Colour Analysis and The Classification of Fruit

By Gary R. Kay

Submitted to the University of Cape Town in fulfilment of the requirements for the degree of Master of Science in Electrical and Electronic Engineering

Cape Town, May 1992

Declaration

I declare that this dissertation is my own unaided work. It is being submitted for the degree of Master of Science in Engineering at the University of Cape Town. It has not been submitted before for any degree or examination at this or any other university.

Signature of candidate: signature removed

G. R. Kay

Dated this 26 day of MAY, 1992.

Acknowledgements

I would like to thank the following for their assistance with this project:

My supervisor, Professor Gerhard de Jager, for his guidance and unfailing enthusiasm throughout this project;

Mr. Bernard van Zyl, Jopie Ten Cate and others at Teklogic, for their equipment and contributions to this project;

Dr. E.H. Einhorn for his help, and for introducing me to the colour analysis department at Woolworths, Cape Town. My thanks also go to Miss Carol Johnson for allowing me to use the ICS Texicon colour analyzer;

Paul Symonds, Greg Cox, Wayne Borchardt, Fred Hoare, Robert Crida, and Peter Moon for their knowledgable contributions to this project;

Natalie Kay and Debbie Montlake for their support and help.

Terms of Reference

This thesis was commissioned by Professor G. de Jager (Post Office Professor of Telecommunications and Signal Processing, U.C.T) and Mr. B. van Zyl (Manager of Teklogic (Cape)) on February 12 1991.

Teklogic (Cape) is a branch office of Teklogic in Johannesburg, and is a division of Altech Electronic Systems. The agricultural industry has the problem that all fruit to be exported must be of a high standard. Any box of fruit should contain one cultivar in which all the fruit appear to be all the same size, weight, and colour. The manual sorting of fruit has not met up to the expectations of the export market. Importing automated fruit sorting devices cost over a quarter million rands. Generally these devices are insufficiently accurate or slow, and are difficult to maintain. At present, Teklogic is involved in the design and production of an automated fruit inspection system that will overcome the problems experienced with previous sorting methods. For this reason, Teklogic approached the University of Cape Town to assist them in the design of the colour inspection module for the automated sorting system. The project for this thesis was to provide solutions for the colour module.

Mr. van Zyl's initial requirements for this project were to:

1) Do a literature search of related technology and possible solutions to similar problems to formulate alternate ways of categorising colour.
2) Define the specific colour sorting requirements baseline and formulate the colour sorting problem with reference to critical problem areas and identify focus areas.
3) Formulate the advantages and disadvantages of processing colour in RGB or other colour spaces.
4) Conceptualize the solution; inclusive of a proposal for implementation of neural network technology to benefit the solution.
5) Compile a list of algorithms which could be used in the analysis of colour parameters and colour segmentation.
6) Calculate the time budget available for processing.
7) Define the type of hardware that will be required to implement the solution.
8) Formulate the solution of practical implementation with hardware and software to overcome identified problems, and satisfying the requirements for colour sorting.

Synopsis

The increasing high standards of fruit quality expected by the agricultural export market of South Africa has reached a stage that fruit must be accurately graded in a short a time as possible. This thesis describes colour systems and methods to grade the fruit automatically via the clustering and classification methods. After investigating several approaches to automatically sort fruit based on colour, an image processing approach was taken. The colours on the fruit (specifically apples) were analyzed, by capturing a colour image of the fruit and analyzing the pixels in the image.

Several colour representation systems were investigated and they are: colours represented by spectral power distributions and spectral reflectance curves; the CIE 1931 XYZ tristimulus values; the CIE 1931 x,y,z chromaticity coordinates; the CIE 1960 L,u,v uniform chromaticity scale (UCS); the Munsell colour wheel of hue, value and chroma (HVC); the L*u*v* system; the L*a*b* system; the Red, Green and Blue (RGB) system; and the hue, saturation and intensity (HSI) perceptual colour representations. In addition, several clustering and classification techniques were investigated and they are: the supervised methods of Parametric Bayesian classification and minimum Euclidean distance classification; and the unsupervised methods of the K-means algorithm and the ISODATA classification approach.

The ICS Texicon computer spectrophotometer (ICS Texicon Spectraflash Manual (1991)) was used to check the performance of most of the colour systems described by analyzing apple sample colours. It was observed that all the colour systems described are capable of adequately representing the colours on a fruit. However, the L*u*v*, L*a*b*, and HSI were among the better colour systems investigated. The HSI colour system was chosen for further investigation since colours could be perceptually understood and adequately represented in this space, also transformation to this colour space proved to be fast.

An HSI system was developed based on several existing transformations from RGB. It was found that one method was sufficient to evaluate hue (H), two methods could be used to represent saturation (S), and two methods could be used to evaluate intensity (I). Methods of using these features to represent the colours on apple images revealed that the H feature combined with the I feature (calculated as an average of RGB) was one of the best feature

combinations to use. Clustering and classification results using unsupervised and supervised methods, confirmed that the H-I feature combination was ideal in, most cases, for accurate colour segmentation and hence grading of the fruit images.

The main conclusion that can be made is that if lighting conditions allow for a uniformly illuminated fruit sample then the H-I feature combination, together with previously set up colour classes, will allow for a fast and accurate classification of fruit.

It is therefore recommended that the automated fruit inspection system implements a hardware transformation to the H and I features. The colour classes for classification should be provided as a selection of colour palettes in terms of the H-I feature combination. For each cultivar run there should be specific palettes to represent the colour class into which the fruit should be graded.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

> *" Colour is attractive and interesting to everyone. Consequently, control of colour is important to all producers, buyers, sellers, and users of coloured materials. In various ways, colour is an indication of freshness, quality, or other desirable (or undesirable) characteristics of goods. To assure acceptability, saleability, and favourable price - especially in contracts and monitoring of conformance to specifications - numerical expression of colour is greatly superior to verbal descriptions. Disagreements concerning words or visual comparisons with samples are all too likely and frequent. Such disagreements underlie much unpleasantness and loss in commerce in consumer goods. Such loss of money and goodwill must amount to billions of dollars per year, world wide.*
>
> *" Persistent efforts to substitute measurements of colour for visual judgement have marked the twentieth century. Because visual perception of small colour differences is so acute, the requirements for accuracy and world-wide reproducibility of colour measurements have been severe. Only during the last half century have practical spectrophotometers with adequate accuracy been available."* (MacAdam (1981), preface).

In order to understand the measurement of colour, an introduction to colour vision and to fundamental colour reproduction with respect to human vision and machine vision must first be given.

Human Vision

The human eye contains receptors that collect light from surfaces that absorb illuminant light and reflect the unabsorbed light. In the eye the *retina* contains the *rods* and *cones* which are the visually sensitive elements. The cones are concentrated at the *fovea*, this region provides the most acute vision allowing one to perceive the finest detail. The cones are responsible for what is known as *photopic* vision, that is they allow one to perceive colour, and they require a relatively high level of stimulus

before they can become operative.

The rods are more sensitive than cones, and are spread throughout the retina except for the foveal region. Rods are used in *scotopic* vision - which cannot discriminate fine detail since groups of rods have a common connection to the optic nerve. Scotopic vision may also be termed *dark-adapted* vision, and it has no colour sensation associated with it. The spectral sensitivity of scotopic vision is represented by the dashed curve in Figure 1.1, while the solid curve - also known as the V(λ) curve refers to conditions of photopic vision (Einhorn (1990a)).

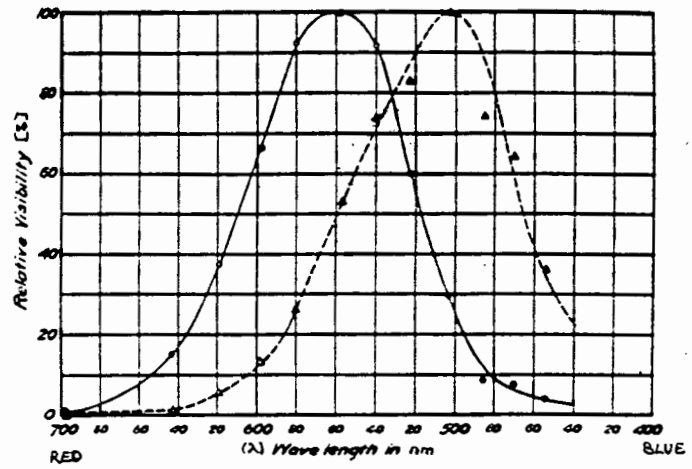


Figure 1.1: The spectral sensitivity of the eye. Dashed curve is scotopic (rod) vision, the solid (V(λ)) curve is photopic (cone) vision.

The characteristics of colour vision differ even among persons with normal colour vision. If different laboratories or countries used different visual data for interpretation of the significance of measurements obtained from spectrophotometers, the need for interchangeably reproducible colour specifications would be subverted. To forestall that danger, the International Commission on Illumination (CIE) recommended data that characterize a standard observer for colorimetry. Related to those data, a coordinate system for maplike representation of the results of colour measurements was recommended at the same time (1931) by the CIE. Methods for using the CIE data and coordinate system constitute the subject of colorimetry.

Machine Colour Vision

Machine colour vision in this context refers to instruments that measure colour, and interpret the colour as a certain value or vector of values. The colour is generally interpreted as a set of *tristimulus* values X, Y and Z which are measured from the reflectance data of an illuminated surface, also observer data may be used if the values are to represent human observation at specific field widths. The CIE (1931) standard colorimetric system is based on 2^0 purely foveal (cones) observations. A CIE (1964) supplementary colorimetric system, based on observations on a 10^0 field, differs somewhat from the former which is most commonly used.

(Wyszecki and Stiles (1967) p270)

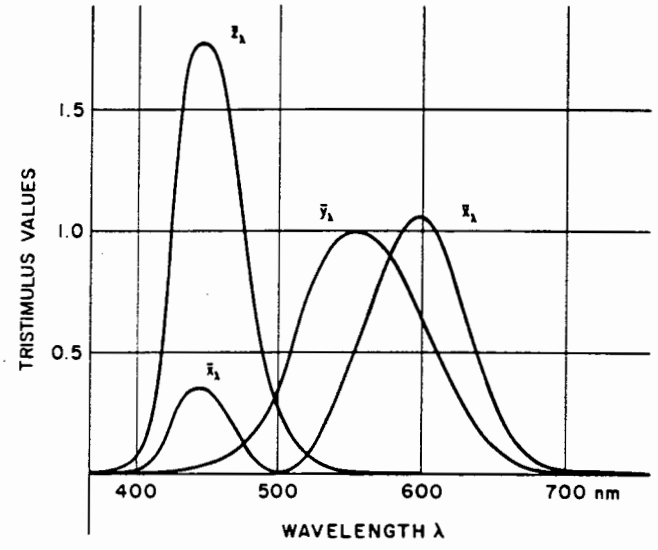


Figure 1.2: Spectral tristimulus values per watt of indicated wavelengths, for CIE 1931 standard observer. These values for the spectrum are the CIE 1931 colour-matching data

Tristimulus values refer to a weighting of three primary colours (e.g. red, green and blue) in such a way that any colour can be matched to some combination of these primaries. The tristimulus values that were adopted by the CIE for various spectrum colours are represented graphically in Figure 1.2. The curves of $\bar{x}(\lambda)$, $\bar{y}(\lambda)$, $\bar{z}(\lambda)$ in Figure 1.2 indicate the amount of each of the CIE primaries that is required to match the colour of one watt of radiant power of the indicated wavelengths. (MacAdam (1981) Ch1).

Although the choice of primaries is essentially immaterial, the interpretation of CIE tristimulus values gives one some idea of the characters of those primaries. The value X (which is conventionally designated $\bar{x}(\lambda)$ in the case of the spectrum colours) represents the amount of a reddish primary that has higher saturation than any obtainable red. The value of Y (or $\bar{y}(\lambda)$ for a spectrum colour) represents the amount of a green primary that is considerably more saturated but of the same hue as the spectrum colour whose wavelength is 520 nm. The value Z (or $\bar{z}(\lambda)$) represents the amount of a blue primary that is considerably more saturated than, but of the same hue as, the spectrum colour whose wavelength is 477 nm.

