Real-time Colour Recognition in Symbolic Programming for Machine Vision Systems.*

Bruce G. Batchelor,

Department of Computing Mathematics, University of Wales College of Cardiff,

P.O. Box 916, Senghennydd Road, Cardiff, CF2 4YN, Wales, United Kingdom.

Tel: Int+ 44-222-874390, Fax: Int+ 44-222-666182.

Email: bruce@cm.cf.ac.uk

Paul F. Whelan,

School of Electronic Engineering, Dublin City University, Dublin 9, Ireland.

Tel: Int+ 3531-704 5489, Fax: Int+ 3531-704 5508.

Email: whelanp@eeng.dcu.ie

Key words: Machine Vision, Systems Engineering, Teaching by showing, Real-time colour recognition, Prolog.

* Please address all correspondance to **Paul F. Whelan,** School of Electronic Engineering, Dublin City University, Dublin 9, Ireland. Telephone: Int+ 353-1-7045489, FAX: Int+ 353-1-7045508, Email: whelanp@eeng.dcu.ie

Real-time Colour Recognition in Symbolic Programming for Machine Vision Systems.

Bruce G. Batchelor,

Department of Computing Mathematics, University of Wales, Cardiff, Wales, United Kingdom.

Paul F. Whelan,

School of Electronic Engineering, Dublin City University, Dublin, Ireland.

Abstract: It is impossible to collect more than a tiny proportion of all of the possible examples of a given hue, to form a training set for a machine that learns to discriminate colours. In view of this, it is argued that colour generalisation is essential. Three mechanisms for learning colours as defined by a human being are described. One of these is based upon an idea developed by AP Plummer and is implemented in a commercial device known as the Intelligent Camera. This implementation can learn the characteristics of coloured scenes presented to it, and can segment a video image in real-time. This paper presents four procedures which allow the range of colours learned by such a system to be broadened, so that recognition is made more reliable and less prone to generating noisy images, which are difficult to analyse. Three of the procedures can be used to improve colour discrimination, while a fourth procedure is used when a single and general colour concept has to be learned. Several experiments were devised to demonstrate the effectiveness of colour generalisation. These have shown that it is indeed possible to achieve reliable colour discrimination / recognition for such tasks as inspecting packaging and fruit. A practical system based upon the Intelligent Camera and controlled by software written in Prolog has been developed by the authors and is

being used in a study of methods for declarative programming of machine vision systems for industrial applications.

Key words: Machine Vision, Systems Engineering, Teaching by showing, Real-time colour recognition, Prolog.

1 Introduction

1.1 Colour Recognition in a Symbolic Programming Environment

Consider the task of designing a machine to inspect printed cardboard and plastic cartons, such as those used for food products, household goods, toiletries, etc. These are often printed in several colours. For example, the container of a certain brand of margarine has a wavy yellow streak across its top, in addition to the product name, which is printed in red. Other features on the top are printed in blue (large blob-like regions) and black (small lettering). (See Figure 1.) Inspecting such a container could be achieved by first isolating the different colours and then applying conventional (i.e. monochrome) image analysis procedures to each of the colour separations. In this article, we shall discuss electronic filters that are capable of performing such a separation of colours. (Plummer 1991; Intelligent Camera 1990) A *Programmable Colour Filter (PCF)* might, for example, isolate the yellow streak on the margarine tub, so that it can be examined in detail. Once the yellow streak has been inspected, other coloured features can be treated in the same way. Of course, such a filter should be able to tolerate wide variations in the brightness of the scene being examined; it should be sensitive to colour, not intensity.

Inspecting coloured cartons, containers, etc. is a typical application for the techniques that we shall discuss. Amongst the other applications areas, we may list:

reading resistor colour codes, identifying coloured wires, inspecting iconic displays on car dash boards, etc.

To illustrate how colour recognition may be used in symbolic programming of a vision system, we shall explain how we can identify bananas, which is a rather simpler task than inspecting margarine tubs, which we shall discuss in detail later. The following is a description of a banana, expressed in formalised English.

X is a banana **if**
    X is yellow **and**
    the length of X is U **and** % U is the length in mm
    $60 \leq U \leq 800$ **and**
    the width of X is V **and** % V is the width in mm
    $10 \leq V \leq 50$.

This recognition rule for bananas[1] can be translated into Prolog in the following way. (Prolog has been used for several years as a medium for programming vision systems. (Batchelor 1991; 1992; Batchelor and Whelan 1993; Whelan 1993) and will be discussed again in this article.)

```
object(X, banana) :-
    colour(X,yellow),
    length(X,U),
    60 ≤ U,
    U ≤ 800,
    width(X,V),
    10 ≤ V,
    V ≤ 50.
```

[1] A more complicated rule for recognising bananas is needed in certain parts of the world. In Sri Lanka, for example, there are 14 different varieties of banana, including types that are bluish-green and red. There is also a considerable variation in size. In this situation, we simply need to add further clauses of the type listed here.

This demonstrates how we can incorporate colour recognition in a symbolic description of an object. A recognition rule for a multi-coloured artefact, such as the top of a margarine tub can often be expressed in a similar way as a program written in Prolog. To do this, we might isolate each colour in turn, then perform a series of simple tests, such as counting blobs, then measuring the area, and the dimensions of the minimum-area bounding rectangle for each one.

The top of the margarine tub described above can be inspected thus:

```
inspect( ) :-
        remember(parameters,[]),          % Initialise parameter store[2]
        member(X,[margarine_tub_yellow, margarine_tub_red,
                margarine_tub_blue, black]),
        isolate(X),                       % Digitise image. PCF recognises <X>
        measure(Y),                       % X is a list of measurements of white
                                          % areas
        recall(parameters,Z),             % Refer to parameter store
        append(Y,Z,Q),                    % Append new parameters to list …
        remember(parameters,Q),           % … and store the results
        fail.                             % Clause 1 always fails

inspect(margarine_tub) :-
        recall(parameters,X),             % Refer to parameter store
        ideal_parameters(Y),              % Consult the database
        euclidean_distance(X,Y,D),        % Measure similarity between X and Y
        small(D).                         % Is D small enough?
```

Notice that the goal **inspect(margarine_tub)** fails, if the tub is defective and succeeds, if it is acceptable.

We have tried to demonstrate that colour recognition can be used to good effect in a symbolic programming environment. To do so, we need a machine that can

---

[2] **remember** and **recall** are features of MacProlog (1992) and are similar in function to properties in LISP. The stored parameters can be updated, unlike instantiated variables.

quickly learn to recognise colours such as *"margarine-tub yellow", "margarine-tub red"*[3], given a few examples of each. The samples on which such an inspection machine is to be designed would most conveniently be obtained by examining a small number of margarine tubs on a production line.

1.2 The Naming of Colours

It is not difficult to find authoritative statements in the literature of colour theory about the subjective nature of colour perception. For example:

> *"One cannot, strictly speaking, measure colour instrumentally, because it is a subjective sensation. … In everyday conversation, one compares colours to some readily understood physical standard, e.g. orange, lemon, cream".* (Chamberlain and Chamberlain 1980)

and

> *"Careful observations by qualified observers have shown, in fact, that the defining features of the concept of colour are not, and probably cannot be, realised at will, even in the experimental laboratory". (*Optical Society of America 1953).

