, we shall examine their use in a little more detail.

3.3.1 How Rules are Used

Many inspection applications involving natural products can be specified in terms of simple rules. Grading and inspecting vegetables and fruit are prime examples of this kind. Similar comments apply to many areas of food processing. For example, loaves may be examined by applying rules that define what is an acceptable shape. These may be expressed in terms of either two-dimensional slices, or three-dimensional height maps. Ready-made deserts, such as trifles, are specified by a series of rules, defining such parameters as the minimum/maximum amount of fruit. Similarly, limits are placed on the amount of custard and number of bubbles visible on the top surface. Pizzas are specified in a similar way, with limits being expressed for the amount and the distribution of each type of topping. Food products with an ill-defined form, such as croissants, cakes, meringues, and biscuits (cookies). Dollops of cream, icing (frosting) and even adhesive (applied to engineering components) can all be specified using a set of rules. In many applications, it is not possible to use self-adaptive learning, so a set of simple inspection rules specified by an experienced production engineer provides the best approach.

Many existing Machine Vision systems in engineering manufacturing effectively employ naïve rule-based decision criteria. For example, a porcelain plate may be judged to be "acceptable" if

Rule 1 *It contains a single crack-like feature not more than 1 mm long.*
Rule 2 *The total length of all crack/spot- like features is less than 1.5 mm.*
Rule 3 *There are no more than 20 spot-like defects (i.e., length of each is less than 0.1 mm).*
Rule 4 *There no more than 5 crack-like defects (i.e., length of each is more than 0.1 mm).*

As is evident here, individual rules are often quite trivial and may, for example, simply define the maximum length, or width of an object or feature. Sometimes, rules are contrived simply to avoid silly or dangerous situations that might arise as a result of an accident, or human negligence. For example, we might use a simple set of rules to detect a foreign object, such as a screwdriver, or a pair of spectacles, that has inadvertently been left on a food processing line. Simple rules, based only on length, colour and shape factor (ratio of the area to the square of perimeter), can readily distinguish objects that bear some superficial resemblance to one another and can therefore perform a useful "sanity check". For example, the following rule for recognising bananas, rejects cucumbers, courgettes, gherkins, carrots, lemons, oranges, tomatoes, parsnips, etc.

An object is a banana if
i. *it is yellow* and
ii. *it is curved* and
iii. *it is between 75 and 400mm long* and
iv. *it is between 15 and 60 mm wide.*

A computer program, which implements this rule, is listed in the next section. As evidence of the power of simple rule-based decision criteria, look around the room where you are sitting and try to find something, other than a banana, that conforms to this description. Rule-based criteria are used to define classes of fruit for the purposes of the General Agreement on Tariffs and Trade[2] (GATT) and to grade both fruit and vegetables. At the moment, human inspectors are more widely used for these inspection functions than vision systems. Rule-base criteria, based on size and shape, can also distinguish the seeds of certain varieties of cereals. This function is important, as there is a need for a machine that is able to check the honesty of suppliers of rice, wheat, other types of grain and even bananas!

3.3.2 Combining Rules and Image Processing

The idea of integrating image processing operators with the Artificial Intelligence language Prolog [4] was conceived in the mid-1980s and has since been

[2] A few years ago a popular UK national newspaper staged a campaign ridiculing the notion of classifying bananas, in an attempt to discredit certain political views. The relevance to the GATT was not mentioned!

implemented in a number of ways (Figure 3.7). Early systems used external hardware to implement the image processing functions, with Prolog running on a standard desktop computer [5]. The most powerful system of this type developed to date is called *PIP (Prolog Image Processing)* and combines image processing operators written in C with LPA MacProlog [6]. (Both Prolog and the image processing functions run on the same computer.) The latest system, called CIP (Cyber Image Processing) [7], is still under development and incorporates CKI Prolog, which was written by S. van Otterloo of the University of Utrecht, Netherlands [8]. This is a "freeware" implementation of the language and is written in Java. The image processing operators were written separately, also in Java, by students at Cardiff University, Wales, UK [9]. Hence, CIP employs only one computer language, whereas PIP uses two and is therefore more difficult to maintain. PIP currently has a command repertoire consisting of about 275 primitive image processing commands and CIP about 150.

Readers, who are familiar with Prolog, merely need to note that it is possible to implement a system like PIP, or CIP, by building on a single predicat*e (ip/1* in what follows), within the standard language. Readers who have not encountered Prolog before are referred elsewhere for a more detailed description of this fascinating language [4,10].

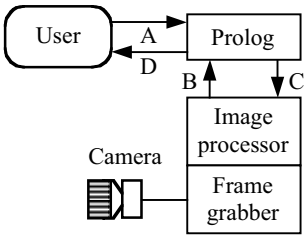
Sample programs are given below, in order to demonstrate how Prolog can incorporate image processing operations.

Program 1: Move the Centroid to the Middle of the Picture.
In this example, we use the predicate *ip/1* to send individual commands to the image processor, from Prolog.

```
shift_to_image_centre :-
ip(enc),                    % Enhance contrast.
ip(thr),                    % Threshold at mid-grey
ip(cgr(X,Y)),               % Locate the centroid, [X,Y]
ip(imm(X0,Y0)),             % Locate the centre of the image, [X0,Y0]
X1 is X0 – X,               % Prolog-style arithmetic
Y1 is Y0 – Y,               % More Prolog-style arithmetic
ip(psh(X1,Y1)).             % Shift so that centroid is at middle of
                            image
```

Program 2: Revised Version, Using the # Operator
This time, we use the operator #, instead of *ip/1*. This is really just a cosmetic change but it can make both programming and subsequent reading of the program easier.

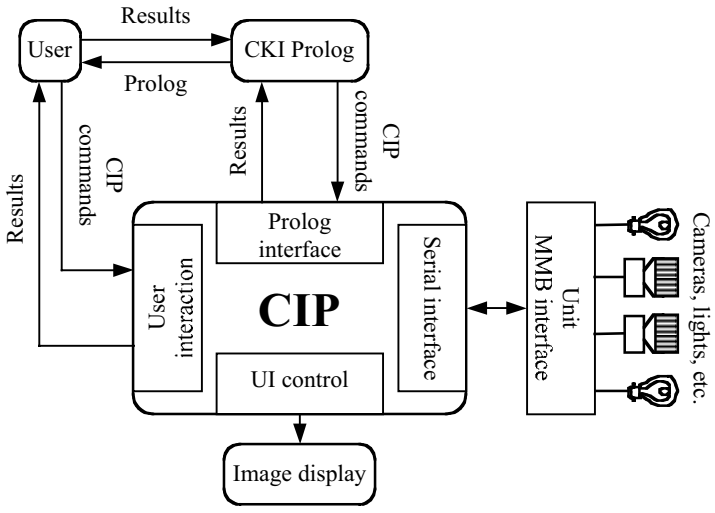
```
:- op(900,#,fx).            % # is prefix operator with precedence 900
# A :- ip(A).               % Defines what # does.
% Revised version of shift_to_image_centre/0
shift_to_image_centre :-
# enc,                      % Enhance contrast
# thr,                      % Threshold at mid grey
```



(a)

A	B	C	D
neg	neg	[0].	YES
thr (115, 134)	thr (115, 134)	[0].	YES
avg(A)	avg	[0,123].	A = 123 YES
cgr(X,Y)	cgr	[0,78,156].	X = 78 Y = 156 YES

(b)



(c)

Figure 3.7. Using Prolog to control an image processor; (a) system block diagram; (b) signals transmitted through the system when various commands are issued by the user (these are PIP commands mnemonics); (c) CIP system architecture. MMB is a multi-function, low-speed interface unit. CKI Prolog was written by S. van Otterloo, University of Utrecht.

```
# cgr(X,Y),                    % Locate the centroid, [X,Y]
# imm(X0,Y0),                  % Locate the centre of the image, [X0,Y0]
X1 is X0 – X,                  % Prolog-style arithmetic
Y1 is Y0 – Y,                  % More Prolog-style arithmetic
# psh(X1,Y1).                  % Shift so centroid is at middle of image
```

Program3: Crack Detector

In this example, commonly-used functions are defined in a library, to avoid using either *ip/1* or the # operator.

```
% Standard library definitions – available to all programs

% The library contains clauses like these, for each image procesing  operator

lnb :- # lnb.
snb :- # snb.
wri(a) :- # wri(A).
rea(A) :- # rea(A).

% Operator: N•G satisfies a goal (G) N times - used in lieu of FOR loops:
:- op(1000, xfx, •).           % • is  infix operator with precedence 1000

0•G :- !.                      % Terminates recursion. Stops when N = 0.

N•G :-
call(G),                       % Satisfy goal G (once)
M is N-1                       % Arithmetic
M•G, !.                        % Now, satisfy the goal (N-1) more times
% End of the library
% Crack detector (Morphological Closing operator)
crk(N) :-
wri(a) and                     % Save image in archive area #1
N•lnb and                      % Brightest  neighbour; grey-scale dilation
N•snb and                      % Darkest  neighbour; grey-scale erosion
rea(a) and                     % Recover  image from archive area #1
sub.                           % Subtract images
```

This and the two preceding programs are simple linear sequences of image processing operations; there is no back-tracking or recursion

Program 4:Finding a Banana

The following program uses back-tracking to search for an object called “banana” that satisfies the rules given earlier. It fails, if no such object is found.

```
object(banana) :-
object(X),                     % Find object in given image. Call it X.
colour(X,yellow),              % Colour of  X is yellow.
curved(X),                     % Object X is curved.
```

```
length(X,L),                    % Measure the length of X.
L ≥ 75, L ≤ 400,                % Check the length limits
width(X,W),                     % Measure the width of X
W ≥ 15, W ≤ 60.                 % Check the width limits
```

Colour recognition, implicit in the sub-goal *colour (X,yellow)* will be discussed in Section 3.4.3. For completeness, we list the following program, which determines whether or not an object in a binary image is curved.

```
curved(X) :-
isolate(X),                     % Isolate the object called X
blb,                            % Fill in the lakes, if there are any
((cvd,                          % Convex deficiency. Call this Image Z
big(2),                         % Find the second largest bay (B2)
cwp(A2));(A2 is 0)),            % Measure its area (A2)
swi,                            % Switch images - revert to Image Z
big(1),                         % Find the largest bay (B1)
cwp(A1),                        % Measure its area (A1)
((cvd,                          % Find the convex deficiency of  B1
big(1),                         % Isolate its largest meta-bay (B3)
cwp(A3));(A3 is 0)),            % Measure its area (A3)
A1 > 5*A2,                      % Area of B1 ≥ 5 times area of B2
A1 ≥ 10*A3.                     % Area of B1 ≥ 10 time area of B3.
```

This is a heuristic procedure and is explained in Figure 3.8. Notice that, although this definition of "curved" may seem a little sloppy, it works most of the time – that is what good heuristics do! It will, for example, succeed when it is applied to the silhouette of a banana.

Program 5: Checking a Table Place Setting

This is a more sophisticated program. It examines a binary image, in order to verify that it conforms to an English-like description of a formal table place setting (Figure 3.9). It uses the relationships *left/2*, *right/2*, *above/2* and *below/2* to compare the positions of objects in two-dimensional space.

```
describe_table_scene :-
loa(original_image),                    % Load the original image
retractall(cutlery_item_found(_,_,_,_)), % Forget all stored items
eab(identify_cutlery_item(_)),          % Identify each item in turn
loa(original_image),                    % Reload the original image
acceptable_place_setting.               % Check layout is satisfactory

% Find an object of type Q, positioned at [X,Y] with orientation Z.
identify_cutlery_item([Q, [X,Y,Z]]) :-

wri,                            % Save image to temporary file on disc
cvd,                            % Convex deficiency
kgr(100),                       % Discard blobs with areas < 100 pixels
```

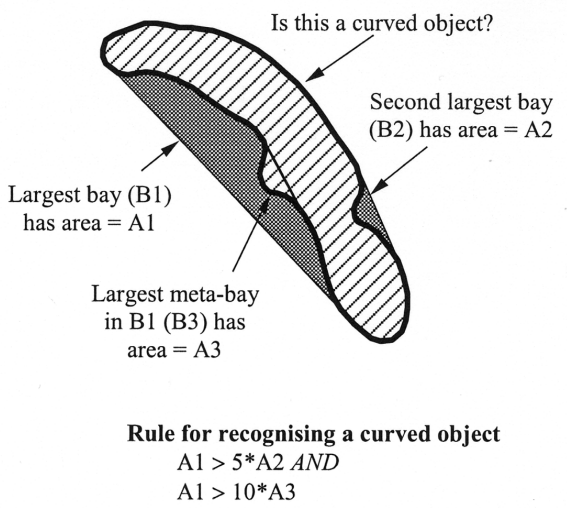


Figure 3.8. Heuristic procedure implemented by the predicate *curved*/1listed in the text.