The mechanism briefly described above for colour measurement enables one to measure the colour of any sample and to relate this colour measurement to what would normally be seen by a human observer. There have, however, been several drawbacks of the CIE (1931) colour standard with respect to some industrial colour representation applications. A multitude of colour representation systems have since been developed, each having its own advantages for being specific to certain colour representation applications. With all the different colour languages that now exist it is difficult to choose one single colour system for a specific colour application.

There are two major criteria for industrial colour inspection systems, namely speed and accuracy. For the project about to be described in this thesis, one of the main objectives was to find such a colour system for the real-time accurate classification of fruit on an automated inspection system. The fruit to be graded is aimed specifically for the high standards of the South African export market. Existing colour inspections systems produced overseas have several drawbacks which promoted the design of a new South African designed colour inspection system. The drawbacks of existing devices are that: if the system was accurate it was also slow (about one fruit per second); if the system was fast (real-time classification) then a large percentage of products inspected were classed inaccurately, since the system would only give an average value of the objects colour; and most systems were expensive to import and maintain, costing over a quarter million rand (van Zyl (1991)).

When designing the colour inspection system the following must be accounted for:

- For accurate colour analysis the system must firstly store an image of the object. This image must then be processed by segmenting the colours in the scene. This data must then be compared to model samples, to hence classify the colour of the object.
- The system should be able to account for non-uniformity of a colour distributed on the objects surface and background.
- The system must allow for rapid colour identification, and sorting of the object into its appropriate grade (classification within 100ms is required). In terms of fruit sorting, the system must be able to distinguish between fruits of the same cultivar.
- Optimum lighting of the object being analyzed is required. i.e. the problem of gloss and the types of lamps to be used should be considered.
- Colours which may be just distinguishable by the human eye may not be distinguished by some colour measurement methods, hence accurate colour measurement is required by the system.
- The system must be able to function independent of surrounding factory disturbances, such as dust, electric fields, etc.
- The system must be designed as cost effectively as possible.

The objectives of this thesis are:

- To describe various methods of representing colour, and provide examples of where related existing technology categorises colour for industrial applications.
- To determine from experimental results, the advantages and disadvantages of perceptual colour spaces over other colour representation systems, and hence find a colour system that can adequately represent the colours in a single fruit.
- To provide the theory behind several colour systems so that their fundamental transformation equations could be implemented in hardware, and so that a formal comparison can be made between the different colour systems.
- To determine from experimentation the best colour sorting features of the perceptual HSI colour space with the intention of providing a fast and accurate colour representation of the objects being inspected.

- To provide several theoretical solutions to the colour inspection problem, by using clustering and classification theory.
- To determine from experimentation, the best colour parameters and colour segmentation techniques needed for the on-line colour inspection of specific fruit cultivars.
- To provide sufficient theory, algorithms and a software simulation of the eventual hardware system so that final hardware implementation can be based on these solutions.
- To make recommendations for the eventual colour inspection system and state alternative solutions.

The colour systems described in this thesis are limited to those described by material researched by the author in 1991. It was discovered recently, however, that there is a colour system which will apparently be made an international industrial standard for textiles and other industries associated with colour matching. The system is the CMC standard which has recently been developed in the U.K by the Colour Measurement Committee of the Society of Dyes and Colourists (Burke (1992)). This thesis does not provide indications of the time it takes to process algorithms or calculate colour segmentation data for the eventual hardware system. Instead rough estimates of processing times are given by software used on a 16MHz 386 computer with no co-processor.

In this thesis chapter 2 will introduce the colour systems considered for the colour inspection system. These colour systems range from spectral reflectance representation, to general colour spaces involving the CIE tristimulus values, to perceptual colour spaces which involve the hue, saturation and intensity components. Chapter 2 will also provide examples of where the colour spaces have been applied in existing colour technology.

Chapter 3 will then use most of the colour systems described in chapter 2, in an experimental comparison between the colour systems. The experimentation involves the use of the ICS Texicon computer spectrophotometer colour analyzer (property of Woolworths, Cape Town), and results are compared for apple samples observed under two types of lighting (CWF and TL84) and at two observation field angles (2^0 and 10^0).

In chapter 4 reasons are given as to why the HSI colour system was chosen for subsequent experimentation. This chapter then provides further theoretical insight into the many ways of transforming from the RGB to the HSI colour space. This chapter ends with experimental results. These results show the many features and combinations of features that can be used to represent the colours in a colour image.

Based on a selection of the fewest features that best represent the colours in a sample image, chapter 5 aims to describe several clustering and classification theories. The theories include both supervised and unsupervised clustering methods, and examples are given to illustrate how these theories can result in the colour classification of fruit.

In chapter 6 experimental results are shown. These results aim to show that certain classification techniques work well on some colour feature representations, and not so well on other colour features. This chapter, in presenting the experimental results, inherently presents the final results of this thesis.

The final chapter draws conclusions from the previous chapters and makes recommendations for the final automated colour inspection system.

CHAPTER 2

Colour Systems and Representation Techniques

2.1. Introduction

Colour representation has, until recently, been a psychological interpretation to human observers. Colour evaluation has been transformed from an art into a science in the past 25 years (Galloway (1989)). Colours were generally described in words. Since there has been a need to accurately describe a colour in terms of a value, several colour systems have been developed. There is not one system which has been made a rigid universal standard.

There are predominantly two major methods of representing a colour or set of colours using numerical values. Each of these representations have further variations which result in a multitude of different colour systems being available. According to Niblack (Niblack (1986) p32) more than 20 may be found in the literature on colour and colorimetry.

The first method of colour representation uses spectral reflectance, or power distribution, with associated wavelengths (reflectance is the ratio of amount of light reflected divided by the amount incident). A spectrophotometer is the instrument used to colour match sample categories by measuring reflectances of the visible spectrum wavelengths. The second method is by way of combining controlled amounts of primary colour sources. The primaries consist of at least red, green and blue. By calibrating the controls, the amount of each primary can be recorded. The unknown colour can then be specified by those amounts. These are known as the *tristimulus values*, where each number represents the amount of one of the primary stimuli. A colorimeter is the instrument used to colour match the tristimulus values of the samples by using filters.

A combination between the two above mentioned colour measuring systems, results in a multifilter colorimeter. This instrument contains a set of narrow band filters, which allows one to measure the whole spectrum. This spectrum is then used to calculate the appropriate

colour coordinates. Multifilter colorimeters facilitate the analysis of the colour spectrum in greater detail than with three filter devices, but less detail than with spectrophotometers.

In this chapter, section 2.2 serves as a general introduction to colour spaces. These include the standard CIE XYZ tristimulus coordinates, the *x, y* chromaticity coordinates, and general perceptual colour spaces. Before any detail is given on these colour spaces, section 2.3 is used to explain how colour is represented using spectral reflectance and spectral power distribution curves. Section 2.4 then explains colour representation in terms of equivalent stimuli. This is a more detailed analysis of the colour spaces introduced in section 2.2. At the end of sections 2.3 and 2.4 there are examples of where the colour representation systems discussed, are used.

2.2. General Colour Spaces

A colour space is a coordinate system designed to allow colours to be measured and quantitatively specified. From tristimulus colour theory, a three dimensional space is necessary, but various choices for the three coordinates are possible. Three general forms of colour space are:

1) Tristimulus Coordinates

A rectangular space in which the three coordinates, called tristimulus values, give the amount of each of three fixed primaries. The recommended set of primaries are the CIE 1931 XYZ primaries, but other sets may be used. It is customary to set the scale factor so that (1,1,1) gives a reference white.

2) Chromaticity Coordinates

The (x,y) chromaticity coordinates are derived from the tristimulus coordinates (X,Y,Z). To fully specify a colour, its luminance Y must be specified in addition to its chromaticity.

3) Perceptual Colour Spaces

These are colour spaces based on perceptual parameters such as hue, purity, brilliance, brightness, and saturation. Many different perceptual colour spaces have been defined, and a representative form is shown in Figure 2.1.

Some chromaticity and perceptual colour spaces are defined so that the perceptual difference between two colours is given (approximately) by the Euclidean distance between the colours. In this case, the spaces are called uniform colour spaces. Within these three general types of colour spaces, many specific spaces may be defined. Those investigated are given after the following section.

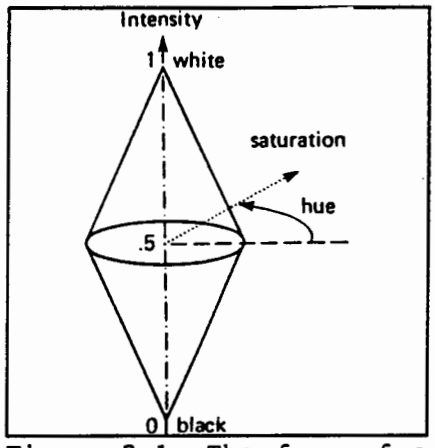


Figure 2.1: The form of a typical perceptual colour space.

2.3. Colour Represented by Spectral Power Distributions and Spectral Reflectance

In this section the spectrophotometric specification of colour is explained by way of an example. Consider an object coated with a paint that would commonly be called green. Let this sample be illuminated by a suitable source of light. A prism is placed so as to disperse the light that falls on the sample into its spectral components - violet, blue, green, yellow, orange, and red. Consider only a single component - the violet for example. It is evident that a surface cannot reflect more violet light than falls upon it.

An inherent property of coloured surfaces is that they reflect, in different proportions, components of the spectrum. A spectrophotometer measuring the reflectance on a region of the green object may give the following reflectances (from MacAdam (1981) p3):

Spectral Region	Reflectance	Wavelength Ranges
Violet	0.11	400 - 450 nm
Blue	0.28	450 - 490 nm
Green	0.33	490 - 560 nm
Yellow	0.17	560 - 590 nm
Orange	0.12	590 - 630 nm
Red	0.06	630 - 700 nm

This subdivision of the visible spectrum into six broad regions is arbitrary, since the colour of the spectrum varies without abrupt change with wavelength. The reflectance measurements for six spectral regions, although useful as an illustration of the principle underlying spectrophotometric analysis, does not define the colour of a reflecting surface with sufficient precision.

The method of colour representation using statistical pattern recognition (by Parkkinen and Jaaskelainen (1987)), is based on reflectance measurements at intervals in the spectrum. Colour recognition is obtained by a statistical pattern recognition method, called the subspace method. In this article it is shown that colour spectra can be accurately reconstructed using a few principle spectra. It also shows that this method is capable of discriminating samples which are inseparable using chromaticity coordinate (x,y) matching (i.e. filter matching). In addition the system can be as fast as conventional three parameter systems. This is because of easy optical implementation of the algorithm using optical information processing principles.

Statistical pattern recognition uses the principles of spectrophotometry for colour representation, but is as fast as three-filter colorimetry.

In general, the more wavelengths at which the reflectance is measured at, the greater the colour recognition precision. However, more time is needed for such a process. Sufficient

reflectance measurements results in a spectrophotometric curve (see Figure 2.2).

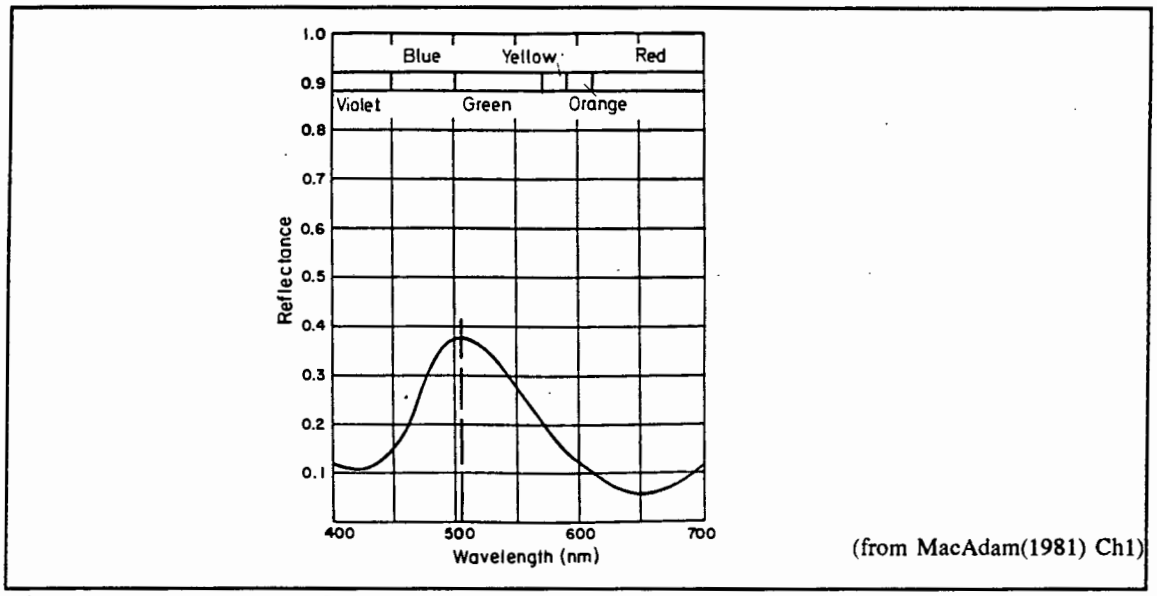


Figure 2.2: Spectral reflectance curve of a typical green paint. The dashed vertical line indicates the dominant wavelength. In this case, the dominant wavelength is 506 nm, which is bluish green.