---

[3] Colours such as *"margarine-tub yellow"* and *"margarine-tub red"* are probably specified, by the printer, in terms of some standard colour atlas, such as the Munsell Book of Color. (Munsell Color Co.) However, for systems reasons, a colour atlas cannot be used as the basis for programming a real-time colour recognition machine working in a factory environment. The terminology used (for the colours of cartons) in a food processing plant, as distinct from a printing works, is likely to be informal and non-scientific; factory workers would certainly prefer to use to the term *"margarine-tub yellow"* rather than *"5Y 5\4"*, which is a typical Munsell colour label.

The axiom on which this article is based is that the names of colours cannot be defined mathematically. The standard CIE (1931) Chromaticity Diagram, reproduced in Figure 2, should properly be regarded as a conceptual aid, since it cannot form a precise basis for discriminating between colours. The position of the boundary between any two named colours in the Chromicity Diagram is plotted for an hypothetical *standard observer*, working in carefully controlled lighting conditions. In a practical situation, however, an industrial machine vision system is likely to be taught by a person untrained in colour science, working in a factory environment, where the lighting is highly variable. Schettini (1993) points out that camera, lighting and filter combinations affect the RGB values measured by a video camera.

Several authors, have represented the colours of ordinary everyday objects as points plotted on the Chromaticity Diagram and some of these are plotted in Figure 2. (Chamberlain and Chamberlain 1980). However, it should be noted that each point represents just one instance of a broad class of objects. The set of all ripe tomatoes, for example, is represented more accurately by a cluster of points, while ripening tomatoes generate a broad serpentine curve in the Chromaticity Diagram. The Chromaticity Diagram does not include definitions for the range of such colours as *"margarine-tub yellow"* or even *"sky blue"*.

The fact that people recognise colours by some mental process that is not fully understood simply has to be accepted. (Chamberlain and Chamberlain 1980; Optical Society of America 1953) The authors suggest that a colour recognition filter used when inspecting artefacts such as food packaging, household good and pharmaceutical cartons could be designed using the principle of teaching-by-showing.

1.3 Notation

The set notation introduced in this section allows us to define colour generalisation process in formal mathematical terms. Generalisation is seen as being an essential function in any learning system.

Let <X> denote the set of colours of objects in that class defined by human beings and which is called X. Thus , <banana> is the set of colours of bananas, while <green> is the set of colours of those objects that are referred to as "green" by human beings. A machine that is designed for colour recognition, might well use the conventional RGB colour separations. To take account of this fact, we shall therefore take {X} as being the set of all of those (R,G,B)-vectors that can be associated with the label X. Notice that in this notation, <X> is defined by a person, while {X} is a set of 3-element (R,G,B)-vectors, derived by a machine.

1.4 Recognition and Generalisation of Colours

We are all familiar with certain classes of colour, such as <yellow>, <leaf green>, <sky blue>, etc., while others are quickly learned, when we are engaged in certain specific activities, such as gardening, cooking, manufacturing motor cars, etc. For example, <daffodil> is quickly learned by Welsh children, since the daffodil is the national emblem of Wales. Obviously, <daffodil>, <canary>, <banana> and <lemon> are all proper sub-sets of <yellow>.[4] Hence, we may regard, the last mentioned as a more general colour concept than the others.

---

[4] Clearly, we are ignoring albino canaries, unripe / diseased fruit, red bananas from Sri Lanka and mutant daffodils.

Implicit in our approach to colour recognition is the concept of teaching by showing. It is important, of course, to make the maximum use of each colour sample, since they may be difficult and / or expensive to collect. It is impossible, in practice, to obtain more than a very small proportion of all the colours of a class such as <yellow>, so we must teach our machine using a few well chosen samples and leave it to generalise. Generalisation is universally accepted as being essential in *all* pattern recognition machines, of which the PCF is an example.

Given that <daffodil> $\cup$ <canary> $\cup$ <banana> $\cup$ <lemon> $\subseteq$ <yellow> it is reasonable to expect that {daffodil} $\cup$ {canary} $\cup$ {banana} $\cup$ {lemon} $\subseteq$ {yellow}. Now, we want to find some operation upon the set {daffodil} $\cup$ {canary} $\cup$ {banana} $\cup$ {lemon} which will generate an enlarged set E, such that

$$E \supseteq \{\text{daffodil}\} \cup \{\text{canary}\} \cup \{\text{banana}\} \cup \{\text{lemon}\}$$

and

$$\forall X: X \in E \rightarrow X \in \{\text{yellow}\}$$

An important but ill-defined condition is that the set E should be as small as possible, thereby avoiding over-generalisation.

This is one of the two types of colour generalisation we discuss in this paper. It is appropriate for those situations in which we are interested in *colour recognition* (single colour class), as distinct from *colour discrimination* (more than one colour class).

We shall present one procedure for generalisation in colour recognition. (*Procedure 4* defined below) A different type of colour generalisation is needed when we have to discriminate between colours. For reasons of economy, we might, for example, need to use a small data set to learn to distinguish between {apple} and

{tomato} and wish to make the discrimination more reliable, so that colours in these sets that were not represented in the training data are classified appropriately. We shall describe this type of colour generalisation in more formal terms later, after we have described how colour recognition can be performed in electronic hardware.

2 Colour recognition

2.1 Previous work

The inspection of coloured objects and surfaces by machine has, of course, been studied by numerous researchers over many years. Particular attention has been paid to the characterisation and matching of subtle colouring of fabrics, paint, paper, printing and automobiles. Since very precise colour measurement is needed in these areas, non-imaging techniques have been widely used. It should be understood that the approach that we have taken is quite different, since we are concerned with relatively coarse, high-speed recognition and discrimination processes. A typical application for the techniques we shall discuss is the inspection of packages and containers for food, domestic goods and pharmaceutical products on a factory production line.

The recognition of crops, grass, scrub, forests, water and other types of ground cover from colour and multi-spectral satellite / aerial imagery has also been studied extensively, since the early 1970s. During that early period, one of the authors (BGB) applied pattern recognition methods to data consisting of 3- and 4-element vectors describing the spectrum of the light leaving each point in a scene. (unpublished) Various classification techniques were used, including nearest neighbour, potential function and compound classifiers. (Batchelor 1975) Generalisation is an implicit function in all of these classification techniques. Furthermore, it is a trivial matter to

bias any of them, so that the likelihood of generating a selected type of error can be diminished to an arbitrarily small value. Self-adaptive learning methods developed for these classifiers lend themselves to teaching by showing. More recent work, by Stoksik et al (1991) used a Neural Network for colour recognition. They reported good accuracy of recognition, on a set of fifty colours. Broadly similar work has also performed by Yu et al (1993) and Parkkinen et al (1988). Among the other recent approaches to colour recognition has been one based on colour histograms (Swain and Ballard 1991), while Syeda (1991) discusses how (Euclidean) distances in the CIE Chromaticity Diagram can be used to estimate similarity of colours. The detection of clusters in RGB-space can be used as prelude to identifying regions of "similar colour" and assigning symbolic labels to them. Clustering can be achieved in a variety of ways. For example, Lambert and Batchelor (1991) used a scanning technique based upon the Hilbert curve to locate clusters in RGB-space. Kanamori et al (1990) simulated a colour recognition and interpolation (i.e. generalisation) algorithm that is capable of being implemented in real-time hardware. Klinker (1993) argues that the physics of light reflection / scattering be used to provide clues for the segmentation of colour images. Despite the title of her book, there is no consideration of image understanding in the sense that we use the term here.