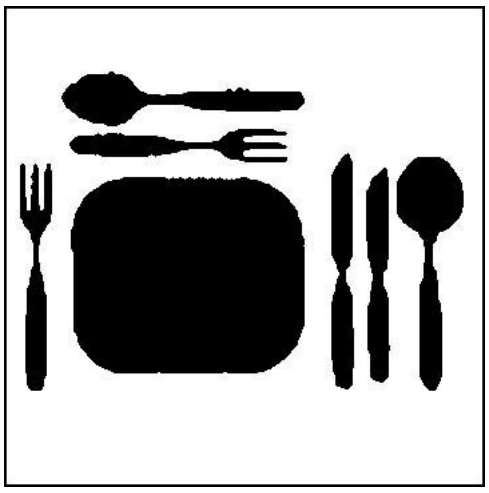


Figure 3.9. Ideal table place setting. Such a scene can be described, in English, using a set of statements of the following form: *"There is a knife to the right of the mat."*; *"A dinner fork can be found on the left of the mat."*, etc. These can be used to program a vision system, which then checks that a real place setting is well laid.

```
ske,                                % Skeletonise the blob
cnw, min, thr(1,1),                 % Identify the skeleton limb ends
cbl(B),                             % Count them
rea,                                % Recover image from temporary file
cwp(C),                             % Count white points (i.e., find area)
lmi(X,Y,Z),                         % Centroid, and orientation of principal axis
cutlery_item(Q,[C,A,B]),            % Identify the type of object found (Q)
tell_user(Q,A,B,C),                 % Tell the user what we have found
assert(cutlery_item_found(Q,X,Y,Z)).
                                    % Remember what we have found
/*_____________________________________________________________________
How to identify objects. The following arguments are used:
cutlery_item(Object_type, [Area, No_of_limb_ends, No_of_bays])
_____________________________________________________________________*/
cutlery_item(mat,[A,_,_]) :-  A ≥ 5000, !.      % Object is a mat
cutlery_item(knife,[_,0,_])  :- !.              % Object is a knife.
cutlery_item(fork,[_,_,4])  :- !.               % Object is a fork.

cutlery_item(spoon,[_,2,_])  :- !.              % Object is a spoon.
cutlery_item('not known',_).                    % Not of known type

% Defining spatial relationships between two objects, A and B
left(A,B) :-
location(A,Xa,Ya),                  % Centroid of object A is at [Xa,Ya]
location(B,Xb,Yb),                  % Centroid of object B is at [Xb,Yb]
Xa < Xb,                            % Check that Xa < Xb
about_same(Ya, Yb)                  % Are Ya and Yb about the same?

right(A,B) :- left(B,A).            % right/2 is the inverse of left/2

above(A,B) :-
location(A,Xa,Ya),                  % Centroid of object A is at [Xa,Ya]
location(B,Xb,Yb),                  % Centroid of object B is at [Xb,Yb]
Ya < Yb,                            % Check that Ya < Yb
about_same(Xa, Xb),
message(['I have found a ',A, ' above a ', B]).

below(A,B) :- above(B,A).

% Finding the position (X,Y) of an item of type A.
location(A,X,Y) :- cutlery_item_found(A,X,Y,_).

% An arbitrary, very naïve, way of checking that A and B are similar
about_same(A,B) :- abs(A-B) < 25.
```

```
/*_____________________________________________________
The following rule specifies what constitutes an acceptable table place
setting. It can easily be reformulated to accept English-language
descriptions, using Definite Clause Grammars (DCGs).  These form an
integral part of Prolog [10].
_____________________________________________________*/
acceptable_place_setting :-
left(fork,mat),                    % There is a fork to the left of a mat
right(knife, mat),                 % There is a knife to the right of a
                                   mat.
left(knife,knife),
left(knife, spoon),
above(fork,mat),
below(fork,spoon),                 % There is a fork to the left of a
                                   spoon
message('The table place setting is acceptable.').

% What to do when we cannot prove that table place setting is acceptable

acceptable_place_setting :-
message('This is not an acceptable table place setting').
```

Why Prolog?

The reader may well wonder why Prolog, rather than a conventional language, such as C or Java, is used to control the image processing engine. In the short space available here, it is impossible to do full justice to this question. The reader is therefore referred elsewhere for a more complete discussion [5,11].

A hint of the expressional power of Prolog is seen in the last two examples. In Program 4, we employed heuristic definitions for *curved/2* and *colour/2* (yet to be described in detail) while in Program 5, we defined the "sloppy" spatial relationships: *left/2*, *right/2*, *above/2* and *below/2*. It is difficult to envisage how this could be done in a conventional computer language. Prolog's expressional power arises because it is a *Declarative Language* and is therefore able to manipulate abstract symbols (words) without explaining what they mean. For example, the relationship *left(A,B)* may refer to objects A and B in physical space, or in "political space".[3] Since we do not need to specify the nature of A and B, the definition given above for *left/2* will suffice for both. When programming in Prolog, we simply describe (a little bit of) the world. We then ask questions about it and rely on Prolog's built-in search engine to hunt for a solution. Conventional languages are said to be *imperative*, since programs consist of a series of commands. Implementing a search algorithm in a language such as C is tedious;

[3] Consider the following: Hitler ? Stalin If we are considering physical objects (words), *left(Hitler,Stalin)* is true, while it is clearly false, if political objects are compared. Our Prolog program can accommodate either.

especially when we do not know in advance precisely what kind of object we shall be looking for.

We conclude this section by considering a task, in which declarative programming excels. Suppose that a certain man (A) wishes to find a wife (B) for himself. The Prolog programming paradigm requires that A first specifies his "requirements" as illustrated in Table 3.1.

Writing a Prolog program to find a wife is very straightforward, as is evident from the program segment below. (To conserve space, only the top level is specified. The lower levels are also programmed easily.)

```
marriage_partner(A,B) :-
male(A),                          % This rule applies to men. A similar
                                  % rule should be added for women.
person(B),                        % Find a person called B
female(B),                        % Check that B is a female
age(B,C), C ≥ 38, C ≤ 42          ,  % Find how old B is and check limits
height(B,D), D ≥ 1500, D ≤ 1750, % Find height and check limits
hair_colour(B,blonde),            % Check that B is blonde
religion(B,protestant_christian), % Check B is Protestant Christian
personality(B, [kind, generous, loyal, truthful]), % Check all are true
hobbies(B, [swimming, theatre, antiques]). %  Check one or more is true
```

Table 3.1 Example of prolog programming paradigm requirements.

Feature	**Value**
Sex	Female
Age	38 – 42
Height	1500 – 1750
Race	Caucasian
Hair colour	Blonde
Religion	Protestant Christian
Personality (all must be true)	[Kind, generous, loyal, truthful]
Hobbies/interests (at least one must be true)	Swimming, theatre, antiques]

Note that we are combining data relating to "natural products" (people) that is derivable from images (e.g., height, hair colour), with non-image data (e.g., religion and personality).

It should be clear by now that Prolog with embedded image processing operators is well suited to tasks such as grading and sorting fruit, vegetables and decorative plants. There is another good reason for using Prolog: it is able to manipulate natural language [10]. Combined with a speech recognition system, this provides a powerful user-interface facility, for hands-free supervision of a robot

vision system [5]. Natural language programming, albeit in a simple form, is also feasible. The level of complexity that we can reasonably expect to accommodate is roughly commensurate with that needed to program a visually guided robot to lay the table, or place tools in a tool-box More significant industrial applications are found in packing natural objects, stacking irregular packages, etc.

3.4 Colour Recognition

We turn our attention now to colour recognition, which we have already mentioned and used. Our goal in this section is to explain how colours can be recognised automatically, so that a person can then employ symbolic colour labels to program a vision system. He/she may then refer either to "standard" colours (e.g., yellow, turquoise, blue, etc.), or to application-specific colours ("tuna-can red", "margarine-tub red", "butter", "banana yellow", etc.). As we proceed, we shall apply the concepts and methods of Pattern Recognition that we introduced earlier.

3.4.1 RGB Representation

A colour video camera measures the so-called RGB components of the light falling at each point in its retina. It does this by placing monochrome photosensors behind three differently coloured filters. These selectively attenuate different parts of the spectrum, to give what are called the R, G, B colour channels. These have traditionally been associated with detecting "red", "green" and "blue", although this idea must not be carried too far. A camera that outputs an HSI video signal (measuring Hue Saturation and Intensity) uses an RGB sensor and performs the RGB-HSI transformation within its internal electronics.

Let $[r(i,j),g(i,j),b(i,j)]$ denote the measured RGB values at the point (i,j) in a colour image. The set of colours in the image can be represented by S_3 where

$$S_3 = \{[r(i,j), g(i,j), b(i,j)]/\ 1 \leq i, j \leq \{N\} \tag{3.9}$$

By mapping the image into this particular three-dimensional measurement space (called RGB-space, with axes r, g and b), we forsake all knowledge of the position of each colour vector. However, by extending this, we can preserve this information. The set of points

$$S_5 = \{[r(i,j), g(i,j), b(i,j), i,j]/\ 1 \leq i, j \leq \{N\} \tag{3.10}$$

lies in a 5-dimensional space and preserves both position and colour information. For most purposes, S_5 is too cumbersome to be useful in practice. The representation embodied in S_3 is normally preferred, when we want to recognise colours independently of where they lie within an image.

We can combine the output of a colour recogniser with the position co-ordinates, (i,j), to obtain essentially the same information as is implicit in S_5. The

following is a hybrid representation which combines symbolic colour labelling (not "raw" RGB values) with geometric position (i,j) within an image.

$$S_{\text{hybrid}} = \{[colour\_name, i, j] / \ 1 \leq i, j \leq \{N\} \tag{3.11}$$

This representation is more economical than S_5 yet contains more information than S_3. Moreover, S_{hybrid} can be stored and processed as a monochrome image, in which "intensity" (i.e., an integer) represents symbolic colour names (Figure 3.10).

It is often convenient to display such an image in pseudo-colour, since similar values of the integers defining the values of *colour_name* may represent quite different physical colours.

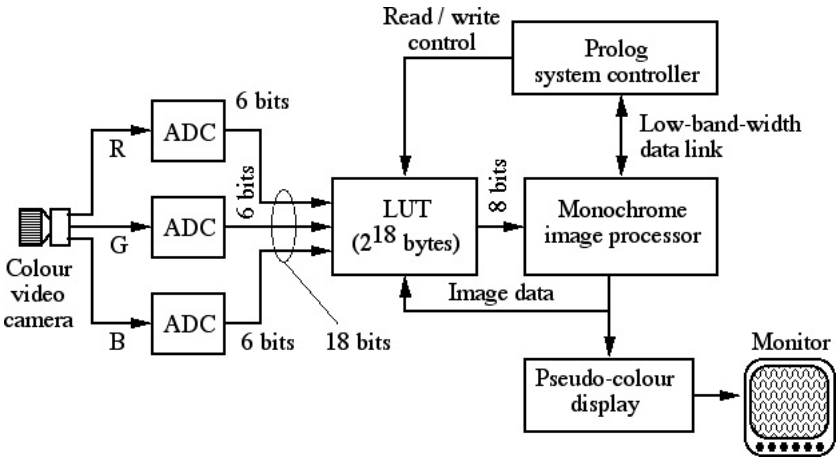


Figure 3.10. System for recognising colours under the control of an intelligent processor

3.4.2 Pattern Recognition

The representation implicit in S_3 (Equation (3.9)) allows us to apply conventional Pattern Recognition analysis to identify colours. We would like to find some convenient way to divide RGB-space into distinct regions, each of which contains all of the points associated with a given colour label, as defined by a person (p). (We might associate person p with the teacher in Figure 3.1.) We could use the Compound Classifier; in which case, the co-ordinate axes in Figure 3.4b are $[r(i,j),g(i,j),b(i,j)]$. The Nearest Neighbour Classifier, or Neural Networks, might be used instead.

3.4.3 Programmable Colour Filter

Another approach to colour recognition is explained in Figure 3.10. This derives the form of representation implicit in S_{hybrid} (Equation (3.4)). The look-up table (LUT) receives the digitised (r,g,b) inputs from the camera and generates a

symbolic colour label (an integer, equivalent to *colour_name*), representing familiar names for colours. The fact that the LUT output is a number does not imply that approximately equal LUT outputs are necessarily mapped to subjectively similar colours. For example, we might arbitrarily associate names and LUT outputs as follows:

"turquoise" with LUT output 136
"sulphur yellow" with LUT output 137
"blood red" with LUT output 138
"sky-blue" with LUT output 154
All other colours are mapped to LUT output 0, signifying *"colour not recognised"*.