The curve accurately defines the property of the sample that was roughly described by the data in the preceding tabulation. Every possible colour can be represented on a chart of this type. For example, a perfectly white surface, which reflects completely all of the visible radiation that falls upon it, would be represented by a horizontal line at the top of the chart. Similarly, an absolutely black surface would be represented by a horizontal line at the bottom.

It is possible to isolate a unique signature of the colour surface (Ho, Funt and Drew (1990)), by separating the colour signal into its illumination and surface reflectance components. The authors present a 'separation' algorithm for achieving colour constancy and theorems concerning its accuracy. Colour constancy means the recovery of perceived surface colour from the strengths of three receptor values representing cone responses of the visual system (or RGB responses of a colour camera) independent of the light illuminating the object. Humans have a property or behaviour of colour vision called chromatic adaption. This adaption allows one to experience colour constancy. For example, blue objects illuminated with daylight or any other usual quality of illumination (such as fluorescent or incandescent light) always appear blue, yellow objects yellow, white objects white, and other objects almost appear to have the same hues as they do in daylight. If one could achieve colour

constancy in an automated colour recognition system, the effects of gloss (illuminant reflectance) can be removed. One method to extract the spectral power distribution of an object, so as to have a colour signature independent of the illuminating source, is described as follows:

The relative spectral distribution of power of the illuminating source must be measured. This is done by using a prism to disperse the light into a spectrum and isolating each spectral region in turn. The amount of energy present in each region can thus give a spectral power distribution of the illuminant. Instruments for such measurements are called *spectroradiometers*.

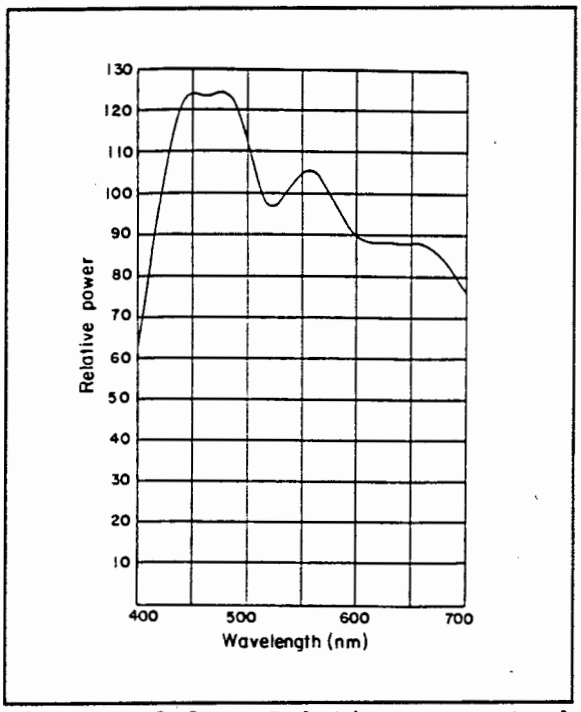


Figure 2.3: Relative spectral distribution of the power (energy radiated per unit time) from CIE Illuminant C.

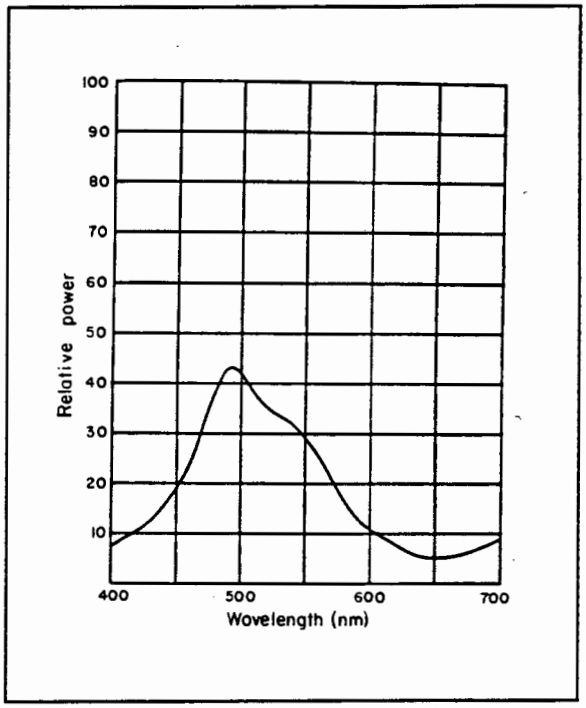


Figure 2.4: Spectral distribution of power in the light reflected from the green paint of Fig. 2.2 when the illuminant has the spectral power distribution of Fig. 2.3 (Illuminant C)

A filter which, when used with a tungsten lamp, provides a source that is a close approximation to average daylight. The spectral distribution of energy in this source is shown by the curve in Figure 2.3. At the meeting of the International Commission on illumination (Commission Internationale de l'Eclairage, CIE) in 1931, the representatives of the various countries adopted a source that has this distribution of

energy as an international standard of illumination to be used for the purposes of colorimetry except when special conditions dictate the use of other sources. This standard is known as CIE illuminant C.

If light that has the spectral quality of illuminant C falls on the surface of green paint whose spectral reflectances are represented by the curve in Figure 2.2, the spectral distribution of the energy reflected into the eye of an observer is obtained by multiplying , at each wavelength, the value shown in Figure 2.3. by the corresponding value in Figure 2.2. The result is shown in Figure 2.4.

Because the curve in Figure 2.3 is relatively flat, the shape of the new curve differs only slightly from that of Figure 2.2. This example illustrates the principle that multiplying at each wavelength the incident energy by the reflectance of the surface gives the distribution of energy in the light reflected by the surface.

The advantages of spectrophotometry and spectroradiometry are as follows:
The spectral reflectance curve of a material constitutes a permanent record that does not require preservation of a sample colour. Furthermore, the units in which the curve is expressed are universally understood and accepted (i.e. nanometres for wavelength). In addition, the curve may contain more information than can be obtained by visual examination.

Applications of Spectral Information

The analysis of colour images (Wandell (1987)) uses the advantage that spectral curves contain more information than can be obtained by visual examination. These images are obtained by using a computer based frame grabber with a colour camera. An overview is given below of the work by Wandell (1987) and Healey (1989) which describes applications of the spectral components of colour images.

Wandell's work describes the colour analysis of images based on two principles. First, image data are represented with respect to the separate physical factors, surface reflectance and

spectral power distribution of the ambient light, that give rise to the colour of the object.

Second, the encoding is made efficient by using a basis expansion for the surface spectral reflectance and spectral power distribution of ambient light that takes advantage of the high degree of correlation across the visible wavelengths normally found in such functions.

Within this framework, the same basic methods can be used to analyze image data into estimates of the spectral power distribution and surface spectral reflectances, which gave rise to the colour signal. This method is useful for the identification of material surface spectral reflectance when the lighting cannot be completely controlled. That is, ambient light (e.g. gloss) is accounted for by analyzing the image sequentially by dividing it into overlapping spatial areas.

Wandell concludes that the ability to use object surface reflectance rather than grey-level image intensity in which surface and light information are not clearly distinguishable, becomes an important aid in the development of visual inspection systems that are robust with respect to ambient lighting. However, the time to process the image for classification, may be slow.

Healey (1989) developed a colour metric that can be applied to images sensed using red, green and blue (RGB) filters. Healey transforms the sensor measurements into the spectral power distribution $(I(x,y,\lambda))$ of the light entering the camera (x and y are coordinates on the image plane and λ is wavelength). This is to account for the infinite number of combinations of filters and cameras that might be used to obtain RGB values. Since (for example) different 'red' filters placed in front of different CCD cameras will give different R values for the same incident light.

The metric thus accounts for the spectral properties of the camera and filters and their noise characteristics . The metric also is insensitive to geometric variations in the scene.

Colours can be compared using the colour distance function derived, which is an estimate of the distance between normalized spectral power distributions. Components of this distance are

weighted to account for sensor noise properties. An example is given which shows that using the metric, distances between colour patches on an image of a chart of 24 matte patches, can be used to compare colours on an image.

The metric is useful for detecting colour edges , classifying intensity edges, and estimating colour variations within an image region. The problem with this system is that it is computationally expensive .

In general, spectrophotometers are not suitable for industrial environments with high and unstable temperatures, dust, moisture, dirt or electric fields. In addition, spectrophotometers are only capable of analyzing small (about 12mm diameter) areas accurately. If a detailed colour analysis of products on an assembly line is required, then the more viable solution would be to use a colour camera with hardware to capture the scene. The colours in this scene can then be analyzed in terms of their spectral components (as shown by Wandell (1987) and Healey (1989)). In the next section, alternative colour representation methods will be described. These colour systems may also be viable alternatives in which to analyze colour scenes.

2.4. Colour Representation in Terms of Equivalent Stimuli

Spectrophotometry depends only on measurements of the wavelength of light and measurements of reflectance or transmittance, both of which can be determined with accuracy. Although tristimulus values do not provide so much information as spectrophotometric data, they are adequate for colour matching and can be derived from spectrophotometric data by a straight forward computational procedure (MacAdam (1981) Ch5). The X, Y, and Z tristimulus values serve as the basis for locating a sample in any model of colour space, and therefore provide a fundamental basis for a language of colour. Two samples having identical tristimulus values would lie at the same point in a model and would be a visual match. Two samples which have different tristimulus values would lie at different points, and would not appear to match.

It is also possible to transform these tristimulus values into an approximation of the spectral

power distribution of light entering the camera (Healey (1989)), thereby allowing colour variation within image regions to be estimated.

This section gives a more detailed explanation of the colour spaces introduced in section 2.2. The colour spaces that are described are the CIE chromaticity coordinates, the RGB colour system, the Munsell colour system, and the L*u*v* and L*a*b* colour systems. Finally described are the perceptual colour systems derived from algorithms and matrix transformations using RGB.

2.4.1. The CIE Chromaticity Coordinates For Colour Representation

Instead of matching and comparing colours with spectral curves, a major simplification was made to this cumbersome colour system. In 1931 the standard CIE chromaticity diagram was created (see Figure 2.5). Chromaticity defines colour ratios by a purely additive system in normalized form. The CIE 1931-XYZ tristimulus values of a colour can be obtained by the following methods:

- from the colour's spectral power distribution curve (MacAdam (1981) Ch5),
- by colour matching with a colorimeter (Einhorn (1990b)),
- or (if the colour is from a monitor image) by a matrix transformation from a monitors RGB values representing the colour. The matrix is derived from the chromaticities (i.e. the x and y standard values) of the three primary phosphor colours. For example the NTSC colour chromaticity standard adopted in U.S.A. and Japan (Tajima (1983), Burger and Gillies (1989)).