Applications of colour in industrial machine vision systems were discussed in recent articles by Zuech (1991), Mital et al (1990). Among the industries that present possible applications for colour recognition are: glass recycling (Foran 1990; Gibboneym 1990; Gitzhofer 1991), textiles (Keesee and Aspland 1988), electronics (Capson and Eng 1988; Xie and Beni 1991; Barth et al 1992; Blaauw et al 1993), food processing (Doney 1991; Jungman and Lozano 1986), pharmaceuticals (Zuech 1991), packaging (Doney 1991). Agricultural applications include harvesting (Harrell et al 1989; Slaughter and Harrell 1989), grading of peaches (Delwiche 1989; Miller and Delwiche 1989; Thai and Shewfelt 1991), inspection of tomatoes, (Thai et al 1990), corn leaves (Tarbell and Reid 1991) and grain (Zayas and Steele 1991).

Robotics applications of colour recognition have also been described. (Barschdorff et al 1988; Zheng et al 1991; Kamel and Elmaghraby 1988; Zuech 1991) Medical applications include the detection of tumours on human skin. (Umbaugh et al 1992)

In a recent paper, Batchelor (1992) explained how a programmable colour filter, based upon a look-up table, can be used in Prolog programs. Batchelor and Whelan (1993) first introduced the theme which is explained in expanded form here.

2.2 Real-time Recognition of Colours in Electronic Hardware

Figure 3 shows the block diagram of a colour recognition system designed by Plummer (1991). This is built into a small self-contained commercially available image processing unit, called the Intelligent Camera (1990). The authors used the Intelligent Camera, in conjunction with control software written in Prolog (Batchelor 1991; 1992) in the experiments reported below. Another implementation using a real-time RGB/HSI converter chip (Umbaugh et al 1992) is suggested in Figure 4. A third implementation relies upon the use of the *xy* parameters used to define the standard Chromaticity Diagram, see Figures 2 and 5. These last two configurations have not yet been implemented.

Notice that in Figures 3 - 5, the output from the Look-Up Table (LUT) is a stream of 8-bit values, which may be regarded as forming intensities in a monochrome image. This image can be analysed in a conventional monochrome image processing sub-system. All of these hardware systems can be fully simulated in a software environment, but not necessarily in real-time.

The authors have shared conversations with several people who are dismissive of the approach to colour recognition exemplified by Figure 3, because hue, saturation and intensity (H,S and I) are not computed explicitly. While it is the accepted wisdom

that the HSI representation is better able than RGB to discriminate colours as we perceive them, this hardware arrangement is, in fact, quite general, since the LUT in Figure 3 can be programmed to generate H, S and I, given R, G and B. Hence, Figure 3 is able to implement any functions which Figures 4 and 5 can. A further advantage of Figure 3 is that it relies upon cheap standard memory devices, rather than custom ICs or real-time divider circuits. Our discussion hereafter is based upon the system using a LUT with RGB inputs, as illustrated in Figure 3.

The LUT forms the heart of the *Programmable Colour Filter*. The use of high-speed random access memory (RAM) to form a look-up table, together with "flash" analogue-to-digital converters, makes the PCF very fast indeed. It is well able to perform transformations upon a digitised video signal, in real time. Training the PCF consists of calculating appropriate values for each of the LUT's 262144 ($= 2^{18}$) storage cells. We shall describe the training process in a little while.

Since the outputs of the RGB colour channels are each limited to a finite range, it is convenient to define a cube in RGB-space within which the vector (R,G,B) is allowed to wander. This defines the so-called *Colour Cube*. (See Figures 7(a), 13 and Appendix). A plane inclined at angles of 45° to the RGB coordinate axes intersects this cube defining an equilateral triangle, called the *Colour Triangle*. Hue and saturation are related to the position of a point in the colour triangle. The *orientation* of any vector (R,G,B) can be determined by the point of intersection of that vector with the colour triangle. The vector (R,G,B) is extended, if necessary, so that it intersects the colour triangle.

2.3 Programming the Colour Filter

Training the PCF consists of three processes. See Figures 6 and 7.

(i) *Projecting all RGB vectors onto the colour triangle*, which thereby contains the so-called *colour scattergram*. This process is defined in Figure 7(a). Let **Γ(X)** denote the process of projecting a point **X** from the colour cube onto the colour triangle. Then, in Figure 7(a) we see that **Y** = **Γ(X)**. The Appendix explains how **Γ(X)** is calculated.

(iii) *Processing the colour scattergram*, to form a synthetic image, S. (Image S is not a "picture of" anything. It is merely a convenient representation of the distribution of colours within the input). How this image can be created forms the main subject of this article.

(iii) *Projecting each point, **Y**, in image S back into the colour cube*. This process is therefore called *Back-projection*. Every vector **X**, within the colour cube, that shares the same values of hue and saturation as **Y**, is assigned a number equal to the intensity at **Y**. These are the values stored within the look-up table. (See Figure 7(b).) Let **Φ(Y)** denote the result of back-projecting a point **Y** within the colour triangle through the colour cube; **Φ(Y)** defines a set of points and is explained in the Appendix.

An understanding of the colour triangle and colour scattergram, their relationship to the colour cube and the two projection techniques outlined in steps (i) and (iii) are essential to the comprehension of the remainder of this article.

Once the PCF has been programmed, the colour recognition process takes place in real-time and does not increase the computational load needed for image analysis in any way.

3 Colour generalisation

3.1 Colour generalisation in formal terms

We have already defined colour generalisation in relation to the *recognition* process. We are now in a position to be able to do the same for colour *discrimination*. Suppose that there exists a set of colours $<P_1>$, $<P_2>$,…, $<P_n>$ and that we have been able to procure a small sample of each one. These samples will be denoted by $\{A_1\}$, $\{A_2\}$,…, $\{A_n\}$. From each of these, we generate a series of distinct blobs in the colour triangle. These will be denoted by B_1, B_2,…, B_n. (Each of the B_i denotes a set of white points in the colour triangle and represents the result of some data-reduction process applied to the colour scattergram derived from $\{A_i\}$. The colour scattergram is created by evaluating $\mathbf{\Gamma(X)}$, for all $\mathbf{X}$ within $\{A_1\} \cup \{A_2\} \cup \ldots \cup \{A_n\}$. Multiple "hits" are recorded in the colour triangle by increasing the intensity stored at position $\mathbf{X}$.) See Appendix. Then

$$B_1 \cup B_2 \cup \ldots \cup B_n \subseteq \{\mathbf{\Gamma(X)} \mid \mathbf{X} \in \{A_1\} \cup \{A_2\} \cup \ldots \cup \{A_n\}\}$$

When we apply a generalisation procedure in colour discrimination, we generate a series of sets C_1, C_2,…, C_n, from B_1, B_2,…, B_n, so that $C_i \supseteq B_i$, $i \in [1,n]$.