Furthermore, notice that only a few of the possible output states from the LUT may actually be used. The diagram shown in Figure 3.10 describes a low-cost, high-speed colour-recognition technique that can be implemented easily in either special-purpose hardware or software. The procedure for calculating the contents of the look-up table is described in [12].

3.4.4 Colour Triangle

A graphical representation for recognising colours will now be described. This allows us to map the variability of natural products into two-dimenional space, in a convenient and meaningful way. It also allows us to classify colours in a straightforward manner. Consider Figure 3.11, which shows the RGB space as containing a cube.

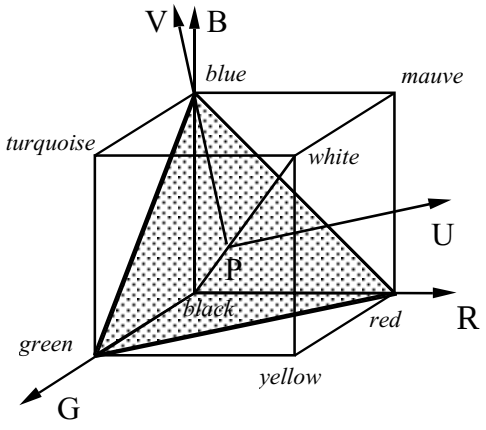


Figure 3.11. RGB space and the Colour Triangle.

The vector (*r*,*g*,*b*) is constrained to lie within this cube, since each channel produces an output, within a finite range: [0,*W*]. It has been found from observation that the subjective impression of colour experienced by a human being

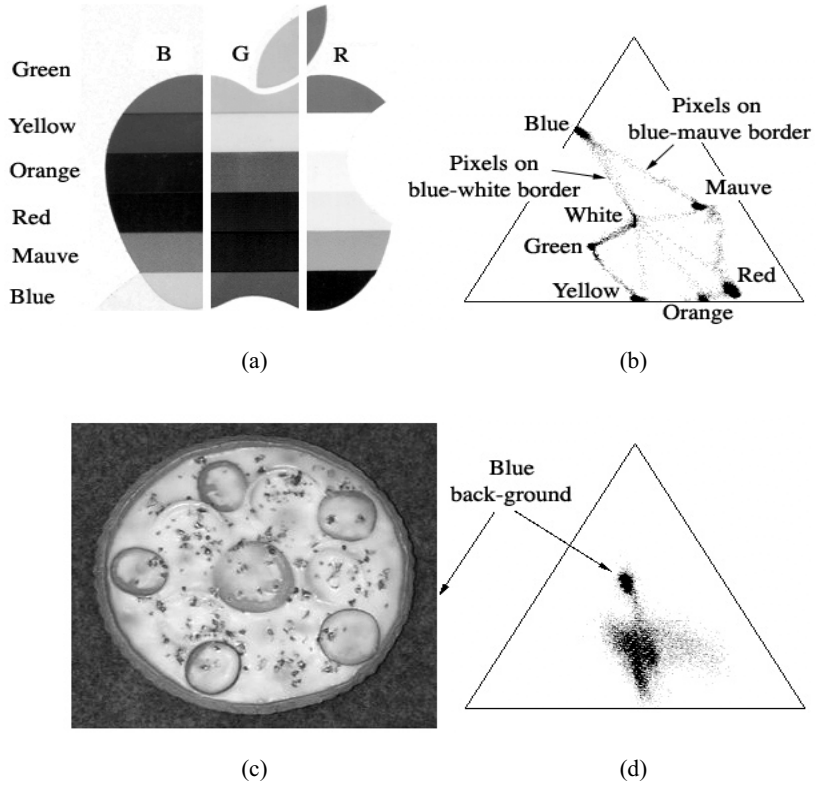
can be predicted with reasonable accuracy by noting the orientation of the vector (r,g,b) within this space. (There are many factors affecting the judgement in practice, but this is a reasonable approximation for many purposes.) Two angles are needed to define the orientation of the (r,g,b) vector. Another possibility is to note where this vector intersects that plane which goes through the points (W,0,0), (0,W,0) and (0,0,W) in RGB space. This plane intersects the cube to form a triangle: the so-called *Colour Triangle* or *Maxwell Triangle*. Now, consider a polychrome image, such as the logo used by Apple Computer, Inc. This is a good example for our purposes, as it is composed of a number of well-separated block colours. When this is mapped into the Colour Triangle, a number of clusters are created (Figure 3.12). There are just a few points lying between these clusters and these are due to pixels that straddle the boundary between neighbouring colour bands. However, most points lie very close to the centre of one of the clusters. (Camera noise causes regions of constant colour to be mapped into clusters of finite size.) On the other hand, when a food product (e.g., a pizza, or quiche) or a natural product (e.g., apples) are mapped into the Colour Triangle, the clusters are much more diffuse and tend to merge into one another with no clear gap existing between them (Figure 3.12c – h). Blending of colours into each other is, of course, characteristic of natural products and this is manifest in the Colour Triangle by indistinct clusters.

The Compound Classifier, this time working in two-dimensional space can be used for colour recognition. This time, the inputs to the classifier (U and V) are those defining points in the Colour Triangle (Figure 3.13). This method of making decisions about the colours present in an image is unable to distinguish between bright and dark regions and, for this reason, its output is noisy in regions of low intensity. Of course, these can be eliminated by simple masking.

The heuristic techniques that have just been described do not provide a perfect solution to the task of recognising colours. If they are used intelligently and without unduly high expectations, they provide a useful analysis tool. To be most effective, they should be combined with other image descriptors, possibly in a rule-based system.

3.5 Methods and Applications

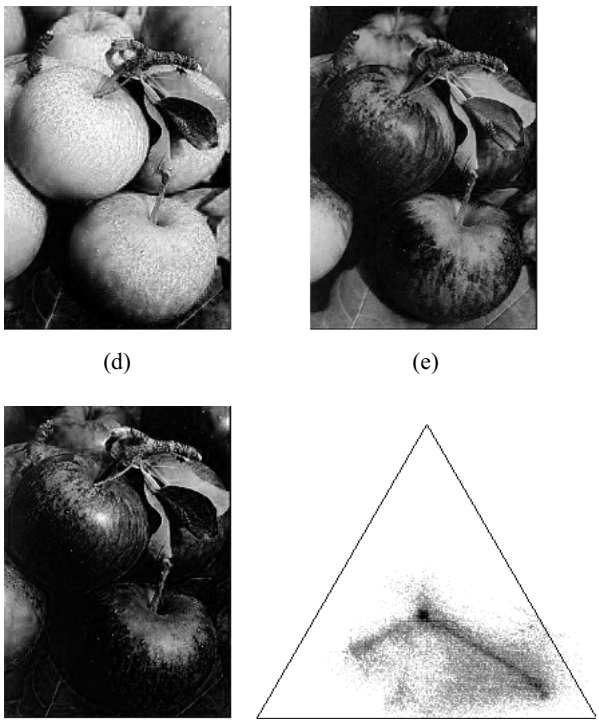
Automated Visual Inspection presents a wide variety of problems, requiring an equally diverse range of algorithmic and heuristic techniques for image processing and analysis. So far in this chapter, we have described just three of them: i. pattern classification by sub-dividing a multi-dimensional measurement space; ii. rule-based decision-making and iii. colour recognition. Of these, i and ii have never found wide application for inspecting close-tolerance engineering artefacts. It is the lack of a precise specification and the inherent variability of natural products that require their use now. Over the last 25 years, Machine Vision systems have been successfully applied in manufacturing industry and we might think that the same ideas can be applied to natural objects as well. In some cases they can, while in



(a) (b)

(c) (d)

Figure 3.12. Mapping artificial and natural images into the Colour Triangle: (a) colour separations for the logo used by Apple Computer Inc.; (b) clearly identifiable clusters are generated from this logo; (c) baked quiche (this is the intensity component of the original, which is an RGB colour image); (d) mapping the colour image of the quiche into the Colour Triangle. Notice that the compact upper-most cluster is produced by the background and that no well-defined clusters are attributable to the quiche itself.



(d) (e)

(g) (h)

Figure 3.12 (continued) Mapping artificial and natural images into the Colour Triangle: (e) apples and leaves (R component); (f) apples and leaves (G component); (g) apples and leaves (B component); (h) Colour Triangle produced by the "Apples and leaves" image. Again, there are no well-defined clusters.

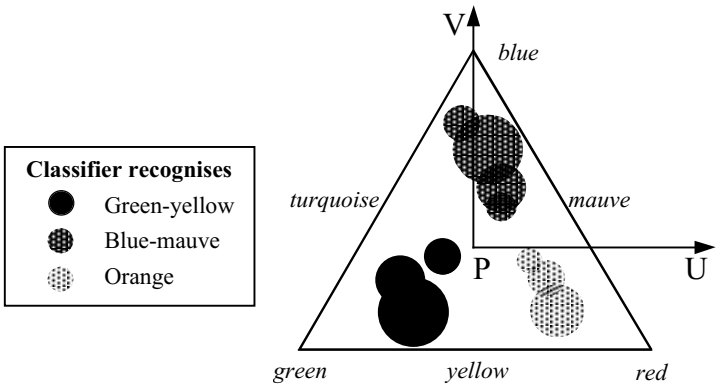


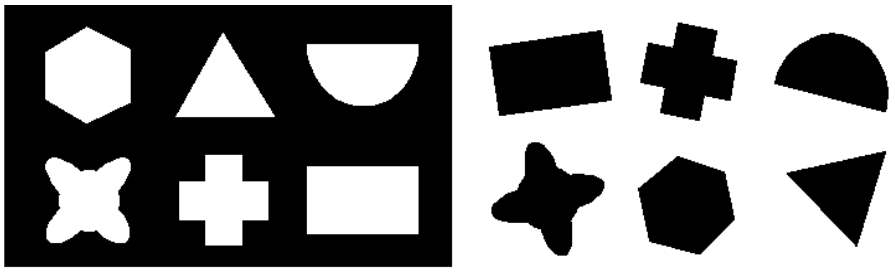
Figure 3.13. Three Compound Classifiers, working in parallel, for colour recognition in 2-dimensional space. The variables *U* and *V* are explained in Figure 3.11.

others they require some adjustment. The purpose of this section is to explain and illustrate this point, by presenting a series of brief case studies. We shall emphasise the differences that exist between inspection methods for engineering artefacts and highly variable objects, rather that the applications *per se*. The lesson is that tried and trusted procedures do not necessarily work well on highly variable objects. Some of our examples are contrived using human artefacts: a child's toy, a pair of scissors, a coil of wire. We shall use the terminology of the PIP image processing system to define image algorithms and heuristics.

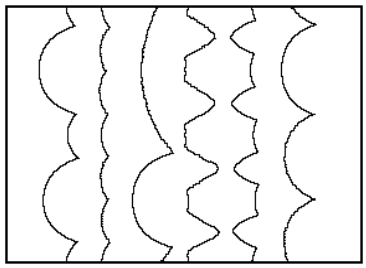
3.5.1 Human Artefacts

Child's Toy

The child's toy illustrated in Figure 3.14 presents unexpected problems for a visually guided robot. Our task is to design an image processing procedure that can guide a robot, so that it can place each shape appropriately in the template. Both position and orientation have to be determined. Finding the position is easy; in each case, the centroid can be used. However, several of the established techniques for finding orientation are unable to accommodate such a wide variety of shapes. Among the popular techniques that fail to calculate a reliable value for the orientation are:



(a) (b)



(c)

Figure 3.14. Child's toy: (a) template; (b) shapes (c) plot of the distance ($r(\theta)$) from the centroid, vesus angle (θ, vertical axis).

- Hough and Radon transforms (fails on the 4-point star).
- Principal axis (axis of minimum second moment) (fails on the cross, star, hexagon and triangle). Since mathematical moments do not measure orientation directly, some combination of measures of this kind has to be used.
- Edge-follower operator for detecting sharp corners, fails on the star.
- Convex deficiency, fails on the rectangle, semi-circle, hexagon, triangle.
- Distance of the furthest edge point from the centroid, fails on the semi-circle. (It also fails to distinguish the star, or cross, from their respective convex hulls.)