A normalisation of the tristimulus values yields the chromaticity coordinates x,y,z:

$$x=\frac{X}{X+Y+Z} \qquad y=\frac{Y}{X+Y+Z} \qquad z=\frac{Z}{X+Y+Z}=1-x-y \tag{2.1}$$

Though the XYZ system is adequate for accurate colour description, Euclidean distances in XYZ space do not correspond to colour differences as perceived by humans. It is well known that the green part in the (x,y) chromaticity diagram is much exaggerated . This type of chromaticity diagram, however is often used for computerized colour image processing

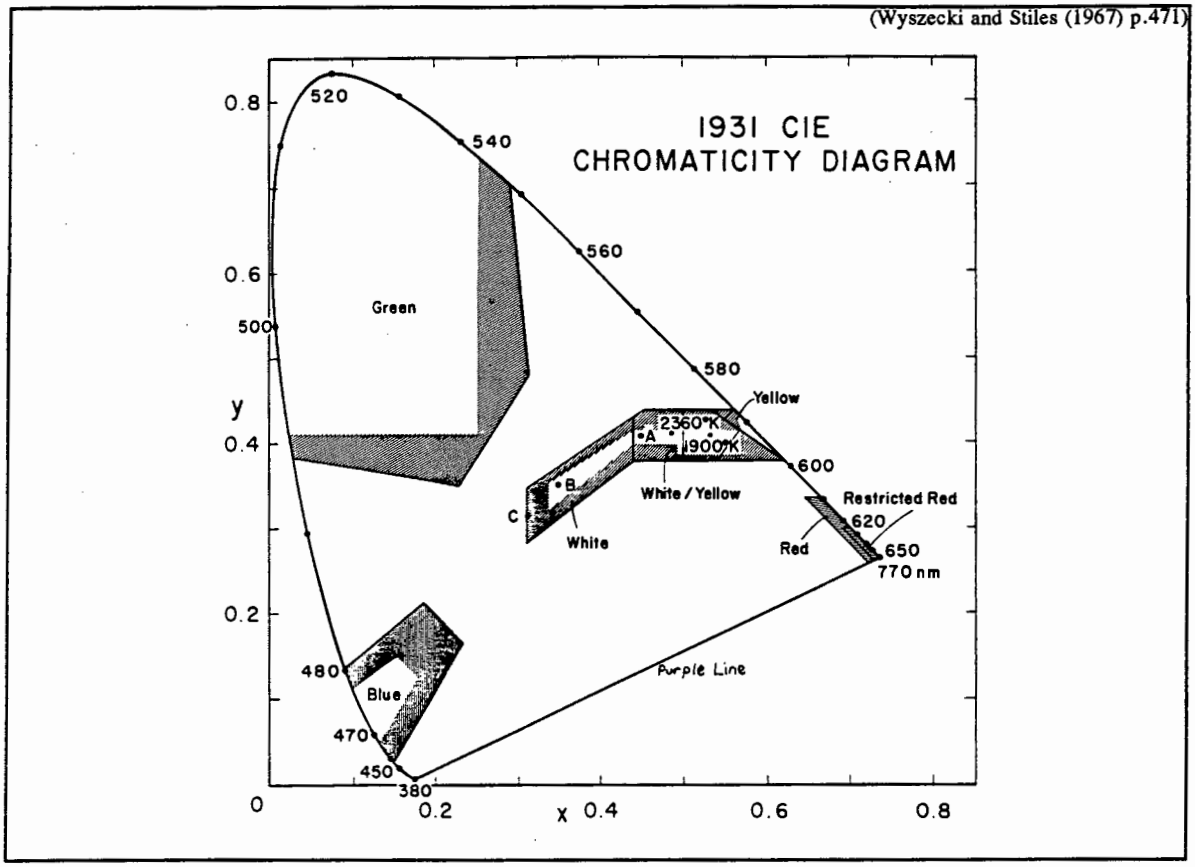


Figure 2.5: 1931 CIE chromaticity diagram with spectrum locus, purple line, the chromaticity points of CIE standard sources A, B, C, and certain domains recommended for signal lights.

because of its convenience (Tajima (1983)).

2.4.2. The RGB Colour System

The varying intensities of light on the red, green or blue phosphors of a colour monitor or camera outputs, result in colour representation by RGB values. A linear transformation from RGB to XYZ is given (shown later in section 3.3.5). It appears that this transformation is similar to that derived for the colour signal **c** in Brainard and Wandell (1990), where colour images are calibrated. Hence, the colour signal is a vector of the tristimulus values XYZ. One way of representing a complete set of colour shades, including intensities, is to use the 3D RGB cube, as shown in Figure 2.6 (Burger and Gillies (1989) p332).

The cube is in the positive quadrant and the primary colours are the variables along the axes. The maximum intensities for all three primary colours can be normalised to the range 0 to

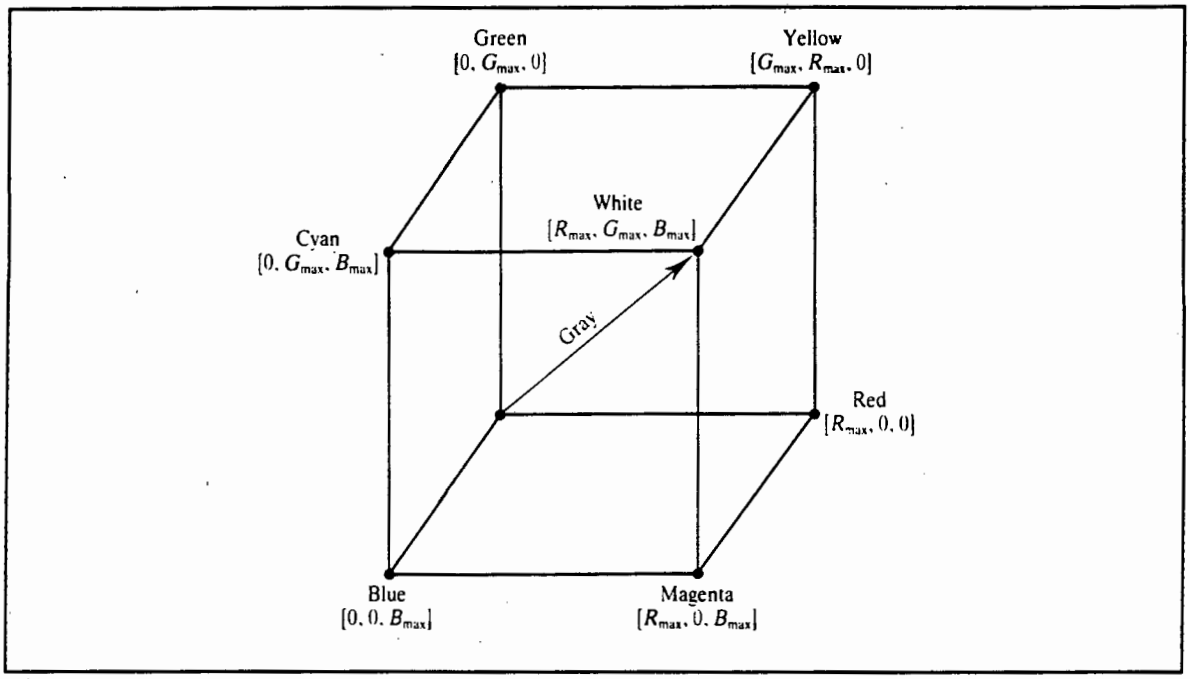


Figure 2.6: The RGB colour cube is an alternative representation of the additive model of a colour monitor. The intensities are not normalized and the cube contains all realizable colour.

1. In general, for computer based systems, each primary can be represented with 8 bits, giving 256 different intensity levels for each primary. Since there are three primaries, it is possible to form 256^3 (over 16 million) different colours on a colour monitor. According to Burger and Gillies (1989) the human eye can distinguish approximately 128 different hues. For each hue, around 20 to 30 different saturations may be seen as different colours. The human eye is also capable of distinguishing between 60 to 100 different brightness (or intensity) levels. Therefore the eye can distinguish approximately 350 000 different colour shades. Hence, in terms of fruit colour sorting, differences unseen to human eye may be noticed by a device using the RGB colour system. The rest of this section highlights where the RGB system has proved to be useful in colour recognition processes.

Applications of the RGB colour system

Healey (1989) and Wandell (1987) use the RGB values (from colour images) as tristimulus values to represent the colour signal to be analyzed as a spectral power distribution. The reason they keep with the spectral curve idea, is so that the R, G and B phosphor (or filter) spectral distributions are calibrated out of the image information. This means that any camera or monitor will represent an image the same way on each device, whereas if the RGB spectral curves are not accounted for, each monitor or camera will have a slightly different

representation of an image. This effect can be seen in a television store, where no two monitors show the exact same colours of the same picture.

If only one monitor and one camera were to be used for on line colour classification, then there would be no need to calibrate out the RGB spectral curves. Since the RGB spectral curves would be a constant throughout classification, the timely spectral distribution representation of an image need not be used. Classification could be based purely on the RGB colour system.

An application using the RGB system is given by Ludman *et al.* (1987). They use the RGB values as a basis for colour object identification by monochromatic binary correlation. The authors describe a real-time polychromatic image correlator that uses a magnetooptic (MO) spatial light modulator (SLM) device for pattern recognition based on both the colour and shape of an input object. The object tested was uniformly coated with one colour. The proposed system utilizes a multichannel spectral matched spatial filter employed in a binary coherent optical correlator. Input colour images are transformed into binary colour coded coherent images by a colour grating. The colour encoded images are read out by a charged coupled device (CCD) interfaced with a MO SLM. The colour encoded binary images are then processed by a multichannel joint spectral matched spatial filter synthesized by monochromatic light. Recent advances in the design of SLMs have led to the development of real time optical pattern recognition systems with potential applications in automatic inspection.

Another application of the RGB system is given by Keller *et al.* (1986). They demonstrate a technique for colour image analysis of food. The analysis of each colour image is based on histogram analysis of the three primary colour components (RGB). Their experimentation was on analyzing steak and how well cooked the inside of the steak was. The results show that this method is more sensitive to actual colour distribution of the meat analyzed than a previous temperature measuring method. As. expected the shape of the histogram of the red component provided the most distinctive features for separating the various levels of cooked steaks, with the shape of the blue component providing the least differences.

A further application using the RGB system is given by Eklundh *et al.* (1980). They show three approaches to reducing errors in colour pixel classification. The methods use the RGB pixel components as features for classification. The first method is a postprocessing approach using iterated reclassification based on comparison between the class of the pixel and the classes of the pixel's neighbours. The second method is the preprocessing approach using iterated smoothing, by averaging pixels with selected neighbours, prior to classification. The third method is by relaxation, which is a probabilistic classification of pixels followed by iterative probability adjustment. In experiments using a colour image of a house, the relaxation approach gave a markedly superior performance, by eliminating 4 to 8 times as many errors as the other methods did.

The RGB system, which is the simplest and the most practical does show promise as a colour system for on-line fruit sorting, provided all three colour components are used in the classification process.

2.4.3. The Munsell Colour System

The Munsell system provides a perceptually uniform colour space defined in the three attributes called H (hue), V (value), and C (chroma) which are determined on human perceptual experience of object colours. Hue (H) is the attribute of a colour perception denoted by red (R), yellow (Y), green (G), blue (B), purple (P), and so on. The major hues consist of R, Y, G, B, P, and the five half way hues of YR, GY, BG, PB, and RP. Each major hue is divided into 10 points, and then the total 100-point hue scale is arranged at perceptually equal spacing as a hue circle (see Figure 2.7). Value (V) corresponds to lightness. The scale gives 10 perceptually equal steps ranging between ideal black (V=0) and ideal white (V=10). Chroma (C) is similar to saturation or purity, and represents the amount of grey in the colour with the same lightness.