The blob defined by C_i is obtained by enlarging the corresponding blob defined by B_i. (Figure 6) However, we also need to impose an important condition on the blob expansion procedure: since B_i and B_j are disjoint before expansion, C_i and C_j must

also be disjoint. This condition must be obeyed for all values of i and j. Then, by applying **Φ(X)** to each point in C_i, we hope to obtain a superset of A_i and more specifically

$$\{A_i\} \subseteq \{Pi\} \subseteq \{\mathbf{\Phi(X)} \mid \mathbf{X} \in C_i\}$$

Let us summarise this process. We have a series of colour classes <Pi>, $i \in [1,n]$, which have been sparsely sampled and measured using the hardware, thereby generating the $\{A_i\}$. From $\{A_i\}$, we have been able to generate a blob B_i in the colour triangle and which is then expanded to form the blob C_i. By projecting C_i back into the colour cube, we define a volume that is bigger than $\{A_i\}$ and we hope also bigger than $\{Pi\}$. The trick is to do this without allowing the C_i to overlap and without making $\{\mathbf{\Phi(X)} \mid \mathbf{X} \in C_i\}$ excessively large.

Procedures 1 - 3, defined below, are implementations of this broadly defined scheme for generalisation in colour *discrimination*, as distinct from that intended for colour *recognition. (Procedure 4)* Let us turn our attention now to the practical implementation of these ideas.

3.2 Procedures for Colour Generalisation.

The colour scattergram may be displayed as a grey-scale image, in which intensity indicates the number of pixels with the same values of hue and saturation. (See Figures 8 - 11 for some results.) The colour scattergram must be simplified before any further processing takes place. An obvious step is to threshold it, This process will generate a compact blob for each region of similar colours. See Figure 6, for example. It is often difficult to choose the value for the threshold parameter with any degree of confidence. Choosing too high a value will result in a binary image containing a large spot surrounded by a cloud of small spots, many of them single, isolated pixels. This is unsatisfactory, since the outliers have the same weight as those at the centre of the

cluster, which are much more representative of the general population of points within the input. Outliers are also prone to the effects of camera and quantisation noise. By placing the threshold too high, tolerance of colour variation in the input is lost. As a result the PCF output will consist of a noisy image in which large regions are not recognised properly, even though they may be identical to areas in the scene used during training of the PCF.

Our approach to colour generalisation consists of adjusting the sizes of the blobs created by thresholding the colour scattergram. It will be necessary to do this in such a way that blobs which were distinct when the (multi-cluster) scattergram was first thresholded, remain separate.

In a typical application, a number of coloured scenes are used to design the PCF. As each scene is being viewed, a scattergram is generated in the colour triangle. Noise is then removed from the scattergram, using common image processing operations, such as low-pass filtering. This is followed by thresholding. If the input scene consists of a single colour, such as <margarine-top yellow>, thresholding the scattergam, after clean-up, creates a single blob, B_i. This process is repeated for each colour we wish to use to design the PCF. As each new blob (B_i, i = 1,…,n) is generated, it is superimposed on the colour triangle. Therefore, prior to generalisation, the colour triangle consists of a number of blob regions, each of which corresponds to one of the trained colours. The aim of the generalisation procedures is to expand these regions, forming the regions C_i, i = 1,…,n. By projecting the C_i, back onto the colour cube, we generate the contents of the PCF look-up table. If the C_i have been generated appropriately, the resulting colour recognition process is more reliable, than it would have been if the smaller blobs B_i had been used instead.

Here are the definitions of four suggested procedures for colour generalisation. Two more generalisation procedures will be described later.

Procedure 1: Simple Dilation.

Each blob, B_i, in the colour triangle is dilated (expanded) by single a pixel for a fixed number of iterations. The number of iterations is denoted by the variable N. The resultant blob is C_i. Notice that this is a standard binary morphological operation applied to a derivative of the colour scattergram.

Procedure 2: Dilation with preservation of connectivity.

A single layer of background pixels is stripped from the (binary) image in the colour triangle. Unlike the previous approach, pixels critical for connectivity are retained. The number of iterations in this 'onion peeling' operation is denoted by the variable N. Again, this function can be performed by a standard binary morphological operator.

Procedure 3: Watershed.

This approach involves finding the watershed for each of the blobs B_i. The watershed is generated by finding the medial axis transformation of the image background.

(Figure 8 shows the results of applying *Procedures 1 - 3* to the colour scattergram derived from several samples of fruit. A comparison of their performances can be found in Table 1.)

Procedure 4: Convex hull generalization.

The convex hull is drawn around the set of blobs B_i (i = 1,…,n) in the colour triangle. It is reasonable to expect that points within this convex hull will

correspond to a generalisation of the observed colours, $\{A_i\}$, i = 1,…,n. (An example of this approach to generalisation can be found in Figure 11.)

It should be understood that *Procedures 1 - 3* are intended for use in those situations where colour *discrimination* is required, whereas *Procedure 4* is more appropriate for those occasions when simple colour *recognition* is needed.

4 Demonstration of Colour Recognition

Our first experiment to demonstrate the effectiveness of the three colour generalisation procedures defined above used three samples of fruit: an orange, a Golden Delicious apple (yellow-green) and a Star Crimson apple (yellow-green and deep red). Figure 8(a) was obtained by thresholding the composite colour scattergram, derived from these three items of fruit. In the case of the Star Crimson apple, only the red portion was used to generate the colour scattergram. In Figures 8(b - d), these blobs have been enlarged, using *Procedures 1 - 3*, respectively. In each case, a PCF was then programmed. The accuracy of recognition was measured by finding the percentage of pixels in the image generated by the PCF that had the correct value. The performance statistics are shown in Table 1. Notice that the recognition rate for the yellow-green region of the Star Crimson apple, improved from 2% to over 99.9%, even though this had not been included in the original set of training data.

The second experiment was based upon a set of six coloured stripes. (Figure 9) The colour scattergram was generated in the usual way. This consisted of a set of six bright clusters. After thresholding, the colour triangle contained six small white spots, which were then enlarged, using *Procedures 2 and 3*. In both cases, a PCF was then programmed, in the usual way. When these PCFs were applied to the same scene, the

images shown in Figures 9(b - d) were obtained. Notice the reduction in the noise level as the blobs are enlarged.

In our third experiment , we used a Golden Delicious apple to train the PCF, see Figure 10. The resulting filter was then applied to a high quality printed picture of the visible spectrum. (This will be referred to as <rainbow>.) By testing the PCF on <rainbow>, we have an objective means of measuring the extent of the colour generalisation. The process was repeated with the blob in the colour triangle expanded by two different amounts, see Figures 10(b - c). The expected broadening of the range of colours recognised by the PCF took place. However, we had not anticipated that the extension would be so asymmetric. The cause of this asymmetry is still being investigated using a prism, or grating to generate a spectrum from white light. (We suspect that the asymmetry is due to the fact that a printed spectrum is a poor approximation of an optical spectrum.)

Our last experiment was devised to demonstrate the learning of generalised colour concepts from a few specific examples. Four samples of <green> were obtained, from printed materials (magazine covers). The colour scattergram of each sample was obtained and was found to consist of a single bright compact cluster, surrounded by a small halo of low intensity. Three of the scattergrams were thresholded at the same level, yielding a set of three small, nearly elliptical blobs, B_1 - B_3. These were merged to form a single image and the PCF was programmed on it. (Black blobs in Figure 11) The result was a filter that did *not* recognise the fourth sample of <green> properly. (Recognition rate: < 1%) The next step in the experiment was to form the (filled) convex hull of B_1 - B_3. The resulting image was then used to reprogram the PCF. The recognition rate on the fourth sample of <green> was much higher than before (63 %). Examination of Figure 11 shows why this is so: the scattergram of the fourth sample does not overlap the union of B_1 - B_3 but does overlap their convex hull to a significant extent. In this simple experiment, which was

devised purely for illustrative purposes, the convex hull was able to generalise three similar but distinct colours, thereby enabling a fourth sample to be recognised with increased accuracy. In practice, learning a very general colour concept, such as <green> would require far more samples than the three we have used here.