The task is made easier if the type of shape is identified first. Then, an appropriate technique for calculating orientation can be applied. Several techniques do work but with varying degrees of success:

- Fourier coefficents of the function $r(\theta)$, where r is the radius (measured from the centroid) at angle θ. (This technique will not work if the shape is more complex (e.g., crab-like) and produces a multi-value function for $r(\theta)$.)
- Fourier coefficents of the function $r(d)$, where d is the distance measured around the edge from some convenient starting point (e.g., right-most pixel). ($r(d)$ is always a single-valued function but distances around the edge cannot be measured very accurately.)
- Correlation of image arrays (three degrees of freedom) always produces a satisfactory result but the calculation is slow. The execution time can be reduced to that of a one-dimensional correlation procedure, if the position is fixed first, by using the centroid.
- Correlation of the function $r(\theta)$ (Figure 3.14c.) Again, this will not work for complex shapes. This procedure does produce satisfactory results for recognising or placing shapes like those produced by ivy leaves (Figure 3.19i)).
- Combinations of mathematical moments. It is possible to define a set of moments that are insensitive to rotation, translation and scale changes [13]. These can be used to recognise the shape first, before selecting an appropriate orientation algorithm.

Articulated Assemblies: Scissors

Of course, we expect variable objects to display significant changes of geometry but they can have a variable topology, as well. For example, a pair of scissors open by even a very small amount may have a different Euler number (a measure that is sensitive to its topology) from that of the scissors in the closed state (Figure 3.15a and b). This means that we have to be very careful when designing image analysis algorithms. For example, we might wish to build a visually guided robot to pick up the scissors, which could be closed, slightly open or wide open. It so happens that the two largest lakes in any of the silhouettes produced by the particular pair of scissors shown in Figure 3.15d correspond to the two finger holes. However, this is not necessarily the case for all types of scissors. If we decide that we can use the

finger holes to pick up the scissors, it is a simple matter to calculate their centroids, as possible places to position the robot's fingers.

Trying to define a general procedure that can pick up any pair of scissors is quite difficult, as not all have closed finger holes, while some have three, or more. Closely related objects, such as shears and pliers, may have no finger holes of any sort. Here, we have two high-tolerance components, with a simple hinge connection between them. The result of this combination is an object that taxes us severely. More complex assemblies of levers are, of course, common-place in a variety of instruments, automobiles, etc. Manipulating these is even more difficult. However, the worst situation of all is found in handling objects with long pendant tubes, ropes, strings wires or cables. Plants fall in the same category. How do we cope with this huge variety? The answer is that there is no single solution that is appropriate for all situations; even our marvellous eye-brain complex finds unravelling a ball of string very difficult. In the simpler cases, a rule-based methodology is likely to suffice and, in most cases, would be our preferred solution. The trick is to sub-divide the overall task, so that each sub-problem can be tackled separately, probably using methods that are already familiar to Machine Vision practitioners. A system like CIP, or PIP, is well suited to a divide-and-conquer approach. (Each Prolog clause deals with a separate case.) However, we have to accept that some problems remain far beyond the ability of systems like this and, at the moment, have no known solution.

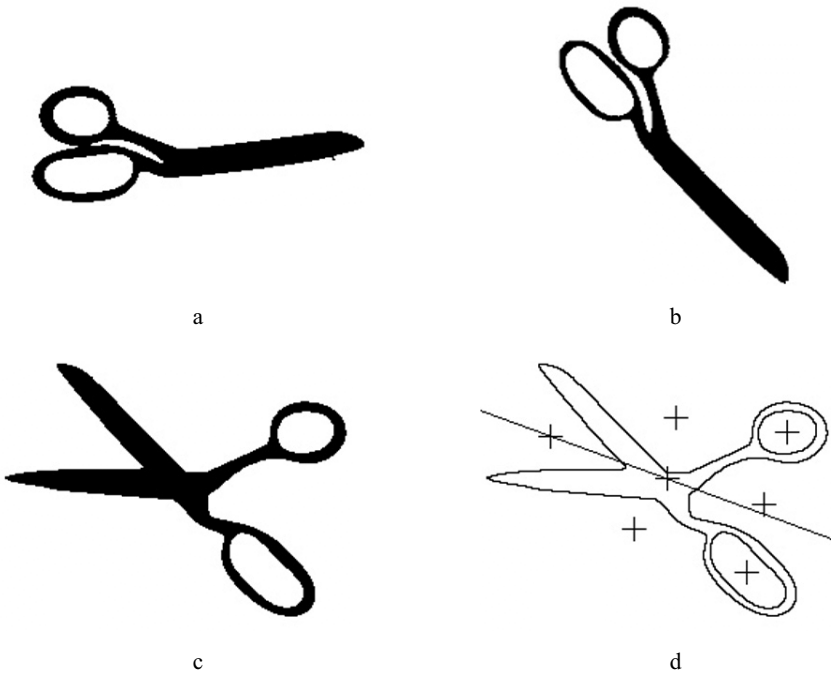
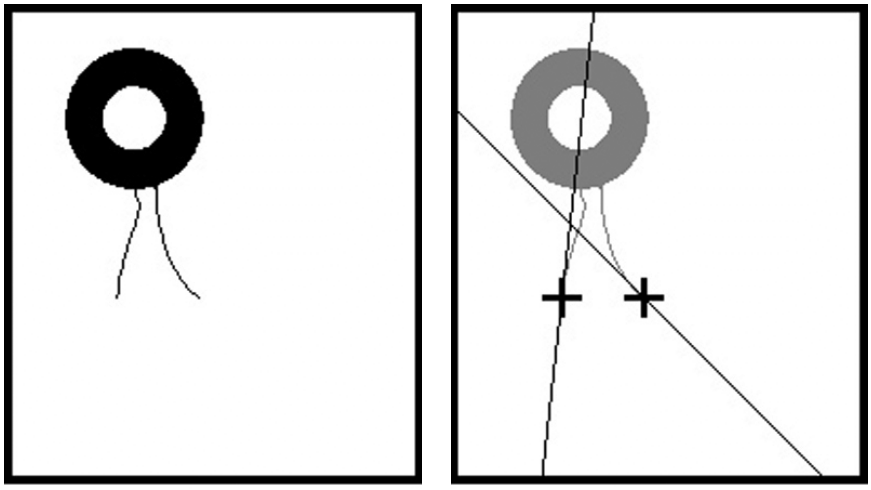


Figure 3.15. Scissors: (a) fully closed (Euler number is –2); (b) nearly closed (Euler number is –1); (c) open wide (Euler number is –2) (d) centroid and the principal axis, centroids of four bays and two lakes.

Flexible Objects

We have already pointed out the difficulties of working with flexible objects. Relatively simple situations like the coil shown in Figure 3.16 can be solved by interpreting images from several cameras. Our task is to design a vision algorithm that can locate and orientate the ends of the wires. Simple image filtering (grey-scale morphology), thesholding and skeletonisation yields a binary image that can be analysed in a straightforward way. The skeleton limb-ends are located first. Then, the ends of the skeleton limbs are eroded back a short distance, and the ends of the shortened limbs are found. This process yields two points: one at the end of each wire and another close to the end. These allow us to estimate the orientation close to the tip of the wire. However, notice that this yields only the orientation in two-dimensional space (e.g., the horizontal plane for Figure 3.16). At least one more view, this time from the side, is needed to locate and orientate the end of each wire in a vertical plane.



(a) (b)

Figure 3.16. Coil with two flying leads: (a) binary image (the original image was processed, by grey-scale morphological filtering and thresholding, to improve visibility of the wires); (b) ends of the wires (crosses) and tangents at the ends of the wires

More than one side-ways view of the coil is needed if there is any likelihood that the body of the coil will obscure the views of the tips of the wires. In this example, a rule-based (Prolog) system is needed to cope with all of the different situations that can occur as a result of the uncertain orientation of the coil (in the horizontal plane) and bending of the wires.

A more sophisticated approach might be needed sometimes, if for example, the wires become tangled. We can probably cope with minor entanglement, by using simple physical adjustment of the wires, prior to the process that we have just described. A robot could be used to make a slight adjustment to the flying wires, by simply inserting a finger and pulling gently. Guiding the robot as it performs

this initial manipulation is another role for a vision system. An important part of this process is knowing when to give up. By now, the reader will be aware of the high level of machine "intelligence" needed to perform even simple manipulation on objects, like this coil, that have floating tubes, strings or wires.

3.5.2 Plants

There are many applications involving either whole plants, or parts of plants, that we might discuss here, for example: grading decorative house plants, harvesting fruit, trimming rhubarb, inspecting rice plants for infestation by parasitic insects, replanting seedlings, selective application of weed-killer, separating seeds from husks, "assaying" a ship-load of wheat (to ensure that it is of the correct variety), etc. We have selected just three tasks that illustrate the fact that a high level of intelligence is needed. We must not always expect established techniques, which have long been standard in industrial vision systems, to work reliably.

Leaves

To the human eye, the leaves of some plants, such as certain species of ivy, oak, and maple, appear to have very characteristic shapes. On casual observation, mature leaves of a given species often look so much alike that we not aware of the high level of variation that actually exists. This says more about our ability to perform Pattern Recognition than it does about the constancy of leaf shape. As we shall see, when we analyse leaf shape objectively, we become much more aware of the high level of variability.

Figure 3.18 shows the silhouettes of five ivy leaves, taken from the same plant. (The stems were removed manually, to make the image analysis a little easier. However, similar results could have been achieved using morphology.) In the first instance, we chose to analyse their shapes using so-called *Concavity Trees (CTs)* [14]. This method of representing shape has some attractive theoretical properties and CTs have been studied for applications such as identifying leather shoe components [5]. We might imagine that a technique that is suitable for matching shapes such as that shown in Figure 3.17 might also be appropriate for doing the same for leaves.

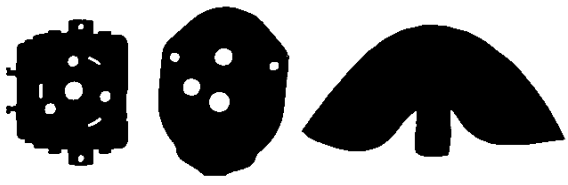
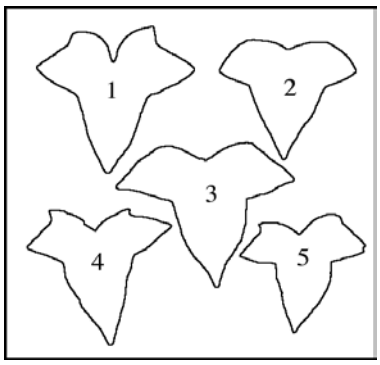
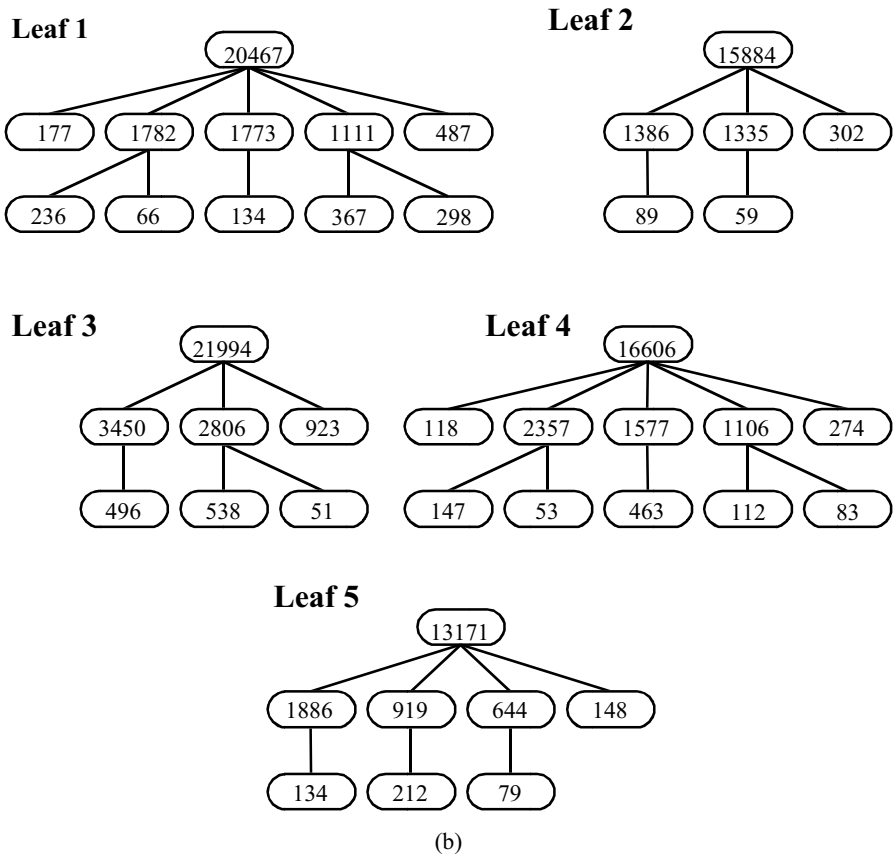


Figure 3.17. Metal castings and a shoe component are suitable for concavity trees analysis.