Tajima (1983) states that the Munsell system is widely applied in industry. However, because of the cylindrical property, it is complicated to compute a colour difference between two given colours in the Munsell system. Tajima (1983) along with Wyszecki and Stiles (1967) and MacAdam (1981), mention that no mathematical expression exists for the transformation

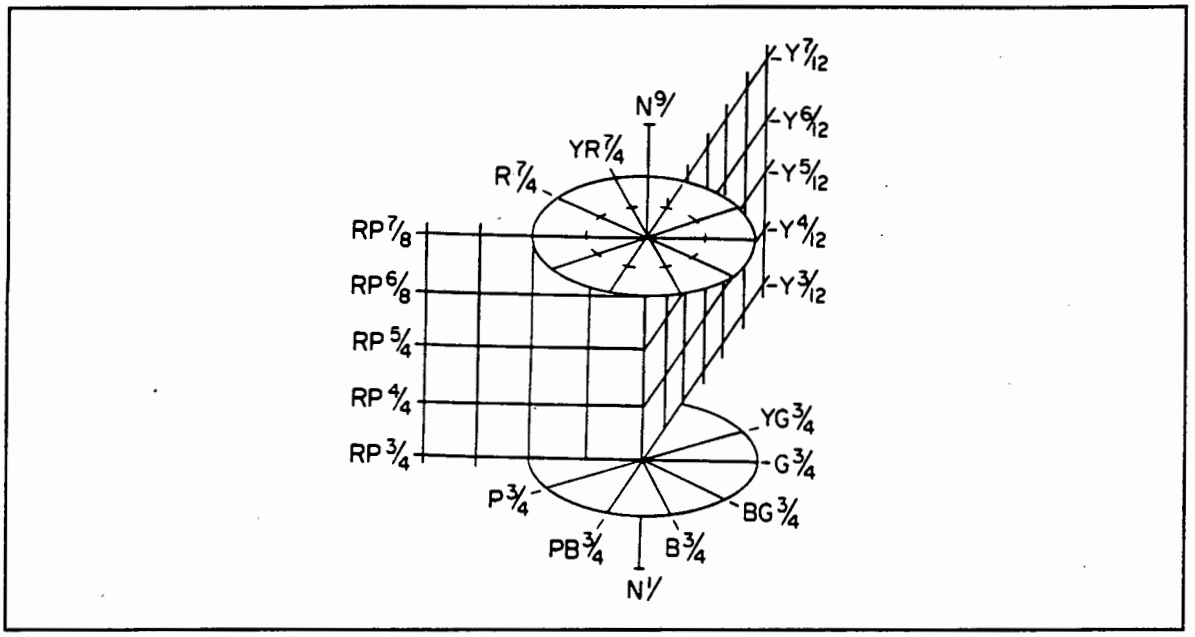


Figure 2.7: Munsell's cylindrical arrangement of colours. Constant-value scales of chroma are shown on horizontal lines in radial planes of constant hue, RP and Y. Constant-chroma scales of value are vertical.

from XYZ to Munsell. Tables do exist in which every Munsell colour is written in XYZ values. It is necessary to retrieve X, Y, Z values in the table which are closest to available X, Y, Z values and get corresponding Munsell values by means of interpolation. Therefore, the Munsell system does not suit high speed computer graphics.

In contradiction to the above paragraph, Tominaga (1986) refers to a mapping method he developed to transform the observed colour signals to the Munsell colour space. From this, the perceptual attributes of a colour image can be predicted quantitatively. The histograms of the three attributes are then analyzed, based on a recursive thresholding method, for image segmentation. This operation corresponds to the recursive detection of compact clusters in a colour specification space. Experimental results demonstrated the feasibility of the proposed method. Although colour images could be segmented fairly accurately, the method did not seem appropriate for real time colour analysis.

2.4.4. The CIE L*u*v* and CIE L*a*b* Colour Systems

To employ human eye sensitivity for computer processing, a mathematical expression is necessary in which calculated colour differences approximate those sensed by human eyes.

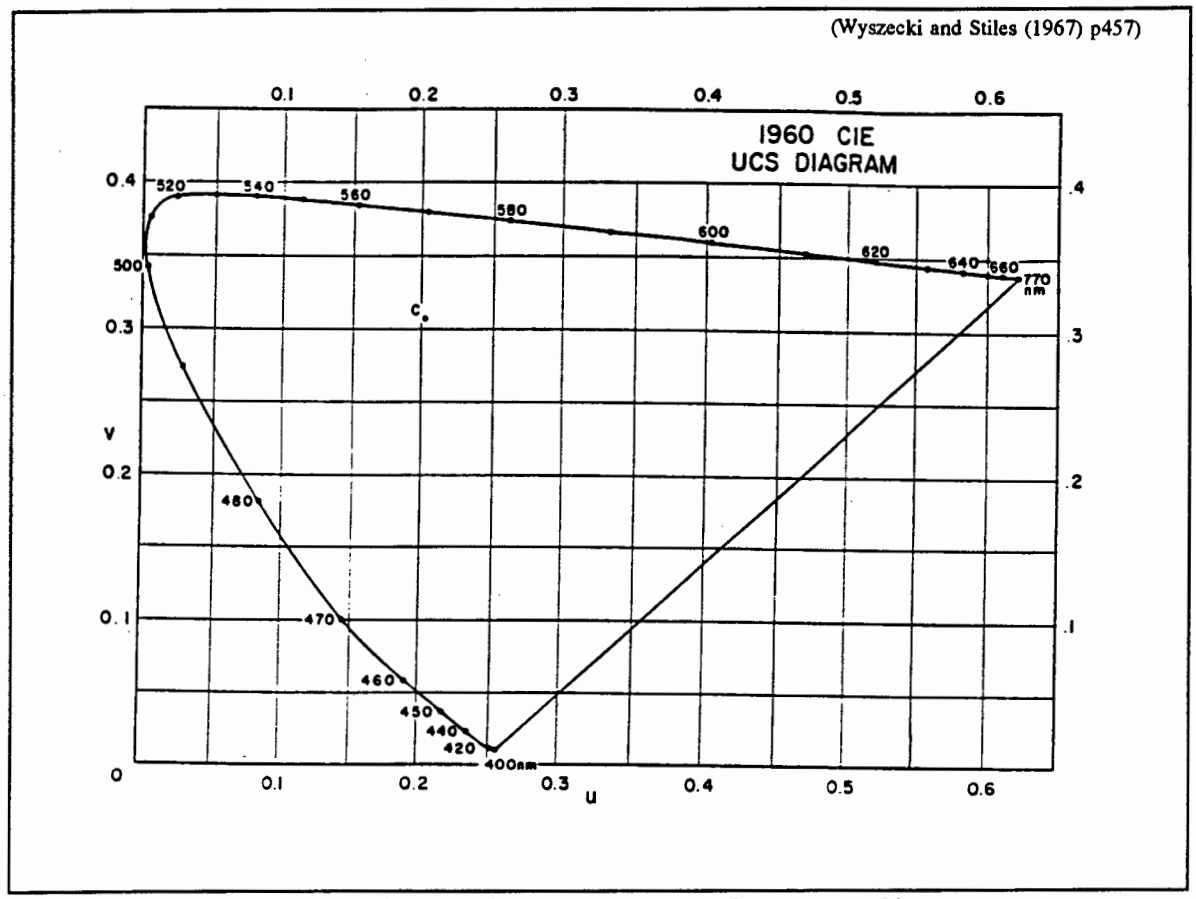


Figure 2.8: 1960 CIE Uniform Chromaticity Scale (UCS) diagram.

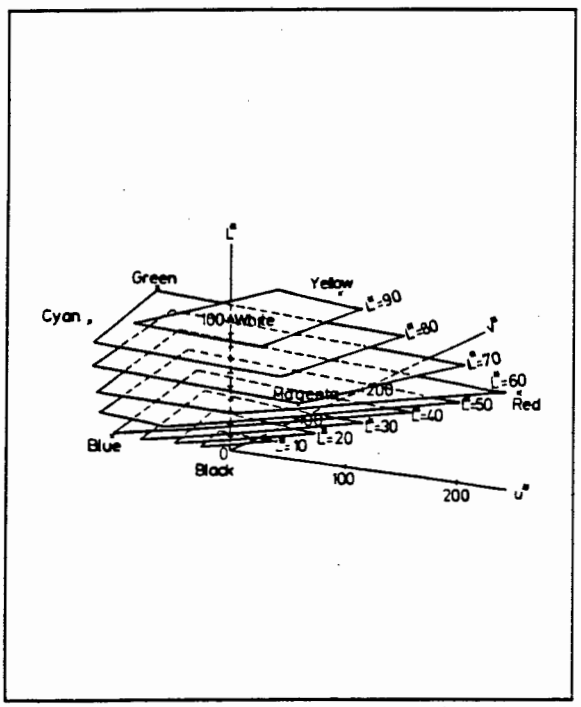


Figure 2.9: Cross sections of displayable regions in CIE L*u*v* space.

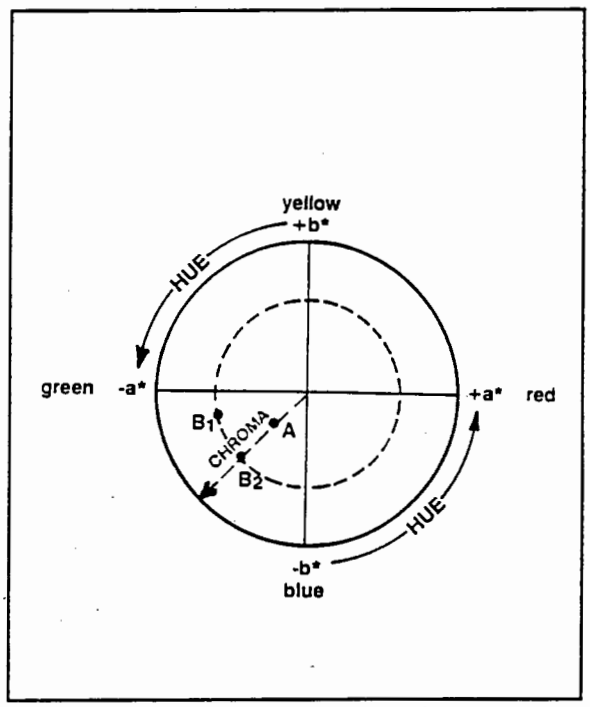


Figure 2.10: A cross section of the CIE L*a*b* colour space at a constant lightness level.

Such a mathematical system is called a uniform colour scale. In general, it is uniform colour spaces that allow for the Euclidean distance between two points to represent their colour difference. The CIE L*u*v* system (Figure 2.9) and CIE L*a*b* (Figure 2.10) system are such systems which are recommended by CIE for colour difference evaluation use. Since the L*u*v* system is the newer system and seems to approximate the colour difference sensed by humans more closely, this system is recommended by Tajima (1983) for computer graphics. The L*, u*, v* uniform colour system used by Tajima is almost identical to the 1964 CIE U*, V*, W* system described in Wyszecki & Stiles (1967) p460. This system is an updated version of the 1960 CIE-Uniform Chromaticity Scale (UCS) L u v system (Figure 2.8). Both systems are simple transformations of the CIE tristimulus values (X, Y, and Z) (to be shown in chapter 3).

The L*u*v* and L*a*b* systems are similar in that L* (lightness) is calculated the same way for both systems, u* and a* are the chromaticity variation from green to red, and v* and b* are the chromaticity variation from blue to yellow. Both systems reference a standard white illuminant. The main differences between the L*u*v* and L*a*b* systems are that u* and v* both account for the lightness L* whereas a* and b* are independent of L*. The L*a*b* system is recommended for viewing under reflected light, while the L*u*v* system is recommended for viewing additive sources (such as a colour monitor) (Robertson (1988)).

By using only the u* value, a linear scale for colours on, say, apples (greens, yellows, and reds) could be investigated. Computation time could hence be greatly reduced since one value (u*) would represent a colour instead of three values (RGB). In general, however, conversion from RGB to L*u*v* is considered computationally expensive if carried out by software. However, high speed conversions can be accomplished by hardware, using combinations of large scale integrated (LSI) circuit conversion tables and adders.

The L*, a*, b* system is a colour difference system more suited to industry because of computation speed (its derivation will be given in chapter 3). The following highlights applications of this colour system.

*Applications of the L*a*b* colour system*

Celenk and Smith (1986) use the L*a*b* system for colour image segmentation by clustering and parametric-histogramming. The recursive procedure described, tends to integrate the fundamental operation mechanism of human colour perception with the statistical data analysis concept of mathematical pattern recognition. It detects the image clusters efficiently and determines their boundaries correctly. At every iteration step, the algorithm detects the most prominent cluster or mode and all of its spectral neighbours in the uniform colour space. In order to make the detection process computationally efficient, it first approximates the underlying clusters by fitting some circular cylindrical volume elements. This best estimates the three-dimensional (3-D) colour distributions of these clusters in accordance with the colour perception mechanism of the eye. That is, the boundaries of each volume element are composed of two constant luminance planes, two constant circular chroma cylinders, and two constant angular hue planes. Each is derived from the sequentially constructed 1-D zero-, first-, and second-order parametric histograms of the cylindrical coordinates (i.e., L*: Lightness, H^0: Hue, C*: Chroma) of the uniform L*a*b* colour space.