The experiments reported in Figures 8 - 11 were performed using the system configuration outlined in Figure 3.

5 Colour Recognition and Generalisation in Prolog

5.1 Interactive procedure for designing a PCF

A broad range of software tools has been added to Prolog+ (Batchelor 1991; 1992; Batchelor and Whelan 1993), to facilitate the programming and use of a PCF. These can be operated interactively, using pull-down menus, or by incorporating the appropriate goals in a Prolog+ program. A colour filter which can *simultaneously* recognise several colours, such as <margarine_tub_yellow>, <margarine_tub_red> and <margarine_tub_blue>, can be designed interactively, in just a few minutes. The following steps are typically needed in an application, such as inspecting margarine tubs. (Prolog+ goals are enclosed in brackets and are set in bold-face type. Also see Figure 12.)

1. Plot the colour scattergram derived from the margarine tub. This simply involves selecting one item from a pull-down menu. (**colour_scattergram**)
2. Examine the colour scattergram and choose a suitable threshold parameter. (**thr**)

3. Apply some standard noise reduction operator. (e.g. eliminate very small blobs)
4. Shade the remaining blobs in the colour triangle, so that each one has a different intensity. Associate these intensity values with the symbolic colour labels: <margarine_tub_yellow>, <margarine_tub_red> and <margarine_tub_blue> (**shade_blobs**)
5. Apply one of the generalisation procedures, selected using a pull-down menu. (**colour_generalisation**)
6. Program the PCF (**program_pcf**)
7. Test the resulting PCF and, if necessary, repeat Steps 5 and 6. It may be necessary to adjust the degree of generalisation interactively.

Although the total time required to design and test a PCF for an inspection task, such as detecting printing faults on margarine tub lids, is likely to be less than than 5 minutes, a reliable and, of course, rapid inspection procedure is achieved.

In addition, a number of standard colour recognition programs are provided, for the convenience of the Prolog+ system programmer . For example, colour filters have been devised for recognising the familiar "named" colours e.g. <red>, <yellow>, <orange>, etc. but, of course, the response can only be guaranteed as being correct for nearly pure tints. As we saw in Section 1.1, Prolog+ programs can be written including goals such as **isolate (yellow)**, which generates a binary image whose white areas identify regions which are <yellow> and black areas which are <not yellow>.

5.2 Equivalence of concepts about colour

A region of nearly constant colour is mapped into a blob in the colour triangle (using **colour_scattergram, thr(X)**, where X has previously been instantiated to some suitable parameter value). This defines an image, which can be processed using any of

the normal Prolog+ operators. A blob, however it was created, can be used to define the behaviour of a colour recognition filter, using **program_pcf**. Thus, operations upon colours, i.e. union, intersection and generalisation, can be regarded as being equivalent to binary image processing operations. (**max**, **min** and **expand_blob** respectively). This duality is similar to, but subtlety different from, that implied when we represent and manipulate sets using Venn diagrams.[5] In fact, there are several levels at which we can see equivalencies among the various concepts / terms relating to colour. These are summarised in Table 2. The transformation from the colour domain to image format is performed by **colour_scattergram**, while the "inverse transform" (i.e. programming the PCF) is performed using **program_pcf**. We can also relate both of these representations to symbolic concepts and operations expressed in Prolog+.

5.3 Representing colours by a set of overlapping discs

A blob in the colour triangle can be approximated, to whatever level of accuracy we may choose, using the union of a set of overlapping discs, or rectangles. Thus, a blob may be represented approximately by a list of the following form:

[[X1,Y1,Z1], [X2,Y2,Z2], …, [Xn,Yn,Zn]]

where [Xi,Yi] is the centre of the i^{th} disc and Zi is its radius. Generating the list of parameters is achieved using **find_disc(L)**. Using a list of the form just given, we can manipulate colour parmetrically, using Prolog+. At any time, we can redraw the discs as an image, using a predicate **draw_discs(L)**, where L is instantiated to a list of lists, each containing three elements.

---

[5] The major difference is that whereas Venn diagrams do not indicate the relative sizes of two or more sets, whereas the colour triangle does.

Using this notation, we see that there exists another technique for colour generalisation (in recognition, as distinct from discrimination); we simply enlarge each of the discs. For example, we may simply add a constant to each of the Zi, or multiplying each one by a constant factor. This defines **Procedures 5** and **6** for Colour Generalisation.

6 Discussion and conclusions

The idea of using a look-up table to perform colour recognition is not new but has considerable appeal for such tasks as recognising (ripe) fruit on a tree, recognising resistor colour codes, tracing wiring, inspecting food products, cartons and pharmaceutical packaging. It lends itself to implementation in fast electronic hardware. Commercial equipment has been available for colour recognition, for some years.

The colour scattergram is a useful tool, which allows the user to associate areas of the colour triangle with colours that he / she can recognise and name. Once a colour scattergram has been generated, the user can think about colour in convenient terms, using the concepts of blob position, shape and size. He / she can also apply a wide range of image processing operators to the colour triangle. This is possible because the colour triangle is an image, like any other. After processing of the colour scattergram is complete, the back-projection procedure allows the contents of the PCF look-up table to be fixed. Thereafter, colour recognition is achieved in real time, although the subsequent procedures for image analysis (i.e. Prolog+ programs) may not be.

Six techniques for colour generalisation have been described and have been studied extensively by ourselves, using an interactive image processing system. As a result of their experience, the authors are convinced that the techniques described above provide a useful addition to the range of facilities available for recognising colours. It is possible to extend the range of colours recognised by the PCF to any extent desired. In *Procedure 2*, for example, the parameter N can be adjusted at will; if N is increased, the degree of generalisation will become higher. It is possible for a person, working with an interactive system, to experiment with the colour generalisation parameter, to obtain the best results for a given application. On the other hand, a (Prolog+) program can be written which chooses a suitable value for N, according to some pre-defined criterion. A discussion of this advanced topic is, however, beyond the scope of this communication.

Clearly, there are many other ways to extend the blobs in the colour triangle which determine the behaviour of the PCF and the reader will have no difficulty in suggesting several others, that we have not discussed. In this article, we have merely suggested some of the numerous ways that might be used to perform generalised colour recognition / discrimination. We have not attempted to provide a critical comparison of these techniques, nor have we tried to define criteria for fixing the control parameters (N for procedures 2 and 3). Clearly, these are possible areas for future research.

We believe that the concept of colour generalisation is an important one, and will have a significant impact upon the acceptance by industry of machines that can make decisions about objects containing several colours. The inspection of multi-coloured printed packaging is one obvious area of application for colour recognition systems. The isolation of the various components in the scene being viewed is made more reliable by the use of colour generalisation. We saw this in Figure 9. Analysis of such scenes is made much simpler if colour generalisation is used. For example:

(a) The “noise” level is reduced.

(b) Regions of constant colour are made more “solid”.

(c) Colour boundaries are better defined.