(a)



(b)

Figure 3.18. Ivy leaves analysed using concavity trees: (a) five leaf silhouettes; (b) concanvity trees. Numbers indicate the areas of the convex hulls for each shape. Minor branches (e.g., corresponding to small shapes) have been removed, for clarity.

At first sight, CTs seem to provide a very natural way to represent shapes of leaves such as ivy, since they allow us to combine global and local information in a systematic way. However, the CTs generated from ivy leaves are highly variable, reflecting differences of leaf shape that would probably elude a casual glance. In view of this, it is not possible to use CTs in a simple way, to compare the shapes of leaves. We could not, for example, compare two shapes by simply overlaying their CTs. A much more sophisticated tree-matching process is needed. It would be too much of a distraction to describe such a procedure here.

Other methods of analysis are needed, if we are to perform shape matching on highly variable objects like ivy leaves. Heuristic, rather than algorithmic, techniques for shape analysis are probably more appropriate for situations like this. Consider Figure 3.19, which shows various methods that might be used for determining the orientation and position of a leaf. Both are needed as a prelude for certain methods for shape matching. Figure 3.19(a) shows that the principal axis cannot be used to normalise the orientation reliably, while Figure 3.19(b) suggests that the centroid of the leaf silhouette and the centroid of the small bay at the top provide a better reference axis for this purpose. However, in some leaves, this bay is shallow and ill defined (Leaf 2 in Figure 3.18(a)). Another possibility is provided by the line joining the centroid to the edge point that is furthest from the centroid (Figure 3.19(c)). Which of these is more reliable needs to be evaluated by detailed study of a carefully selected sample of leaves.

Figures 3.19(d) – (f) illustrate three methods for finding reference points that might be used to control the warping of a "rubber template", as part of a shape-matching routine. It is common practice to use the centroid as a means of determining position but other methods might be more appropriate. For example, the centre of the circumcircle (Figure 3.19(g)), or the centroid of the convex hull (Figure 3.19(h)) provide possible alternatives.

Figure 3.19(i) shows a plot of radius ($r(\theta)$), measured from the centroid, against angle (θ) for three ivy leaves.[4] Simple (1-dimensional) correlation of these graphs could be used for shape matching. Since $r(\theta)$ is a periodic function of θ, it is also possible to apply Fourier analysis techniques. The Fourier coefficients could be used to provide a set of inputs to a pattern classifier of the type discussed earlier in this chapter. A set of low-order Moments could be used for this purpose instead.

The conclusion we are forced to make is that shape matching of natural objects is far from straightforward. Even though similar functions have long been standard in manufacturing of engineering products, the high level of variability makes the established methods unreliable. Moreover, greater intelligence in shape-matching procedures is needed. It is important to appreciate that the greater uncertainty about which technique is best suited to a given application makes testing and evaluating the various options more difficult and therefore more time-consuming. It is imperative, therefore, that we have a properly organised set of image analysis tools for this type of task.

---

[4] For ivy leaves, $r(\theta)$ happens to be a single-valued function of θ. If this is not the case, it may be necessary to use the function $r(d)$ instead, where d is the distance measured around the edge. The latter can be estimated from the chain code.

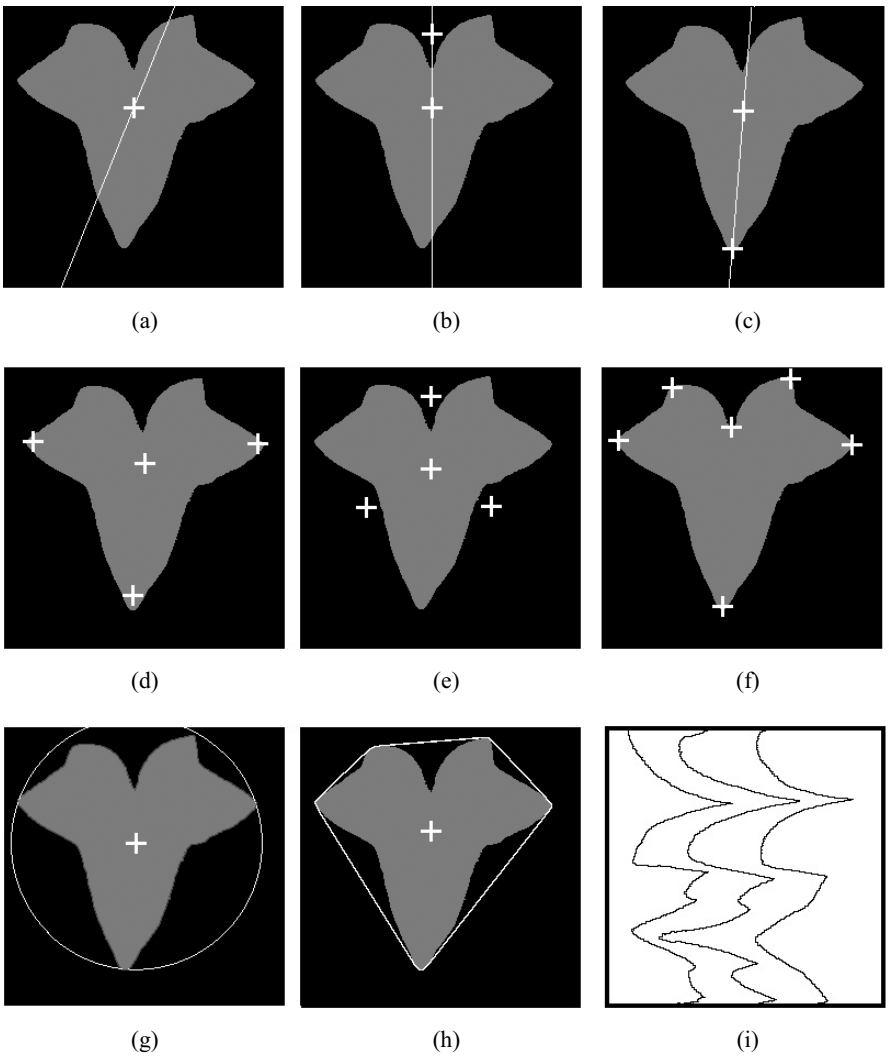


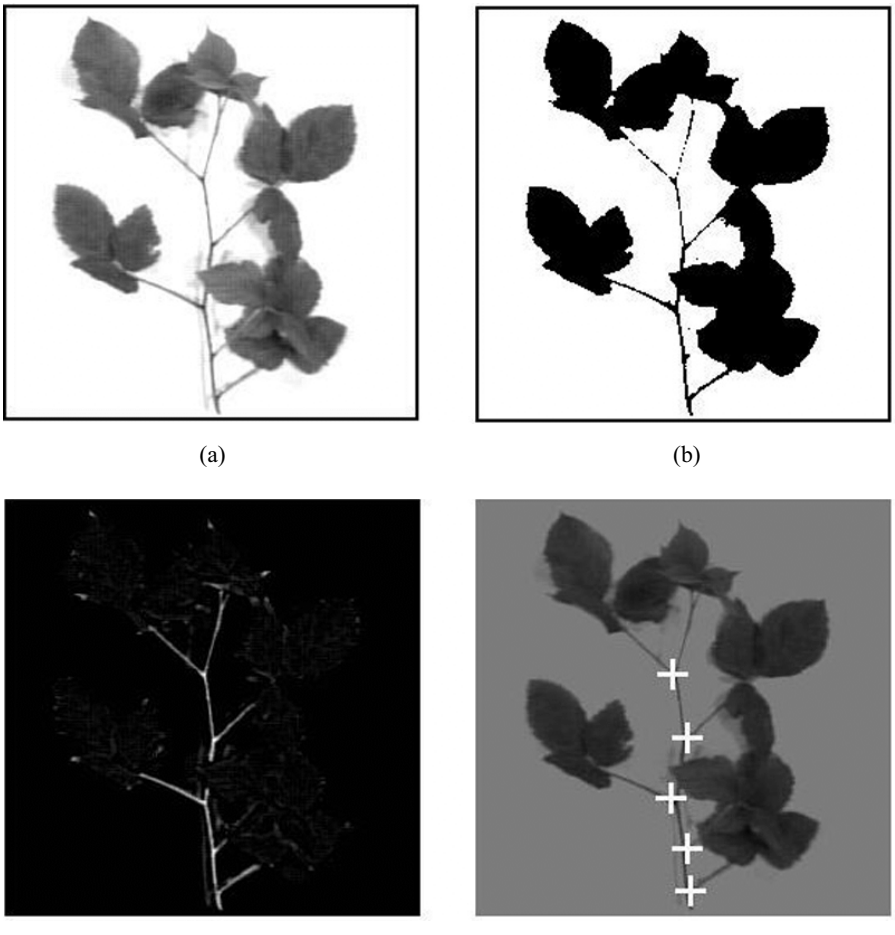
Figure 3.19. Ivy leaves: (a) principal axis; (b) line joining the object centroid and the centroid of the third largest bay; (c) line joining the object centroid to the furthest edge point; (d) tips of the protuberances; (e) centroids of the major bays; (f) points of high curvature; (g) circumcircle; (h) centroid of the convex hull; (i) polar plot ($r(\theta)$) versus θ)

Micropropagation

The technique known as *micropropagation* is used in horticulture is to "copy" a tiny living plant by vegetative reproduction, to create a large number of genetically identical plants. A plantlet is dissected, as appropriate, and the small parts are then replanted in some nutrient material, such as agar jelly. Automation of the process is of considerable economic importance, as manual handling of the plants is prone to causing physical damage and infection by micro-organisms shed by the operator.

Of course, building a visually guided robot for such a task requires considerable skills in mechanical system design, automatic control and vision systems engineering. Our sole concern here is the last of these. A plant such as that shown in Figure 3.20 has an "open" structure that is fairly easy to analyse. The specific task is locate the axial buds, located where the leaf stalks meet the main stem. There are thus two sub-tasks:

- locating the bifurcation points;
- identifying the main stem.



(a) (b)

(c) (d)

Figure 3.20. Locating axial buds for micropropagation of a small open-structure plant; (a) original image; (b) best result obtained by thresholding (notice that the stems are "broken"); (c) crack detector applied to the original image; (d) after thresholding c and removal of noise from the binary image and then skeletonisaton, it is possible to locate the bifurcations (locations of the axial buds).

The Y-shaped junctions can be located using either (grey-scale or binary) morphology, or by skeletonising the silhouette and then locating joints on it.

A plant with slightly denser foliage will produce some situations in which the bifurcation points are obscured by leaves. In this case, it may be necessary to combine the results of processing images derived from two or more cameras. This is fairly straightforward if the leaf density is not too high. As the foliage becomes thicker, the task becomes more and more difficult. Eventually, it becomes too complex to analyse properly and a "brute force" method is needed, in which we deliberately sacrifice part of the plant in order to dissect the rest of it.

Pruning

Pruning a dense rose bush can be very difficult, since it requires careful planning beforehand and continued reassessment of the situation as the process continues. There are certain practical issues relating to this task that must be solved first: lighting, eliminating ambient light, protecting the optical sub-system, placing the camera(s) to obtain a good view of each part of the plant. Each of these is far from trivial; it is difficult obtain a good overall view and close-up views of each leaf, stem and stalk. However, these technicalities are distracting us from the main thesis of this chapter, which is that designing inspection procedures for natural products sometimes requires a fundamentally different approach from that adopted for industrial installations. The main feature here is the high level of intelligence needed. In fact, the close integration of vision and intelligent planning needed for this application is already available in PIP, which employs Prolog to implement the top-level controller. However, a very severe difficulty lies in formulating the rules needed to guide a pruning robot. Moreover, tasks such as this are likely to require the discrimination of subtle conditions in the plant (colour and structural changes), due to wilting, infection, frost damage, etc. There are numerous sub-problems, all of which make this a very much more difficult process than micropropagation.