Celenk and Smith (1986) conclude that for colour segmentation, the colour system should implicitly satisfy the condition that numerical differences in feature space should be directly proportional to perceptual differences in the human visual system. The L*a*b* system satisfies such a criterion.

Robertson (1988), Miller and Delwiche (1989), Galloway (1989), and Uehira and Komiya (1990) are other authors who have used the CIE L*a*b* colour system for their colour analysis work. Their criterion for using the colour system is the same as that given in the above paragraph by Celenk and Smith (1986). The L*a*b* system:

- has been used with a user interface for the effective use of perceptual colour spaces in data displays (Robertson (1988));
- has been the colour system used for a peach grading system, classifying at one peach per second (Miller and Delwiche (1989));
- has been the colour system used on the ACS CHROMA- Quality Control System which is a tool for setting tolerances for colour approval and for running Pass/Fail evaluations on batches in production (Galloway (1989));

- has been used with gradient-index (GRIN) rod lenses for determining pattern edge colour differences in colour images (Uehira and Komiya (1990)).

2.4.5. More Perceptual Colour Systems

The transformations from RGB via the tristimulus values XYZ to the L*a*b* or L*u*v* or Munsell colour systems can generally be considered as complex and time consuming. The L*__ systems require at least a cube root operation and the Munsell system may require interpolation for accuracy. It has been found that in many cases the precise properties of the colour space are not needed (Niblack (1986)). Other colour spaces such as the HSI, HSV, HLS and HYC systems have been defined that are similar to perceptual colour spaces and have simple computational transformations.

The following definitions must first be clarified (Rossotti (1983)):

HUE : is the primary sensation of colour (e.g. blueness, or greenness) and varies with any change in dominant wavelength. It is generally measured as an angle. Hue representative symbols are H, h, or Θ.

SATURATION : (or CHROMA) is the extent to which a wavelength dominates the light, or the colour purity. Saturation representative symbols are S or C.

INTENSITY : (or VALUE, or LIGHTNESS, or BRIGHTNESS, or LUMINANCE) is the degree of brightness. The so called 'natural' or 'achromatic' colours, black, grey and white, are of zero saturation, and differ from each other only in brightness. Intensity representative symbols are I, V, L or Y.

The rest of this section highlights the features of some other perceptual colour systems investigated. These systems can be divided into two main areas. The first system is colour transformation algorithms that derive perceptual colour spaces from the RGB system. The second system is an extension of the first, where matrices are derived from the colour transformation algorithms. These matrices operate on the RGB system to obtain HSI

perceptual colour systems.

2.4.5.1. Colour transformation algorithms available for computer programs

Various methods exist in which one may intuitively derive a perceptual colour space and then relate this to an algorithm for a computer program. Burger and Gillies (1989) p335 derive the HSV model in such a manner. To relate the RGB model with more intuitive colour concepts of hue, saturation and brightness (value), it is necessary to realize that any three absolute intensities (RED, GREEN, BLUE) can be produced by the sum of a pure colour (hue) and a white light. Saturation expresses the relative magnitude of the pure colour and the white components. Since white light contains equal portions of RED, GREEN and BLUE, a white component can be determined from the minimum of the three components of any colour:

$$RED_w = GREEN_w = BLUE_w = MIN(RED,GREEN,BLUE)$$

The pure colour is the mixture of at most two primary colours and can be calculated by the following expressions:

$$RED_h = RED - MIN(RED,GREEN,BLUE)$$
$$GREEN_h = GREEN - MIN(RED,GREEN,BLUE)$$
$$BLUE_h = BLUE - MIN(RED,GREEN,BLUE)$$

The index 'h' indicates pure hue and at least one of these components will be equal to 0. The HSV model can be represented by a hexagonal cone, as shown in Figure 2.11. Colour, or hue, is defined by an angle with red being at angle 0^0, yellow at 60^0, green at 120^0, and so on. A simple intuitive formula is used for the calculation of the hue angle for pure colours (which contain no more than two primaries). For example, between red and green:

$$\text{Hue angle} = (120.GREEN) / (RED_h + GREEN_h) \quad \text{in degrees}$$

When $GREEN_h = 0$, red (hue = 0^0) is obtained, when $RED_h = 0$, green (hue = 120^0) is obtained and when $RED_h = GREEN_h$, yellow (hue = 60^0) is obtained. Similar expressions are derived for the five other sides of the hexagon.

Although it would be possible to calculate the saturation value defined by the CIE curve, a much simpler formula is used for the value of the saturation in this intuitive model:

$$\text{Saturation} = 1 - (MIN(RED,GREEN,BLUE) / MAX(RED,GREEN,BLUE))$$

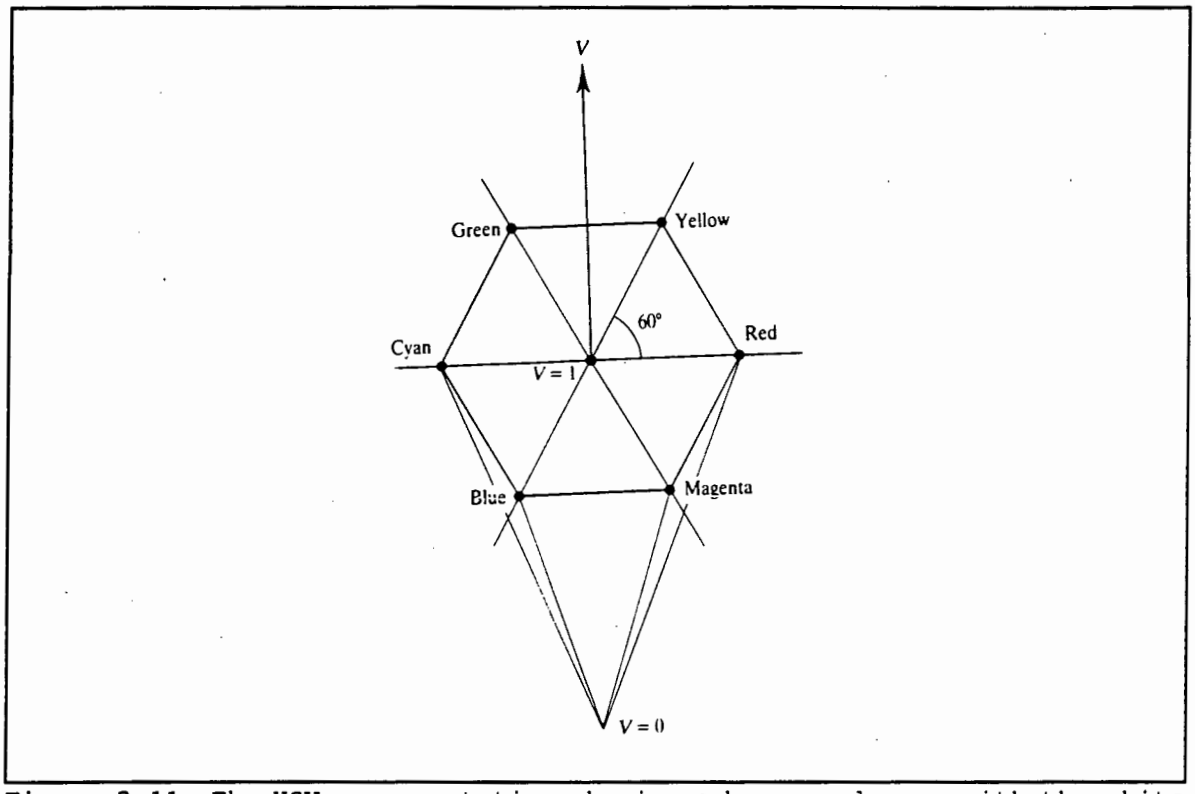


Figure 2.11: The HSV representation showing a hexagonal cone with the white or gray shades lying on the vertical axis. Colour is represented by an angle and constant value by a horizontal plane.

Saturation in the HSV model is similar to purity , being equal to 1 for pure colours and 0 for white. However, its value is not exactly equivalent to the saturation values defined by the CIE diagram. The brightness value, Value, is expressed simply by the intensity of the maximum component:

$$\text{Value} = \text{MAX(RED,GREEN,BLUE)}$$

There are similarities between the RGB colour cube and the HSV hexicube. For example, the top hexagonal facet of the hexicube is a mapping of the three facets of the cube that adjoin the axes (pure hues with maximum intensities) on to a planar hexagon.

Tektronix (in Burger and Gillies (1989) p337) developed another system called the HLS model, which is similar to the HSV model, but a double cone is used with the black and white points placed at the two apexes of the double cone. This is similar to the HLS system recommended for computer graphics by the Graphics Standards Committee, Siggraph, ACM. The algorithm is given in Niblack (1986) p62. Furthermore software examples and program code are given in Niblack (1986) p205 for RGB to HSI and vice versa conversions. These transformation algorithms can be incorporated into integrated circuits, thus yielding hardware

chips capable of RGB transformations to perceptual colour spaces and vice versa. For example, Data Translations Incorporated give the equations used in their DT7910 RGB to HSI conversion chip (Data Translation Manual and Morrissey-Golas (1989)).

2.4.5.2. Matrix transformations from RGB to HSI

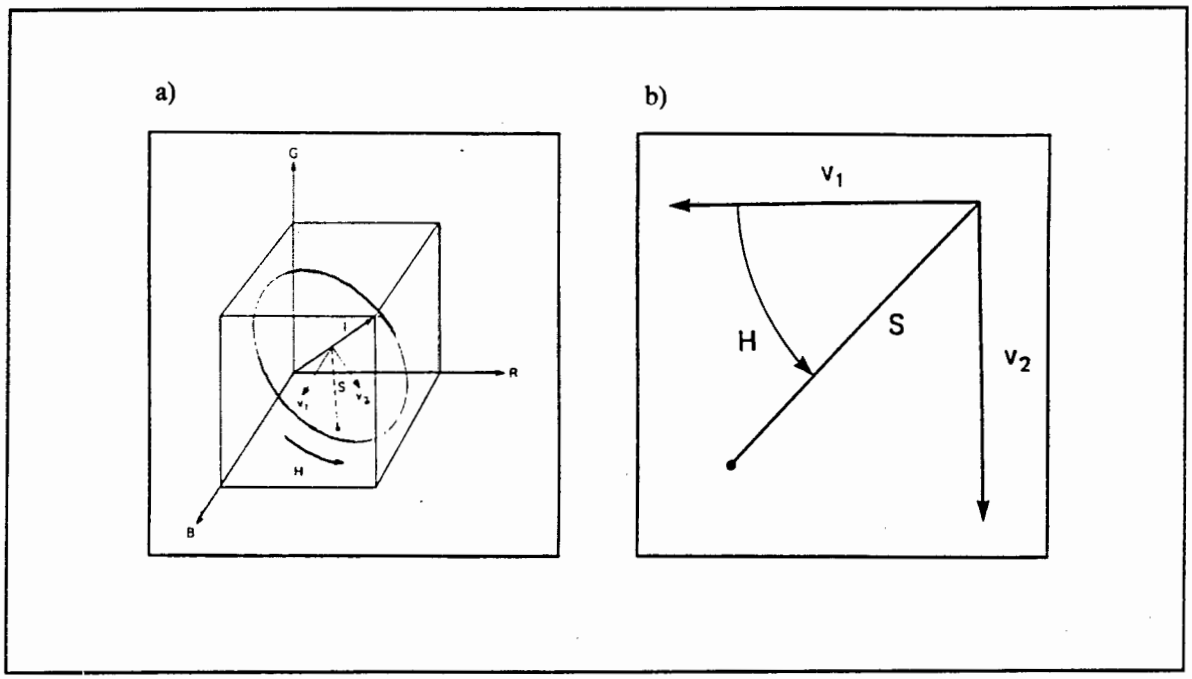


Figure 2.12: HSI coordinates within the RGB cube: (a) The I, v1, and v2 axes in the RGB cube, and (b) converting from v1 and v2 to H and S.