(d) Estimates of the areas of the various coloured regions are more accurate.

(e) It is possible to identify coloured regions from their shapes more reliably.

While we have explained the rationale and context for our work in terms of programs written in Prolog, there is no fundamental reason why other computer languages could not be used instead. However, the authors feel that it would be a pity to forfeit Prolog’s distinctive style of (declarative) programming, which is, we believe, of great importance for ease of programming.

It should be noted that the fast hardware colour recognition system that we has discussed that we have described can be fully simulated in software, using a conventional programming language, such as Pascal or C.

The ideas outlined in this paper are currently being studied by us, in relation to applications in agriculture, the food, printing, electronics and plastics industries, as well as for the inspection of packaging, toiletries, pharmaceuticals, etc.

Procedures for colour recognition based upon Figures 4 and 5 are worthy of further research. Clustering occurs naturally in both *HS*- and *xy-space*, in the same way as we have already seen in the colour triangle. In both cases, a broadly similar process to that described above would be appropriate for thresholding, noise reduction and colour generalisation by blob dilation, prior to back-projection into RGB-space. We hope to report on this in future publications and on their implementation in software.

Acknowledgements. This research was performed in part while BG Batchelor was supported as a Visiting Professor by the School of Electronic Engineering, Dublin City University. In addition, the authors received a travel grant from EOLAS / British Council. They are also pleased to acknowledge the contribution to their understanding of colour recognition techniques and their use in practical industrial applications, by Dr. AP Plummer and Dr. FM Waltz.

References

Barschdorff D, Bentlage D, Jeude M (1988) Farberkennung mit schneller sequentieller Klassifikation ("Color recognition with fast sequential classification", in German). Elektronik 37(13): 61-66.

Barth M, Hirayama D, Beni G, Hackwood S (1992) A color vision inspection system for integrated circuit manufacturing. IEEE Transactions on Semiconductor Manufacturing 5(4): 290-301.

Batchelor BG (1975) Practical Approach to Pattern Classification. Plenum, New York & London.

Batchelor BG (1991) Image Processing in Prolog. Springer Verlag, Berlin & New York.

Batchelor BG (1992) Colour Recognition in Prolog. Proc. SPIE conf Machine Vision Applications, Architectures and Systems Integration SPIE 1823: 294-305.

Batchelor BG, Whelan PF (1993) Generalisation Procedures for Colour Recognition. Proc. SPIE conf. on Machine Vision Applications, Architectures and Systems Integration II SPIE 2064: 36-46.

Billmeyer Jnr FW, Saltzman M (1966) Principles of Color Technology. Interscience Publishers (division of John Wiley), New York.

Blaauw EJ, de Graaf G, Wolffenbuttel RF (1993) Colour measurement system for the testing of an integrated silicon colour sensor. Proc. conf. EUROSENSORS V, Rome. Sensors and Actuators, A: Physical 32(1-3): 442-451.

Capson DW, Eng S-K (1988) Tiered-color illumination approach for machine inspection of solder joints. IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI 10(3): 387-393.

Chamberlain GJ, Chamberlain DG (1980) Colour: Its Measurement, Computation and Application. Heyden & Son Ltd., London, :18-45.

Delwiche MJ (1989) Maturity standards for processing 'Clingstone' peaches. Journal of Food Process Engineering 10(4): 269-284.

Doney TA (1991) Production quality control problems. Proc. SPIE conf. Intelligent Robots and Computer Vision IX: Algorithms and Techniques SPIE 1381: 9-20.

Foran B (1990) Glass container market in California. BioCycle 31(4): 94-95.

Gibboneym DL (1990) Closing the loop with glass recycling. BioCycle 31(4): 90-92

Gitzhofer, K-H (1991) Construction of an experimental unit for the mechanical colour sorting of non-crushed recycled container glass. Glastechnische Berichte 64(1): 9-15.

Harrell RC, Slaughter DC, Adsit PD (1989) Fruit-tracking system for robotic harvesting. Machine Vision and Applications 2(2): 69-80.

Intelligent Camera (1990), Image Inspection Ltd., Unit 7, First Quarter, Blenheim Road, Kingston, Surrey, KT19 9QN, U.K.

Jain AK (1989) Fundamentals of Digital Image Processing. Prentice Hall Int., Englewood Cliffs.

Judd DB (1952) Color in Business, Science and Industry. Wiley, New York.

Jungman D, Lozano RD (1986) Measurement of Color on foods: some experiences at INTI Buenos Aires. Proc. ASTM conf. Review and Evaluation of Appearance: Methods and Techniques. ASTM Special Technical Publication 914: 62-68

Kamel K, Elmaghraby AS (1988) Use of multiple Opticram cameras for robotics applications. Proc. conf. 1988 IEEE Southeastcon.

Kanamori K, Kawalami K, Hiroaki H (1990) Novel Color Transformation Algorithm and its Applications. Proc. SPIE conf. on Image Processing Algorithms and Techniques. SPIE 1244: 272-281.

Keesee SH, Aspland RJ (1988) Factors affecting the potential of on-line color measurement. Textile Chemist and Colorist 20(4): 15-18.

Klinker GJ (1993) A physical approach to color image understanding. A. K. Peters, Wellesley, MA.

Lambert RA, Batchelor BG (1991) Segmentation of colour images on a Transputer array. Proc. SPIE conf. Machine Vision Applications, Integration and Applications SPIE 1615: 187-193.

MacProlog (1992), Logic Programming Associates Ltd., Studio 4, Royal Vistoria Patriotic Building, Trinity Road, London, SW18 3SX, U.K. MacProlog is also sold in U.S.A. by Quintus Inc., Mountain View, CA.

MacAdam DL (1970) Sources of Color Science. M.I.T. Press, Cambridge, MA, U.S.A..

Miller BK, Delwiche MJ (1989) Color vision system for peach grading. Trans. ASAE 32(4): 1484-1490.

Mital DP, Leng Goh Wee, Khwang Teoh Eam (1990) Colour vision for industrial applications. Proc. 16th Annual Conf. of IEEE Industrial Electronics Society - IECON'90, Pacific Grove, CA, USA. In Signal Processing and System Control Factory Automation IECON Proceedings (Industrial Electronics Conference) 1: 548-551

Munsell Color Co. Munsell Book of Color. 2441 North Calvert St., Baltimore, MD.

Optical Society of America (1953) The Science of Colour. Optical Society of America Committee on Colorimetry : 99 - 144.

Optical Society of America (1978) Handbook of Optics. W. G. Driscoll & W. Vaughan (eds), McGraw-Hill, New York.

Parkkinen J, Jaaskelainen JT, Kuittinen M (1988) Spectral representation of color images. Proc. 9th International Conference on Pattern Recognition. (IEEE cat n 88CH2614-6) : 933-935.

Plummer AP (1991) Inspecting Coloured Objects Using Grey Scale Vision Systems. Proc. SPIE conf. Machine Vision Systems Integration SPIE CR36: 78-92.

Schettini R (1993) A segmentation algorithm for color images. Pattern Recognition Letters 14: 499-506.

Slaughter DC, Harrell RC (1989) Discriminating fruit for robotic harvest using color in natural outdoor scenes. Trans. ASAE 32(2): 757-763.

Stoksik M, Nguyem DT, Czernkowski M (1991) Neural net based color recognition system. Proc. 2nd International Conference on Artificial Neural Networks, IEE Conference Publication 349: 86-90.