3.5.3 Semi-processed Natural Materials

Chicken-meat Butterflies

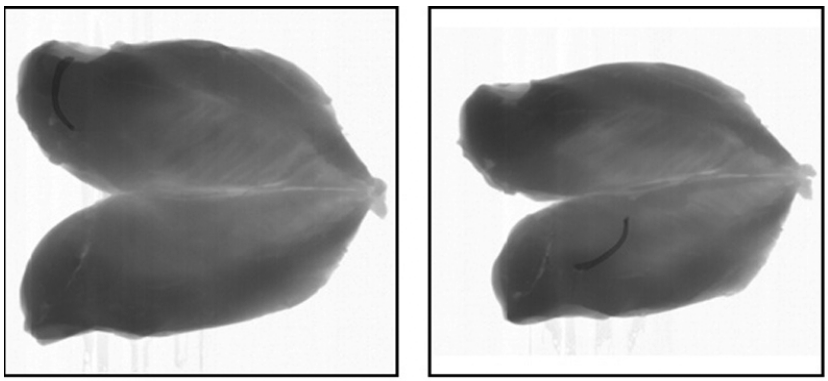
"Butterfly" is the name given to a piece of chicken meat that is obtained by cutting symmetrically around the breastbone. Figure 3.21 shows x-ray images derived from two chicken butterflies and the smoothed intensity contours (isophotes) for one of them. Careful examination of Figures 3.21(a) and (b) will reveal one bone in each sample. Notice that the intensity varies considerably over the area covered by the butterfly and this may obscure the images of any bones (Figure 3.21(d)). This arises because the thickness of the meat changes.[5] It would help a great deal, if we could reduce this intensity variation, before applying an appropriate image filter. The *crack detector,* also called morphological *grey-scale closing,* is one possibility. Fitting a suitable model to the image of a butterfly might help us in two ways:

[5] This fact can be used to measure the thickness of the meat, assuming that it is homogenous.

i to compensate for those intensity changes that are predictable, knowing that chickens have a similar anatomy;
ii. to enable us to anticipate where and what type of bones might occur. (For example, wing bones are not found embedded in breast meat.)

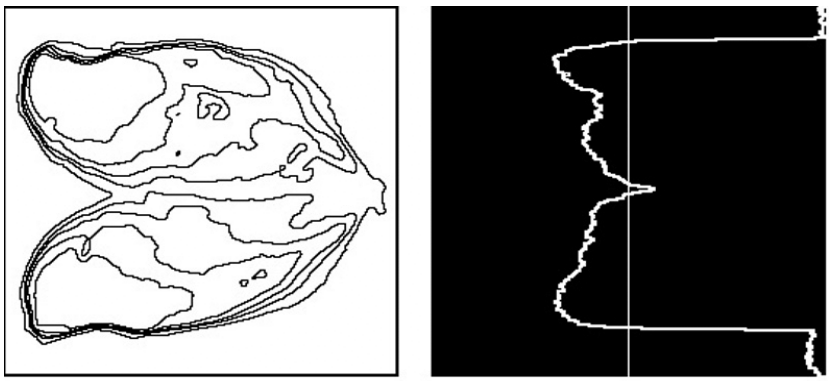
If all chickens were identical, a fixed reference butterfly image could be produced, which could then be subtracted from each sample image. Unfortunately, chickens do not oblige us by growing to an identical size, so it is impossible to create such a reference image (Figure 3.21(e)). However, it might be possible to modify the reference image, by warping and rescaling its intensity locally, so that it fits the sample image better. To guide the model fitting, we need to define a number of "anchor points", so that we can correlate the reference and sample images. We shall therefore explore some of the possibilities for doing this.

Simply thresholding the butterfly image creates a number of "islands" (Figure 3.21(c)). It may be a good idea to smooth their edges, by applying a low-pass (blurring) filter first. We can then find the centroids of these islands. By carefully choosing which islands we analyse in this way, we can obtain a number of anchor points (Figure 3.22). The centroid of the silhouette and the centroid of its largest bay provide two more such points. In Figure 3.22(b) – (d), we see that this process works, even when the butterfly is not symmetrical. Other possibilities exist for finding anchor points. For example, it is possible to fit a standard mathematical shape, such as a cardioid, to the outer edge of the butterfly, or defined intensity contours (Figure 3.22(e)).

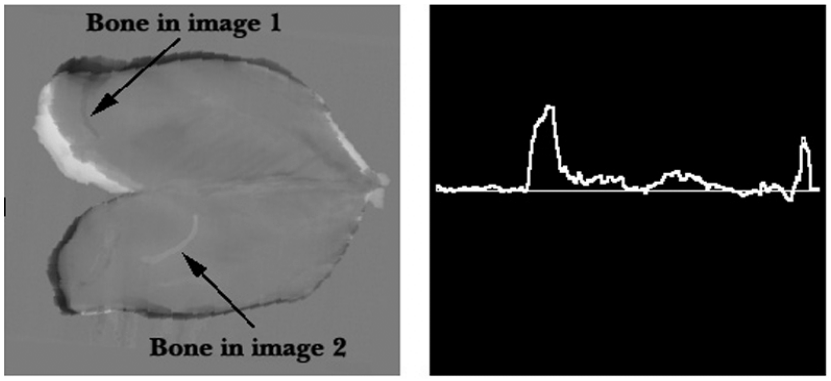


(a) (b)

Figure 3.21. Chicken-meat "butterflies": (a) unprocessed x-ray image; (b) another unprocessed image.



(c) (d)



(e) (f)

(g) (h)

Figure 3.21 (continued) Chicken-meat "butterflies": (c) intensity contours for a; (d) vertical intensity profile for a; (e) images a and b registered (without rescaling) and subtracted; (f) horizontal intensity profile for e; (g) vertical intensity profile through the centre of e; (h) crack detector algorithm applied to e.

The parameters of the cardioid could then be used to control the model-fitting procedure. Other techniques that we have already met might also be considered, although the principal axis is unreliable, if the butterfly is not approximately symmetrical (Figure 3.22(f)).

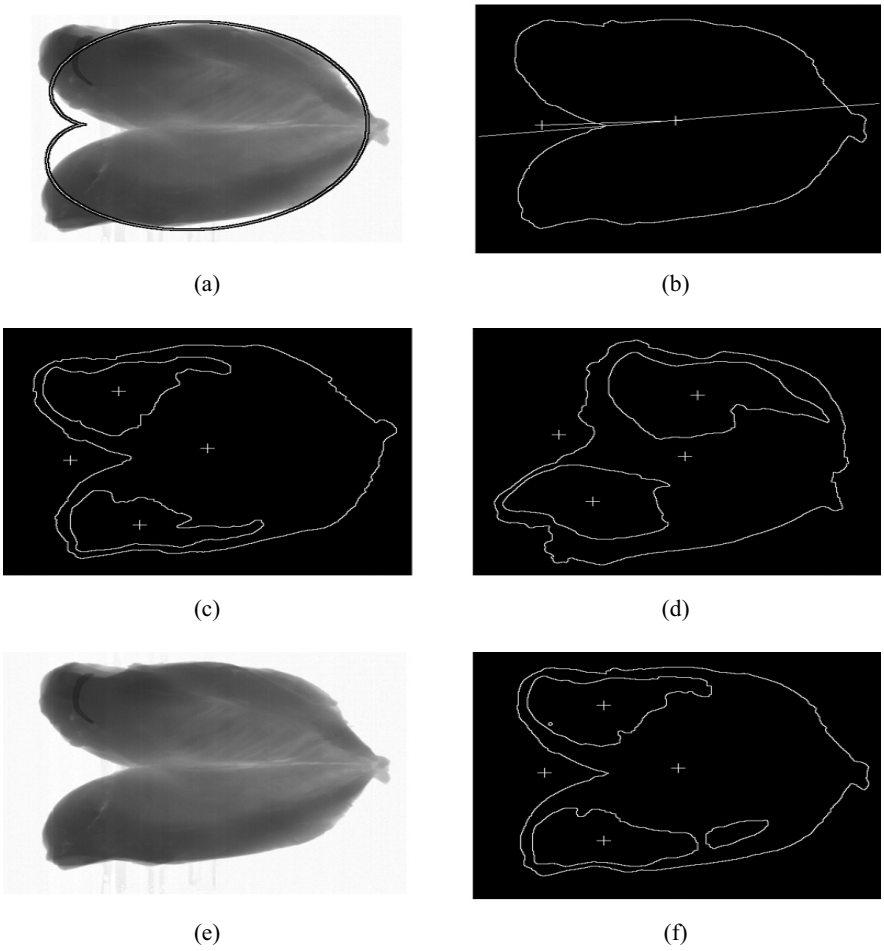


Figure 3.22. Chicken-meat "butterflies; (a) original image; (b) centroids of the outline, the largest bay and of the two largest "islands" created by thresholding at 75% of maximum density; (c) same calculations applied to another "butterfly";. (d) same calculations applied to a third "butterfly"; (e) cardioid fitted to the outline of a [Kindly provided by Dr Paul Rosin.]; f. principal axis line joining the centroid to the centroid of the largest bay.

Fish Fillets

Many of the points just described are also appropriate for processing images of fish fillets. However, the examples shown in Figure 3.23 have no axis of symmetry, or large concavity, that we were able to use on chicken butterflies. Hence some of the methods that we devised for butterflies are unsuitable for fish fillets, although new

possibilities might arise The point to note is that each application involving natural products has to be considered on its merits, so that a suitable analysis procedure can be devised to take account of its characteristic features. Moreover, what may seem like similar applications might, in fact, require quite different types of solution; both the designer and the machine he/she is building must be intelligent.

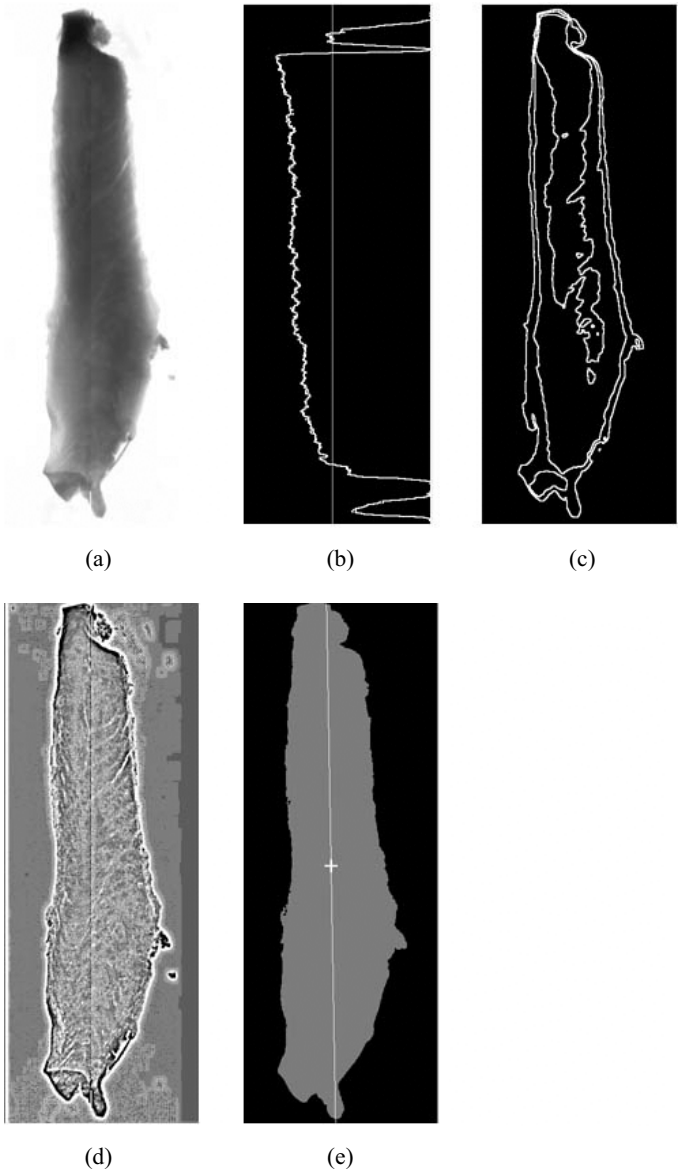


Figure 3.23. Fish fillets: (a) original x-ray image; (b) intensity profile; (c) smoothed isophotes (intensity contours); (d) high-pass filter (notice how little internal detail there is (e) principal axis.

3.5.4 Food Products

Cake Decoration Patterns

Figure 3.24 shows four decoration patterns generated by the same production machinery, making cakes in the form of a continuous strip. Patterns like these are created using one, or more, oscillating nozzles, extruding icing (frosting) onto the cake, which moves at constant speed. Among other patterns, cycloids, sinusoids and zig-zag curves can be created in this way. Although the nozzle-cake motion can be described precisely in mathematical terms, the patterns actually deposited on the cake surface are somewhat uneven. The reason is that the icing does not always flow smoothly through the nozzle. In addition, the cake surface undulates and does not provide uniform adhesion for the emerging stream of icing. The result is a pattern that follows a well-defined path but which is of uneven thickness and may actually be broken in places. Inspecting the decorated cake surface is important, because customers judge the cake, in part, by its appearance. This is an ideal application for a simple heuristic learning procedure, based on morphology and statistics such as zero-crossing frequency (Figure 3.25). Notice, however, that no single morphology operator yields sufficient information for a comprehensive test of cake decoration quality; several such filters would be needed in practice.