It has been shown that several algorithms exist in order to derive, as simply as possible, a perceptual colour space from RGB components. An even simpler approach would be to have a single matrix transformation to convert from RGB to an HSI colour space. This would result in more efficient software code. In general this matrix can be derived from some of the algorithms mentioned above or from Figure 2.12.

The RGB cube shown in Figure 2.12 represents all possible colours available for display on an RGB monitor. From the geometry the HSI coordinates can be placed within the RGB cube. Intensity I is measured along the central R=G=B diagonal of the colour cube. Hue H and saturation S are polar coordinates in the plane perpendicular to the I diagonal.

The transformations are all similar in that they involve a (3x3) matrix multiplication to the (3x1) RGB column vector. This result yields the (3x1) (I,v1,v2) column vector. Where I is

intensity and v1, v2 are line vectors which make a Hue angle and Saturation radius. Some examples of matrices which have been derived and used are given below.

Examples and Applications of RGB to HSI Transformations Using Matrices:

According to Niblack (1986) p59, if H is set to zero in the direction of blue, the transformation equations may be derived from the Figure 2.12. The intensity axis I is R=G=B, the vector v1 is orthogonal to I in the plane of I and B, and v2 is the vector I cross v1 to complete the orthogonal system. The transformation is :

$$\begin{pmatrix} I \\ v1 \\ v2 \end{pmatrix} = \begin{pmatrix} 1/3 & 1/3 & 1/3 \\ -1/\sqrt{6} & -1/\sqrt{6} & 2/\sqrt{6} \\ 1/\sqrt{6} & -2/\sqrt{6} & 0 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{2.2}$$

v1 and v2 are converted to a polar coordinate system giving H and S to complete the transformation:

$$H = \tan^{-1}\left(\frac{v2}{v1}\right)$$
$$S = \sqrt{v1^2+v2^2} \tag{2.3}$$

Were H is in the range 0^0 to 360^0.

Lehar and Stevens (1984) give a slightly different matrix transformation to (2.2) that allows the RGB cube to be tilted to form a regular hexagon (in the Hue/Saturation plane) as if the cube were viewed down the R=G=B diagonal line. Their work involves using this transformation for enhancing digital colour chromaticity whilst reducing processing time. In this method, colour image data are encoded into a greatly reduced subset of colours (from 256^3 to 256 colours) for convenient storage and display. Processing can thus be done on this set, or look up table (LUT), of colours rather than the pixel data values, resulting in chromaticity adjustments that can be viewed interactively (an example is given as to how the object of interest can be enhanced and surrounding unwanted scene totally removed from the image). A colour reassignment from RGB to HSI results in a classified or colour-enhanced

image. A fractal curve (the Peano curve), mapping the three dimensions of colour space to one dimension, is used for encoding the image. Applications for this method include colour enhancement, extraction of features from complex scenes, and classification of multispectral images.

Ramirez in Niblack (1986) p60 presents another matrix transformation that accounts for the luminosities of the phosphors of an NTSC standard monitor. Now the intensity is given as the luminance signal Y. As in PAL Colour Tv (1967), the adding of colour signals to make white light must be done in the proportion 0.31: 0.59: 0.11 (red: green: blue). This ratio now allows the intensity to fit more to the $V(\lambda)$ response curve of the human eye (Figure 1.1).

Benson (1986) in Slaughter and Harrell (1987) also provides a different RGB to HSY transformation, and here too the NTSC monitor phosphors are accounted for. Slaughter and Harrell (1987) use this transformation in developing a colour vision system for robotic orange harvesting. The system segments a scene image into possible orange pixels and background pixels. The criterion used to classify orange and non orange pixels, is based on upper and lower experimentally determined thresholds on hue and saturation. The luminance signal Y, is used in a feedback loop to control the aperture on the camera analyzing the scene. Since real time vision guidance in robotic fruit harvesting requires high speed image processing techniques, the colour segmentation algorithm they describe has potential for this application, particularly if implemented in hardware.

The reason why there are no standard RGB to HSI conversion matrices (and vice versa) is because each transformation matrix was developed with particular spectral reflectance curves of R,G and B in mind, and with an arbitrary colour Hue as the 0 degree starting angle (eg blue in the case of figure 2.12). The matrix transformations introduced in this chapter will be dealt with in more detail in the chapter 4, where the effect of each transformation can be compared.

2.5. Some colour sensing devices known to be available

The CS70 series intelligent colour sensor (Yamatake-Honeywell Manual), Protea laboratories colour analyzer (Roper (1992)), The IMAGE-CLD colour module (Matrox Image Series), and The ICS Texicon computer spectrophotometer (ICS Texicon Spectraflash Manual (1991)) are colour analysis devices know to be imported to South Africa. They all have the features of being able to analyze any coloured object, with some or all of the colour representation systems described in this chapter. Their major drawback, however, is that they all analyze (except the IMAGE-CLD) only one patch (between 8 and 20 mm diameter) on the object. The sensor head must either be in contact with the object, or placed a set distance from the object. Hence, when considering an automatic on line fruit sorting machine, if a detailed colour analysis is required of the total fruit skin, the above mentioned devices may not prove useful in such a situation. However, a device such as the ICS Texicon computer spectrophotometer, is ideal for experimental analysis into which colour system best represents fruit colour. Such an experiment will be covered in the next chapter. Further experimental results given in later chapters, will use and transform the RGB values given by images captured with the Matrox MVP-AT frame grabber (Matrox MVP-AT User's Manual (1989)). The IMAGE-CLD can output RGB, HSI or YIQ signals associated with pixels on an image, whereas the MVP-AT only outputs the RGB signal associated with the pixels. The MVP-AT was used since it is a considerably cheaper module than the IMAGE-CLD.

This chapter has introduced several colour representation methods. All of these methods have been investigated with the intention of finding one that will be the most appropriate system for automatic fruit sorting. It has been shown that spectral reflectance and spectral power distribution curves offer more detail in colour representation than any of the three coordinate colour systems. However, systems such as the CIE chromaticity coordinates and RGB provide sufficient colour representation. The L*u*v* and L*a*b* colour systems are ideal if colours are to be matched by Euclidean distances. In addition, the HSI and other perceptual colour spaces are generally easier to derive and represent colour adequately. All colour systems can eventually be derived from the CIE XYZ tristimulus values. The next chapter analyzes the performance of most of the colour systems described here, by identifying certain colour regions of a fruit.

CHAPTER 3

Experimental Comparisons Between Colour Representation Systems

3.1. Introduction

One of the main objectives for the project being described, is to find a colour system that can adequately represent colour differences in a single fruit. A major criterion for choosing the colour system is that the colours should be classified in real time. Hence, before the classification process can take place, the colours of the scene must first be represented by some colour space in real time. This may involve hardware which transforms each colour observed to the adequate and more processable colour representation.

The main emphasis on fruit colour sorting for this project is to sort out the grades of a particular cultivar. Obviously if the fine detail between colour grades can be determined, then sorting between different cultivars such as red, green and yellow apples, should be a trivial task.

This chapter gives a comparison of most of the colour representation systems highlighted in chapter 2. The comparisons are made by using experimental results measured on the ICS Texicon colour analyzer (property of Woolworths, Cape Town). This apparatus was used to measure spectral power distributions, and X, Y, Z tristimulus values. The device gave the colour reading for samples placed on a 12mm diameter aperture (see Figure 3.1). The ICS Texicon colour analyzer could also simulate various types of lighting conditions. In the two experiments that were conducted, the CWF (Cool White Fluorescent) and TL84 (a fluorescent light with better colour rendering) lighting conditions were chosen. These lighting conditions were to simulate different store display lightings. Colours observed in a large field (within 10 degrees subtended from the eye) and colours observed in a small field (within 2 degrees subtended from the eye) appear slightly different (Wyszecki and Stiles (1967)). This phenomenon was accounted for in the experiments, since the colour analyzer allowed for the readings to be taken at small field and large field observations.

How the spectrophotometer works: A sample is placed against the measuring aperture (1). The aperture is an opening in the wall of a white integrating sphere (2). Light from the Xenon flashlamp (3) passes through a D65 filter to give daylight simulation. The filter wheel (5) permits ultraviolet cut-off. The sample beam (6) measures the reflected light from the sample, and the reference beam measures the reflected light from the sphere wall. Light reflected from the sample passes down the sample beam to the sample analyzer (8), where light is split into its spectral components by a diffraction grating; the diode array records simultaneously the intensity of the light at specific wavelengths. At the same time, light passes down the reference beam from the sphere wall into the reference analyzer (9), which is identical to the sample analyzer. The specular port or gloss trap (10) can be used to include or exclude the gloss component of any measured sample. Lens (11) is used to focus the sample beam onto smaller sample areas. The microprocessor in the spectrophotometer processes the measurements and transmits the reflectance data to the computer.

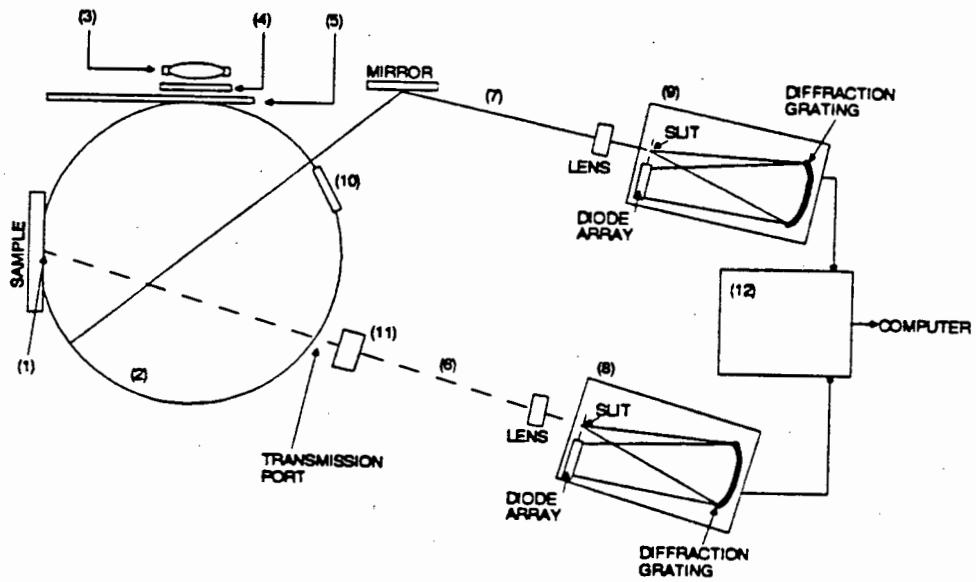


Figure 3.1: A simplified diagram of the Spectraflash, which is the hardware used with the ICS Texicon colour analyzer (ICS Texicon Spectraflash Manual (1991)).