Swain MJ, Ballard DJ (1991) Color indexing. International Journal of Computer Vision 7(1): 11-32.

Syeda T (1991) Finding distinctive colored regions in images. Conf. Intelligent Robots and Computer Vision IX: Algorithms and Techniques SPIE 1381: 574-581.

Tarbell KA, Reid JF (1991) Spatial and spectral characteristics of corn leaves collected using computer vision. Trans. ASAE 34(5): 2256-2263.

Thai CN, Shewfelt RL (1991) 'Redglobe' peach color kinetics under step-varying storage temperatures. Trans. ASAE 34(1): 212-216.

Thai CN, Shewfelt RL, Garner JC (1990) Tomato color changes under constant and variable storage temperatures. Empirical models. Trans. ASAE 33(2): 607-614.

Umbaugh SE, Moss RH, Stoecker WV (1992) Automatic color segmentation algorithm with application to identification of skin tumor borders. Computerized Medical Imaging and Graphics 16(3): 227-235. Refers to Part no. DT 2871, Data Translation Ltd.

Whelan PF (1993) Automated Packing of Arbitrary Shapes: A Systems Engineering Approach. Ph.D Thesis. University of Wales.

Xie X, Beni, G (1991) A new fuzzy clustering validity criterion and its application to color image segmentation. Proc. 1991 IEEE International Symposium on Intelligent Control. (IEEE cat n 91CH3019-7) :463-468.

Yu FT, Uang Chii-Maw, Yang Xiangyang (1993) Exemplar-based optical neural net classifier for color pattern recognition. Proc. SPIE conf. Optical Computing and Neural Networks SPIE 1812: 46-56

Zayas I, Steele JL (1991) Image analysis applications for grain science. Proc. SPIE conf. Optics in Agriculture SPIE 1379: 151-161.

Zheng J, Valavanis KP, Gauch J (1991) Object extraction for color robotic vision systems. Proc. 1991 IEEE International Symposium on Intelligent Control. (IEEE cat n 91CH3019-7) :449-456.

Zuech N (1991) Introduction to color based machine vision. Sensors 8(5): 37-39.

Appendix: Calculating the Colour Scattergram and Programming the PCF.

Consider Figure 13. The position of a point in the colour triangle can be specified by the parameters[6] U and V, which can be calculated from R, G and B using the following formulae:

$$U = (R - G) / [\sqrt{2}. (R+G +B)]$$

and

$$V = (2.B - R - G) / [\sqrt{6}. (R+G+B)]$$

To see how these equations can be derived, view the colour triangle normally (i.e. along the line QPO, the diagonal of the colour cube). When the vector (R,0,0) is projected onto the colour triangle, the resultant is a vector V_r of length $R\sqrt{(2/3)}$ parallel with the R' axis. In a similar way, when the vector (0,G,0) is projected onto the colour triangle, the result is a vector V_g of length $G\sqrt{(2/3)}$ parallel to the G' axis. Finally, the vector (0,0,B) projected into the colour triangle forms a vector V_b of length $B\sqrt{(2/3)}$ parallel to the B' axis. A given colour observation (R,G,B) can therefore be represented by the vector sum $(V_r+V_g+V_b)$. Finally, U and V can be calculated, simply by resolving V_r, V_g and V_b along these axes.

The mapping function $\Gamma(.)$ mentioned in the text is given by:

$$\Gamma((R,G,B)) = ((R - G)/(\sqrt{2}.(R+G +B)), (2.B - R - G)/(\sqrt{6}.(R+G+B)))$$

---

[6] These parameters are not to be confused with the CIE Uniform Chromacity Scale (UCS)-system.

Clearly, there are many values of the colour vector (R,G,B) which give identical values for $\Gamma((R,G,B))$. The set of points within the colour cube which give a constant value for $\Gamma((R,G,B))$ all lie along a straight line passing through the origin in RGB space (O). Let us denote this set by $\Phi(U,V)$, where

$$\forall X: X \in \Phi(U,V) \rightarrow \Gamma(X) = \{U,V\}$$

The colour scattergram is simply an image in which the "intensity" at a point {U,V} is given by the number of members in the set $\Phi(U,V)$.

The details of the process of programming the PCF are as follows:

1. We assume that the colour scattergram has been processed, yielding an image in which there are several blob-like regions of different intensities. Let the intensity at a point {U,V} in the colour triangle be equal to I(U,V).
2. The entry in the $(R,G,B)^{th}$ cell in the LUT is equal to $I(\Gamma((R,G,B))$.
3. Step 2 is repeated for all points in the colour cube.

Notice that all entries in the LUT corresponding to members of the set $\Phi(U,V)$ are given the value I(U,V)

Legends for the Diagrams

Figure 1. Margarine tub referred to in the text (simulation).

Figure 2. The standard CIE (1931) Chromaticity Diagram. The horse-shoe shaped curve is known as the *spectrum locus* and represents the set of points on which all pure spectral colours lie. (Chamberlain and Chamberlain 1980) The coordinates x, y are calculated in terms of the tristimulus variables (X, Y, Z), the significance of these values is not our concern here. Let it suffice to say that (X, Y, Z) can, in turn, be related, albeit approximately, to physically measurable quantities, such as the (R, G, B) variables measured by a video camera. In this composite diagram, three different forms of the Chromaticity Diagram are merged.

(i) The Chromaticity Diagram is sometimes printed in colour, having been tinted by an artist. (Optical Society of America 1978) To avoid using expensive colour reproduction techniques, some authors / publishers have, however, chosen to draw the diagram with regions labelled with symbolic colour names, such as *"bluish green"*, *"blue green"* and *"greenish blue"*. (Judd 1952) However, people frequently differ in their naming of the same coloured surface. Confusion of nomenclature often occurs, for example, on the *blue-turquoise-green* continuum. It is well known that culture, language, age, illness and chemicals (e.g. barbiturates and alcohol) can all cause changes in human colour perception. For this reason, boundaries of the labelled "cells" are not fixed; they differ from one observer to another. Furthermore, the diagram does not indicate the location of colours such as <margarine-tub yellow>, <3M red>, <IBM blue>, or even <sky blue>. The objective of this article is to define computational techniques which allow the boundaries of regions associated with colour names to be learned from a finite and small set of samples.

(ii) Some authors have indicated the relationship between certain well-known colours and points on the Chromaticity Diagram. See for example (Chamberlain and Chamberlain 1980). *Key*: *Lt* - lettuce; *LM* - lemon; *OR* - orange. This is a gross simplification. It is obvious, for example, that an individual tomato can display a wide variation of colours. Additional variations exist among different samples of the same variety and different species of tomato, due to the degree of ripeness, growing conditions etc.. For these reasons, a broad blob-like region is a more accurate representation of colours such as <tomato>, <lemon> and <lettuce>. Smaller but nevertheless significant differences exist in the colours of printed packaging, plastics parts, etc. (Chamberlain and Chamberlain 1980; Billmeyer Jnr and Saltzman 1966).

(iii) MacAdam (1970) shows another version of the Chromaticity Diagram, which contains a series of small ellipses, each of which shows *just noticeable colour differences*. Any colour represented by a point lying just outside an ellipse is perceived by a human observer as being just noticeably different from the colour denoted by the centre of the ellipse. (Also see Jain (1989)) The three black ellipses, labelled A, B and C are enlarged here for clarity. Notice that they differ in size and orientation. For our purposes, machine rather human perception of colour differences is important.