However, as always in Machine Vision, we need to consider the application requirements, in order to optimise the design of the inspection system. In this particular case, the nature of the manufacturing process and the types of fault it produces are significant. There are short-term glitches, when a very short section of cake deviates from the norm. These are unimportant and might actually be regarded as desirable, as a certain element of variation gives a "home made" appearance to the cakes. Small changes in the appearance of the decoration pattern should therefore be ignored. However, once the baking system starts to produce "bad" cake, it will go on doing so. Our inspection system is therefore concerned solely with long-term changes. We usually think of an inspection system as using the same algorithm, which may be quite complex, operating on every image. However, the nature of this particular application allows a different kind of inspection procedure to be used. Let us consider the continuous strip of cake as being represented by a succession of non-overlapping images, $(I_1, I_2, I_3, \ldots)$. Then, we might apply a set of "mini-algorithms" $[A_1, A_2, \ldots, A_N]$ to images $[I_1, I_2, \ldots, I_N]$ respectively and combine their results afterwards. Typically, each mini-algorithm will use different morphology and measurement processes. The results is a set of measurements: $X_1, X_2, \ldots, X_N$, which might be used as inputs to a Pattern Recognition system such as a Compound Classifier. The output of this is a decision that the cake decoration is either "good" or 'faulty". Images $I_{N+1}, I_{N+2}, \ldots, I_{2N}$ are then treated in the same way, and as are succeeding groups of N images. It is worth making several more points here:

- The process just described fits very well onto the Concurrent Processor, which is a highly cost-effective way of inspecting objects/materials on a conveyor belt [15].
- A fast implementation of the morphology operators might be accomplished using the SKIPSM technique, developed by F. M. Waltz [11].

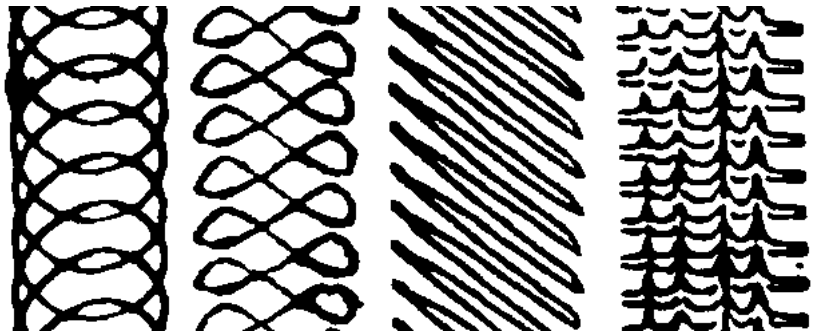


Figure 3.24. Cake decoration patterns

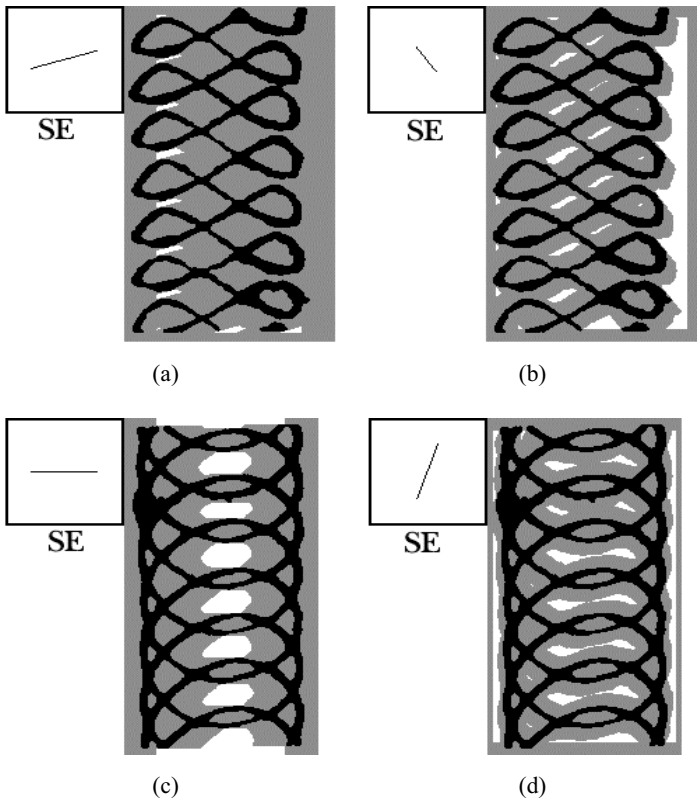


Figure 3.25. Cake decoration patterns analysed using binary morphology operators with linear structuring elements (small squares). For the purposes of illustration, the result of the processing (white regions) has been superimposed onto the binary image.

- The structuring elements used by the morphology operators might be generated. However, carefully matching the structuring element with the decoration pattern will usually result in a more efficient design.

A more complete discussion of this topic may be found in [16].

Loaf Shape

The shape of a loaf is an important criterion that affects customer choice, although nobody knows exactly what customers want. While it is extremely difficult to specify precisely what a "good" loaf is, it is possible to define some of the faults that are obviously *unacceptable*. Of course, it should not be assumed that a loaf is necessarily acceptable if we fail to show that it is unacceptable. However, we shall ignore such niceties and shall concentrate on examining loaf shape, in order to identify malformed loaves. There are two distinct approaches that we can take:

a. Analyse the shapes of individual slices and combine them to form a judgement about the whole loaf. The loaf is represented as a set of two-dimensional images, like that shown in Figure 3.26(a).
b. Analyse the shape of the whole loaf. The loaf is represented by a depth or range map [12].

Figure 3.26 shows the silhouette of a single slice of bread, taken from a loaf that is baked in a tin with no lid.

The top of such a loaf should form a nicely rounded dome. The sides should be straight and parallel, with no major indentations. The overspill (e.g., where the dough has bulged over the rim of the baking tin) should not be too small or too large (this is quantified in practice, of course). Figure 3.26 also illustrates several ways of analysing the shape of this slice. Figure 3.26(b) shows the negated Hough transform of the outer edge contour, while Figure 3.26(c) demonstrates that, if we locate the major peaks and invert the Hough transform, we can locate the sides and base of the slice. It is a straightforward matter then to test whether the sides are vertical and parallel, and have no major bulges (impossible for a tin loaf) or indentations. These same lines can be used to isolate the overspill. (Figure 3.26(d)). The top of the slice can be examined, to measure its radius of curvature. (Figure 3.26(e)).

This requires that we identify three well-spaced points on the top, to provide data for a circle-fitting routine based on simple geometry. The same principle can be used to examine the overspill, by finding the radius of curvature on both sides of the loaf (Figure 3.26(f). The radius ($r(\theta)$), measured from the centroid, as a function of angular position (θ) is plotted in Figure 3.26(g), and can be used to identify the type of slice, by cross-correlation with a standard shape. (This is the same representation of shape that we employed earlier for identifying ivy leaves.) Figures 3.26(h) and (i) show the binary bread-slice image grey-weighted in two different ways. The histograms of these two images provide useful shape descriptions that are independent of orientation and could therefore be used as part of a shape recognition procedure.

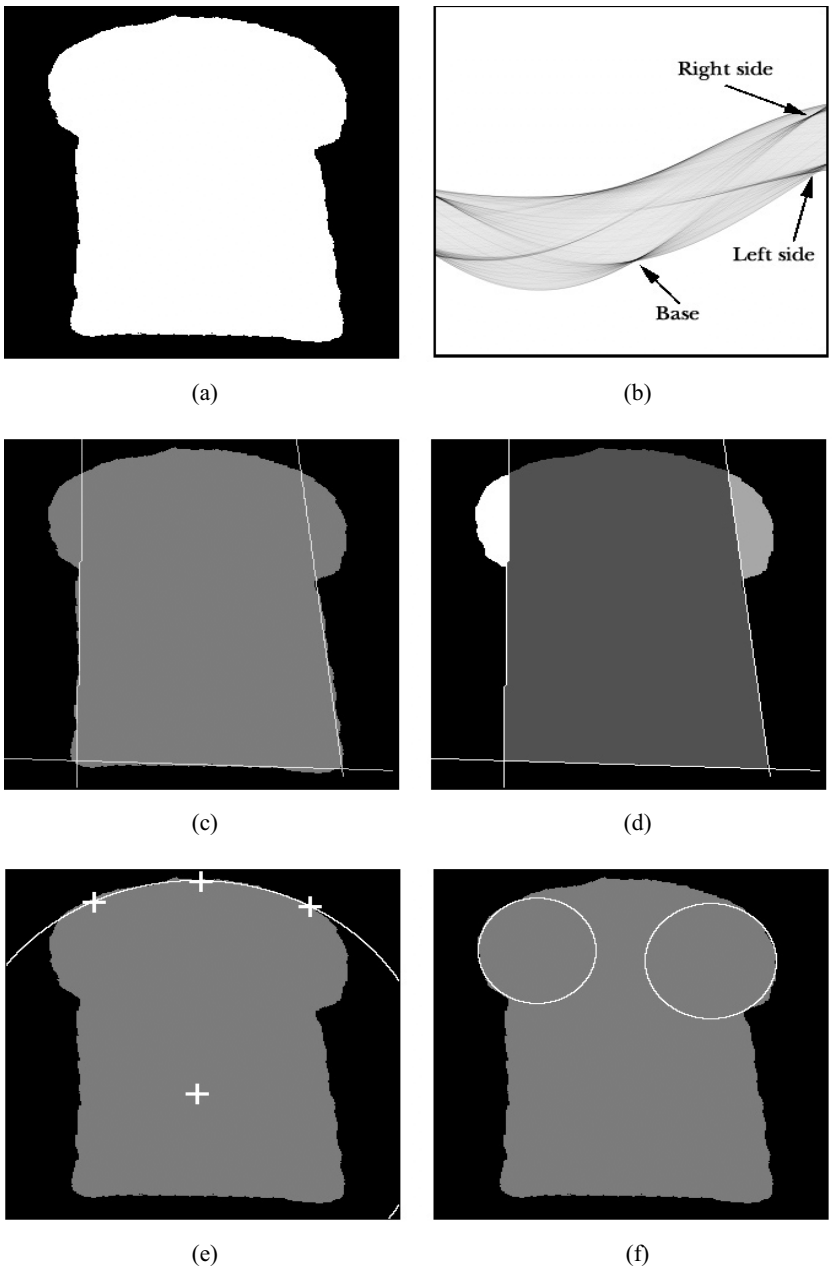
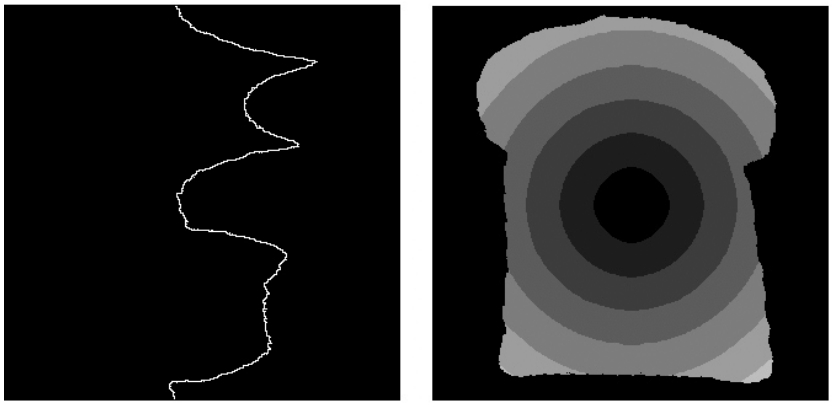
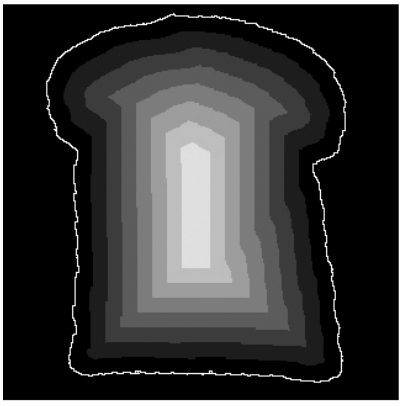


Figure 3.26. Slice of bread 2d shape analysis: (a) original (binary) image; (b) Hough transform (negative); (c) inverse transform applied to the three principal spots; (d) overspill; (e) fitting a circle to the top; (f) fitting circles to the overspill regions.