Figure 3. Hardware structure of the Programmable Colour Filter, based upon RGB inputs to the LUT. This is the block diagram of the implementation of this technique in the Intelligent Camera (1990; Plummer 1991) and was used by the authors in the experiments reported here. The pseudo-colour display is used only for the convenience of the user; it does not form part of the processing chain.

Figure 4. Proposed hardware structure of a Programmable Colour Filter, based upon HSI inputs to the LUT, using a real-time RGB/HSI converter chip (Umbaugh et al 1992).

Figure 5. Proposed hardware structure of a Programmable Colour Filter, based on the *xy* parameters used in defining the standard Chromaticity Diagram.

Figure 6. The training, recognition and generalisation processes

(a) Flowchart for the training and recognition procedures. The line labelled "Additional Inputs" indicates that the procedure represented by the top four blocks can be repeated many times and the results merged.

(b) The colour scattergram consists of a number of diffuse clusters.

(c) After thresholding, the colour triangle contains a number of small distinct blob-like regions.

(d) Simple generalisation the blobs become enlarged.

(e) More complex generalisation procedures allows these regions to be expanded until they touch. The back-projection process outlined in Figure 7(b) is now applied to (d) or (e).

Figure 7. Transformations in RGB space.

(a) Calculating the colour scattergram. The diagram shows the colour cube, defined as a cube-shaped region of RGB space, and the colour triangle. X is a general point lying within the colour cube and Y is the projection of X onto the colour triangle. Hence, OX (or OX produced) intersects the colour triangle at Y. The point Z is located at the intersection of OX-produced with the edge of the colour cube. All points along OZ have the same values of hue and saturation but different values of intensity. To calculate the value at point Y in the colour scattergram, we simply count the number of pixels along OZ.

(b) Programming the colour filter. Calculating the entries in the look-up table is a process of projecting values stored in the colour triangle through the colour cube. Blob A is a region within the colour triangle enclosing a set of points that all have the same colour label, L. L is a scalar quantity, and hence can be regarded as an intensity value in the image containing the colour triangle. Blob A also defines a conical region within the colour cube (light stippling). Blob C is the intersection of that cone with the edge of the colour cube. All (R,G,B)-vectors which fall within this cone can be associated with the colour label L. Thus, all entries in the PCF look-up-table which correspond to points within this cone are assigned to value L. Blob B which has a different colour label is treated in the same way as blob A.

Figure 8. Colour generalization applied to samples of fruit. Also see Table 1.

(a) Colour scattergram after noise removal. The white blob to the left corresponds to a Golden Delicious apple. The central blob corresponds to an orange. The right-most blob corresponds to the red region of a Star Crimson apple.

(b) The application of procedure 1 to the image in (a). N = 5.

(c) The application of procedure 2 to the image in (a). N = 6.

(d) The application of procedure 3 to the image in (a).

Figure 9. Learning to distinguish coloured stripes, with varying degrees of generalisation. The stripes are (from top to bottom): yellow, lime green, turquoise green, sky blue, purple and cerise. Simple fixed-value thresholding and a noise-reduction filter were applied to the colour scatttergram obtained from the stripe image.. This generated a set of six blobs in the colour triangle. (The small white spots in (a).) The blobs were then enlarged and the contents of the look-up table were

calculated in the usual way, by back-projection. The resulting PCF was then reapplied to the stripe image.

(a) Blobs in the colour triangle, at various stages of enlargement. The small white spots show the blobs with no enlargement. For the sake of illustration, these spots were deliberately made to be small, by selecting a high value for the threshold parameter. The larger, darker blobs, were obtained by applying procedure 2. (N = 5) The white lines indicate the watershed boundaries generated by Procedure 3.

(b) Result of applying the PCF designed using the small white blobs in (a). (i.e. with no generalisation) These blobs were shaded before back-projection was applied, enabling the PCF to distinguish the different colours in the stripe pattern. Notice that the PCF does not recognise all of the points seen during training and that the output image is quite noisy.

(c) Image from the PCF generated by applying back-projection to the enlarged dark spots in (a). Procedure 1 gives identical results in this case. Notice the improved recognition performance but that it is still inferior to that shown in (c).

(d) Image from the PCF generated by procedure 3. Notice how the ability to distinguish these six colours has been improved.

Figure 10. Using generalisation in the training of a PCF to recognize {Golden Delicious} (normally described as yellow-green).

(a) Applying the PCF to {rainbow}. Notice that only a very narrow range of colours is recognized.

(b) Applying the PCF generated by procedure 1 (N=5), to {rainbow}. A broader range of colours is recognized than in (a).

(c) Applying the PCF generated by procedure 1 (N=10), to {rainbow}. Notice that the range of colours recognized by the PCF has increased yet again.

Figure 11. Generalising from specific instances of {green}. The black blobs correspond to three different shades of {green}, taken from covers of magazines. It is reasonable to expect that points within the convex hull of the black blobs will also correspond to {green}. The white blob corresponds to a fourth shade of {green} which was used to test this hypothesis. (See text.) Notice that a little over half of the white blob is enclosed within the convex hull. This corresponds well with the observed recognition rate of 63% after generalisation using procedure 4.

Figure 12. Procedures for designing and using a PCF.

Figure 13. Showing the relationship between the RGB- and UV-coordinate axes. The vectors **R'**, **G'** and **B'** all lie in the UV-plane, which also contains the colour triangle.

Table 1. Recognition performance of a PCF, with various generalisation procedures. For example, a PCF was designed, without the use of generalisation, to recognise {Golden Delicious}. When the PCF was subsequently applied to the same sample, it achieved a recognition accuracy of only 82%. However, when this process was repeated with any of the three generalisation procedures, the recognition rate was over 99.9%.

Generalisation Procedure	**Orange**	**Golden Delicious**	**Star Crimson (red area)**	**Star Crimson (green area)**
None	77	82	45	2
Procedure 1	> 99.9	> 99.9	97	97
Procedure 2	> 99.9	> 99.9	99.9	> 99.9
Procedure 3	> 99.9	> 99.9	> 99.9	> 99.9

Table 2. Equivalent entities and operations in Colour Space, Colour Triangle & Prolog+.

Colour Space		**Colour Triangle**	**Prolog+**	**Remarks**
Human	**RGB**	**(Image)**		
<yellow>	{yellow}	Single blob	**yellow**	No universally agreed definition.
<tomato red>	{tomato red}	Single blob	**tomato_red**	
<Q>	Approx. to {Q}	Union of discs	Prolog list	List format: [[X1,Y1,Z1], … [Xi,Yi,Zi], …]
<A> ∪ <B>	{A} ∪ {B}	Blob merge	**max**	Union (OR)
<A> ∩ <B>	{A} ∩ {B}	Blob intersection	**min**	Intersection (AND)
Colour generalisation		Blob expand	**exw**	Procedure 1
Colour generalisation		Blob expand	Command sequence	Procedure 2
Colour generalisation		Blob expand	[**neg, mdl**]	Procedure 3
Colour generalisation		Blob expand	**chu**	Procedure 4
Colour generalisation		Blob expand	Simple list manipulation	Procedure 5 Procedure 6