(g) (h)

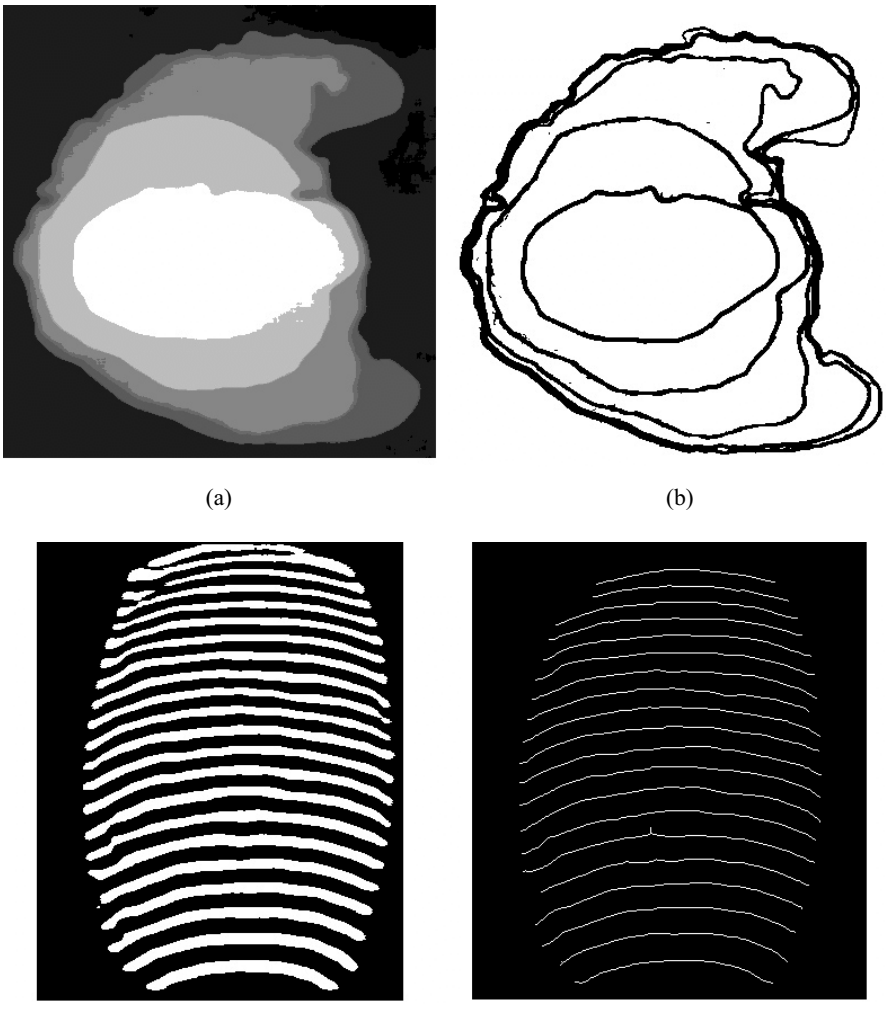


(i)

Figure 3.26 (continued) Slice of bread 2d shape analysis: (g) plot of distance ($r(\theta)$) from the centroid, versus angle (θ, vertical axis); h. grey-weighting according to distance from the centroid (the histogram of this image provides a useful way of describing shape that is independent of orientation); i. grey-weighting according to the grass-fire transform (this is another useful rotation-independent representation of shape).

The analysis described so far is based on a single slice of bread, In order to inspect the whole loaf, we could repeat measurements like those outlined above, for each slice and then relate them together. Two other ways of representing three-dimensional shape are illustrated in Figure 3.27 and require the use of specialised illumination-optical systems. The range map of a croissant shown in Figure 3.27(a) was obtained using structured lighting [16]. In this method of obtaining three-dimensional shape information, light from a diode laser is expanded by a cylindrical lens to form a fan-shaped beam. This is projected vertically downwards onto the top of the object. A video camera views the resulting light stripe from an angle of about 45˚. This yields height data for just one vertical cross-section through the object, for each video frame. To build up a complete 3-D profile of an

object, it is moved slowly past the camera. (In practice, it is possible to map only the top surface of a loaf with a single camera. Hence, in order to obtain profiles of the sides as well, this arrangement is triplicated.) In a range map, the height of the object surface is represented by the intensity in a monochrome image and isophotes (contours of equal brightness) correspond to height contours (Figure 3.27(b)). The pattern in Figure 3.27(c) is the result of simultaneously projecting a number of parallel light stripes onto the top surface of a loaf and applying some simple image processing, to create a binary image.



(a) (b)

(c) (d)

Figure 3.27. 3-D height profiling of a croissant and loaf.

We therefore appear to have three distinct methods for representing 3-D shape visually. In fact, these are very similar in terms of the data that they produce and, as a result, we can employ the same decision-making procedure to complete the analysis of 3-D loaf shape. In Figure 3.26, the slicing is achieved by physically cutting the loaf in a series of parallel, vertical planes. In Figure 3.27(a), horizontal "slicing" is performed computationally, by thresholding the range map. In Figure 3.27(c), vertical "slicing" is achieved optically. We therefore conclude this section with a brief discussion of methods for making decisions based on a series of binary contours, which may or may not be closed.

Let $\mathbf{X}^j = (X_{1,j}, X_{2,j}, X_{3,j}, ..., X_{N,j})$ denote a series of measurements derived from the j^{th} slice of a loaf, where $j = [1, 2, 3, \ldots, M]$. The $X_{i,j}$ could be any convenient measurements, perhaps of the type that we have already discussed when referring to Figure 3.26. For our present purposes, it is the procedure that we use for combining them that is of prime importance. We begin by building a device (or writing a program segment) that makes a decision $Y^j \in \{good, faulty\}$, based on just one slice. This might perform a series of checks of the form:

$$L_{i,j} \leq X_{i,j} \leq H_{i,j} \tag{3.12}$$

where $L_{i,j}$ and $H_{i,j}$ are parameters obtained by learning. They may simply be extimates of the minimum and maximum values observed for the variable $X_{i,j}$, taken over a set of training data. Alternatively, they could be limits based on statistical measures of variable spread (i.e., mean and standard deviation). We might compute the sub-decision Y^j as follows

$$Y^j = \begin{cases} good, \text{if } L_{i,j} \;?\; X_{i,j} \;?\; H_{i,j} \text{ for all } i \in [1,2,3\ldots,N] \\ faulty, \text{otherwise} \end{cases} \tag{3.13}$$

It now remains to combine the sub-decisions (Y^1, Y^2, ... , Y^M). This can be done in a variety of ways, depending upon the application requirements. We might apply a strict rule demanding that all of the $Y^j = good$, or a more tolerant rule that no more than P ($P < N/2$) of the $Y^j = faulty$. Clearly, there are many variations of this basic decision-making procedure that we can apply. For example, we could make Equation (3.13) less severe, by requiring that say only Q out of N ($Q < N$) tests must be passed before we make $Y^j = good$. Which rule we choose in practice is best left to human instinct and whatever appropriate experimentation the vision engineer can devise.

The idea of making up *ad hoc* rules like this, specially to suit a given application, is an anathema to people for whom mathematical rigour is essential for their peace of mind. However, to a vision engineer working with highly variable objects, the freedom to adopt heuristic procedures is essential. The results may not be optimal but, in a situation like this, it may be impossible to do any better, or even prove that we could ever do so.

Loaf shape analysis has been reported in more detail elsewhere [17], as has the vexed subject of texture analysis of the bread matrix [18]. Much research work remains to be done.

3.6 Concluding Remarks

The important lessons of this chapter may be summarised as:

- Many of the techniques that were originally devised for inspecting engineering artefacts can be used for natural products but may well require considerable modification.
- In view of the far greater level of variation that exists in natural products, the image analysis procedures need to be selected very carefully indeed, taking into account the characteristic features of the application. The need for a good experimental image processing "tool box" is of even greater importance than hitherto [19].
- Seemingly similar applications, may, in fact, require quite different methods of solution.
- The designer of the vision system needs to apply a high level of intelligence; he/she cannot simply mindlessly "turn the handle" to design an inspection system for natural products. He/she needs to be alert to the great variation in size, shape, colour and texture that exists.
- An inspection system for natural products is likely to require a far higher level of (artificial) intelligence than industrial systems employed so far.
- The specific technologies that are of prime importance for inspecting natural products are:
  - pattern classification and self-adaptive learning;
  - rule-based decision-making;
  - colour recognition;
  - artificial Intelligence programming techniques (Prolog);
  - intelligent model fitting.
- The standard Pattern Recognition model requires some modification to take account of the fundamental difficulty of obtaining a fully representative sample of data from the "faulty" class. This may involve learning on a single class, or a hybrid combination of rule-based and traditional (hyperspace) decision-making methods.
- We may need to adopt *ad hoc* rules and may have to accept that we can do no better than achieve a *sufficient* solution. The concept of *optimal* is one that we may have to forsake at an early stage in the process of designing a vision system.
- Many applications in engineering manufacture present similar problems to those encountered with natural products. Whenever, fluid, or semi-fluid, materials are sprayed, or extruded, onto a substrate, they flow into shapes that are impossible to predict exactly. The resulting "dollops" are likely to require treatment as if they arose naturally and to require intelligent inspection procedures.

Finally, it must be made clear that all of the principles required for applying Machine Vision to engineering artefacts also apply to natural products as well. These have been expressed in the form of a set of "Proverbs" [13]. The points made above add to that list; they do not, in any way, diminish the importance of

applying sound Systems Engineering principles. Machine Vision applied to the inspection of natural products must not, under any circumstances, be reduced to an abstract intellectual exercise. Machine Vision is not a scientific discipline – it is engineering.

3.7 References

[1] Batchelor, B.G. (1974) Practical Approach to Pattern Classification, Plenum, ISBN 0-306-30796-0.

[2] Haykin, S. (1994) Neural Networks a Comprehensive Foundation, Macmillan, Englewood Cliffs, NJ, ISBN 0-02-352761-7.

[3] Bartlett, S.L. *et al.* (1988) Automated solder joint inspection, *Trans. IEEE on Pattern Analysis and Machine Intelligence*, PAMI vol. 1, no. 10, pp. 31–42.

[4] Bratko, I. (2001) *Prolog Programming for Artifical Intelligence*, 3rd edition, Addison-Wesley, Harlow, UK, ISBN 0-201-40375-7.

[5] Batchelor, B.G. (1991) *Intelligent Image Processing in Prolog*, Springer-Verlag, Berlin, ISBN 0-540-19647-1.

[6] MacProlog and WinProlog, Poducts of Logic Programming Associates Ltd., Studio 4, Royal Vistoria Patriotic Building, Trinity Road, London, SW18 3SX, UK.

[7] Cyber Image processing software, Cardiff University, Cardiff, Wales, UK, URL: http://bruce.cs.cf.ac.uk/bruce/index.html

[8] CKI Prolog, S. van Otterloo, URL http://www.students.cs.uu.nl/people/smotterl/prolog/

[9] CIP Image processing software, URL http://www. bruce.cs.cf.ac.uk/bruce/index.html

[10] Clocksin, W.F. and Mellish, C.S. (1981) *Programing in Prolog*, Springer Verlag, Berlin, ISBN 3-540-11046-1.

[11] Batchelor, B.G. and Waltz, F.M. (2001) *Intelligent Machine Vision: Techniques, Implementation and Applications*, Springer-Verlag, ISBN 3-540-76224-8.

[12] Batchelor, B.G. and Whelan, P.F. (1997) *Intelligent Vision Systems for Industry*, Springer Verlag, ISBN 3-540-19969-1.

[13] M.K. Hu. (1962) *Visual pattern recognition by moment invariants*, *IRE Trans*., vol. IT-8, pp. 179–187.

[14] Batchelor, B.G. (1982) *A laboratory-based approach for designing automated inspection systems*, Proc. Int. Workshop on Industrial Applications of Machine Vision, Research Triangle, NC, May 1982, IEEE Computer Society, pp. 80–86.

[15] Warren, P.W. and Batchelor, B.G. (1974) Recognising familiar patterns, *Journal of Cybernetics*, vol. 3, No2, 1974, pp. 51–5.

[16] Lighting Advisor, URL: http://bruce.cs.cf.ac.uk/bruce/index.html

[17] Batchelor, B.G. (1993) Automated inspection of bread and loaves, Proc. Conf. on Machine Vision Applications, Architectures and Systems Integration II,

Boston, MA, Sept. SPIE, Bellingham, WA, USA, vol. 2064, ISBN 0-8194-1329-1, pp. 124–134.

[18] Zayas, I.Y., Steele, J.L., Weaver, G. and Walker, D.E. (1993) Breadmaking factors assessed by digital imaging, Proc. Conf. on Machine Vision Applications, Architectures and Systems Integration II, Boston, MA, Sept. pub. SPIE, Bellingham, WA, USA, vol. 2064, ISBN 0-8194-1329-1, pp. 135–151.

[19] Batchelor, B.G. and Waltz, F.M. (1993) Interactive Image Processing, Springer Verlag, New York, ISBN 3-540-19814-8.