The first experiment was to demonstrate that green and red apple cultivars can be easily distinguished. The method involved a comparison between six grade one apples consisting of three apple cultivars. Namely two Granny Smiths, two Golden Delicious, and two Red Starkings. The spectral power distribution curve, and tristimulus values were measured at similar positions for each fruit (position A on Figure 3.2). The comparative results are given in section 3.2. The chromaticity coordinates are also given for the two lighting conditions (CWF and TL84) and the two viewing fields (2^0 and 10^0). By way of mathematical

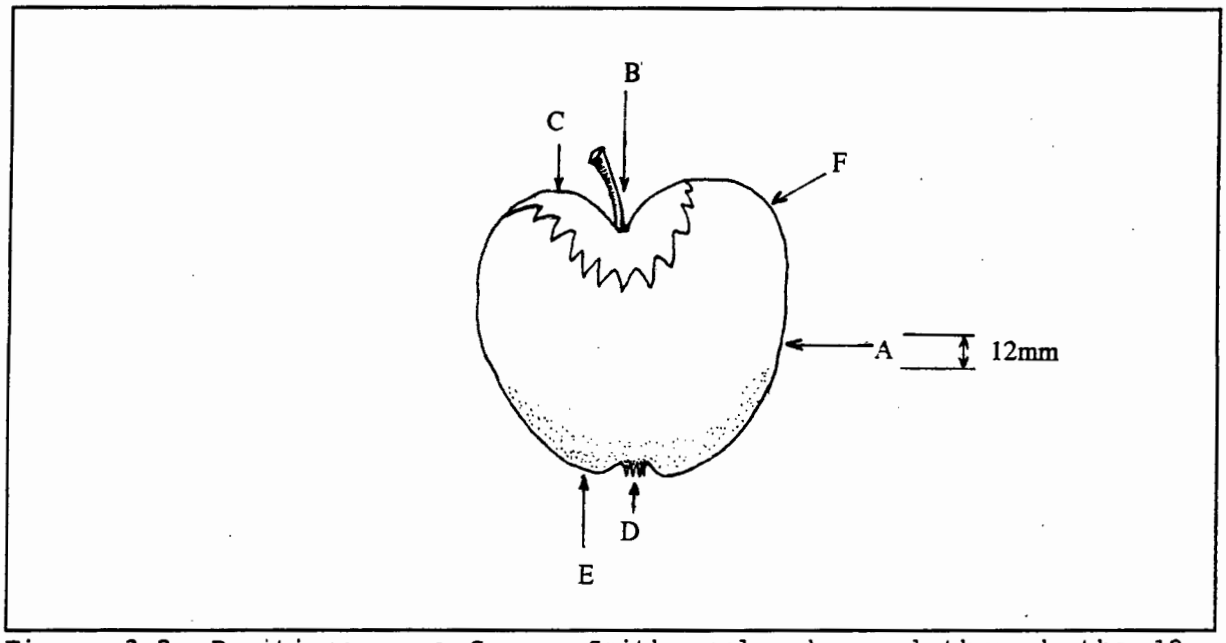


Figure 3.2: Positions on a Granny Smith apple observed through the 12mm diameter aperture of the ICS Texicon computer spectrophotometer.

transformation, the X, Y, and Z tristimulus values measured were used to calculate the colour coordinates of the previously mentioned colour spaces in chapter 2. Since, all colour spaces can distinguish between red, green and yellow, the comparison between the green and red cultivars was considered trivial, hence, values corresponding to other colour spaces were calculated, but are not shown for this experiment.

The second experiment involves a comparison between the different shades of green on a single Granny Smith apple. The aim of the experiment was to determine if all the colour representation methods discussed could equally and effectively distinguish between the green shades. The method involved taking spectral power curves and tristimulus readings for six positions on a grade one Granny Smith apple (see Figure 3.2). For each position the results were given for two lighting conditions (namely CWF and TL84) and for large field (10^0) and small field (2^0) observations. Section 3.3 gives the calculations used to determine, from the XYZ tristimulus values, the:

- CIE 1931 *x,y* chromaticity coordinates;
- CIE 1960 UCS *u,v* values:
- L*u*v* values and L*a*b* values;
- RGB values;
- and HSI values.

Section 3.3 also gives the comparative results of this experiment. This chapter ends with a summary diagram of the relationships between the colour systems and a discussion on the results obtained.

3.2. Comparative Results of Colour Systems Representing Three Cultivars

The three apple cultivars being compared in this section are: green Granny Smith; green Golden Delicious; and red Starking. The two oldest colour systems are used in this experiment, namely spectral power distribution curves and chromaticity coordinates derived from tristimulus values. Only these two colour systems were chosen since the ICS Texicon colour analyzer automatically produced results in these colour representations. In addition, this experiment was aimed at illustrating that the oldest colour representation systems are good for distinguishing between different hues (like red and green) but are not ideal for distinguishing between similar hues (like pale and dark green). The next experiment in section 3.3 will be using most of the colour systems described in chapter 2 in order to find a colour system to distinguish between similar hues.

3.2.1 Spectral Power Distribution Curves for Six Apples

The spectral power distribution curves (i.e. independent of source lighting) of Figure 3.3 distinguish the A position (see Figure 3.2) for two Granny Smiths, two Golden Delicious, and two Starking apples.

Observations

The first grade apples chosen for each cultivar were virtually identical in appearance. The spectral power curves confirm this fact, since the shapes of the curves for each cultivar are similar. The green curves of the Granny Smiths, and Golden Delicious (Figures 3.3a and 3.3b) are undoubtably different to the red curves of the Starkings (Figure 3.3c). All the green apples observed have a prominent spectral peak at about 560nm. It is thus difficult to distinguish between the two green cultivars. However, the curves for the Granny Smiths tend to drop off sharper to the right of their peaks, compared to the Golden Delicious curves. This observation suggests that the Granny Smiths have less yellow, and are thus greener than the Golden Delicious, which is true.

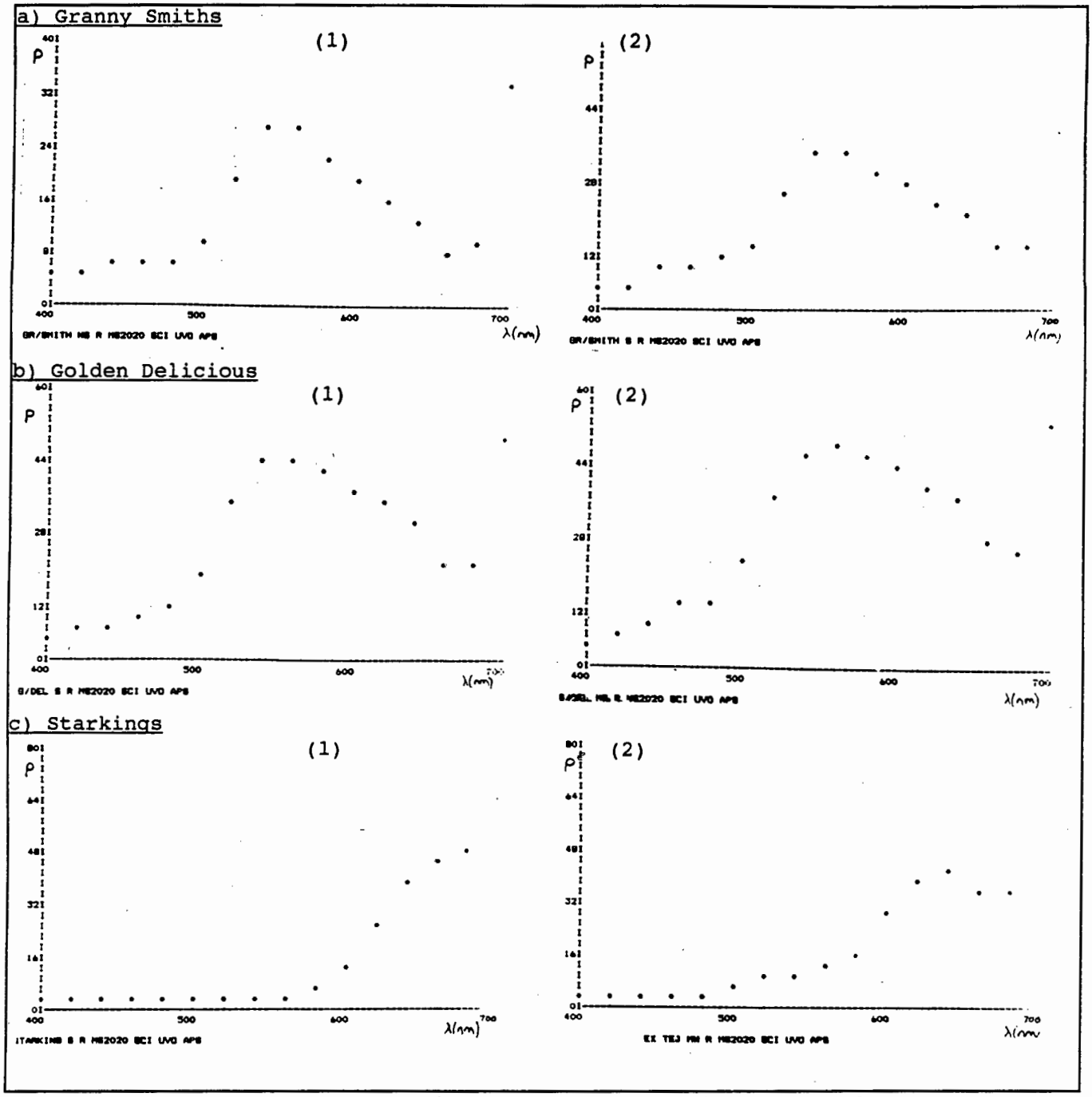


Figure 3.3: Spectral power distribution curves for six apples of three different cultivars.

3.2.2. Chromaticity Coordinates for Six Apples

The colour response of the normal human eye, namely that of the C.I.E. standard observer, can be represented by the ratio of three integrals of the weighted spectrum. The three *tristimulus* values X, Y, Z are defined by:

$$X = \int_{\lambda} P(\lambda)\bar{x}(\lambda)d\lambda$$

$$Y = \int_{\lambda} P(\lambda)\bar{y}(\lambda)d\lambda \qquad (3.1)$$

$$Z = \int_{\lambda} P(\lambda)\bar{z}(\lambda)d\lambda$$

Where $P(\lambda)$ is the radiant power of the light emitted from the object under consideration as a function of the wavelength λ, and $\bar{x}(\lambda)$, $\bar{y}(\lambda)$, $\bar{z}(\lambda)$ are weighting coefficients called *spectral tristimulus* values shown in Figure 1.2. The $\bar{y}(\lambda)$ function has been chosen to be identical with the $V(\lambda)$ function. Hence Y is proportional to the photometric amount of radiation. The ratio of the tristimulus values is of main interest. Hence if the sum of the tristimulus values is used

$$S = X + Y + Z,$$

then the chromaticity coordinates (x,y,z) can be formed:

$$x = X/S\ ; \quad y = Y/S\ ; \quad z = Z/S$$

By their definition $x + y + z = 1$.

It is usual to substitute the continuous integral signs in equations (3.1) with a summation sign. The XYZ tristimulus values can then be calculated by using values from $P(\lambda)$, $\bar{x}(\lambda)$, $\bar{y}(\lambda)$, and $\bar{z}(\lambda)$ at intervals of $\Delta\lambda$ for the visible spectrum (i.e. use λ from 380nm to 720nm). The experiments using the ICS Texicon colour analyzer calculate the tristimulus values for $\Delta\lambda$ equal to 20nm. Obviously, if $\Delta\lambda$ was smaller, say 1nm, then the corresponding X, Y, and Z values would be proportionally much larger than the X, Y, and Z values calculated for $\Delta\lambda$ at 20nm. The magnitudes of the XYZ values will only become important for L*u*v*, L*a*b*, and RGB colour system calculations derived from X, Y. and Z. This fact does not affect the x, y, and z values since they are calculated using the ratios of the XYZ tristimulus values.

Table 3.1 shows the tristimulus values and corresponding chromaticity coordinates for the six apples used in the experiment, and their values under two different lighting conditions and two viewing fields. Figure 3.4 shows corresponding relative positions of the chromaticities for the six colours.

Table 3.1: Chromaticity coordinates for 2 Granny Smiths (GS), 2 Golden Delicious (GD) and 2 Starkings (ST).

Lighting	Apple	x	y	Lighting	Apple	x	y
CWF-2	GS1	0.416377	0.4725	TL84-2	GS1	0.412626	0.481196
	GS2	0.414608	0.484273		GS2	0.406283	0.497365
	GD1	0.430767	0.479248		GD1	0.42982	0.484405
	GD2	0.431521	0.469659		GD2	0.433694	0.472079
	ST1	0.494782	0.359876		ST1	0.527872	0.350102
	ST2	0.501945	0.408831		ST2	0.530328	0.392189
CWF-10	GS1	0.433942	0.45701	TL84-10	GS1	0.428292	0.467078
	GS2	0.433591	0.466639		GS2	0.424255	0.480697
	GD1	0.448543	0.463581		GD1	0.445361	0.470588
	GD2	0.448038	0.455259		GD2	0.447581	0.459754
	ST1	0.494009	0.35717		ST1	0.524461	0.350325
	ST2	0.50882	0.401375		ST2	0.532552	0.389156
					TL84 (x,y)	0.384	0.37
					CWF (x,y)	0.379	0.366

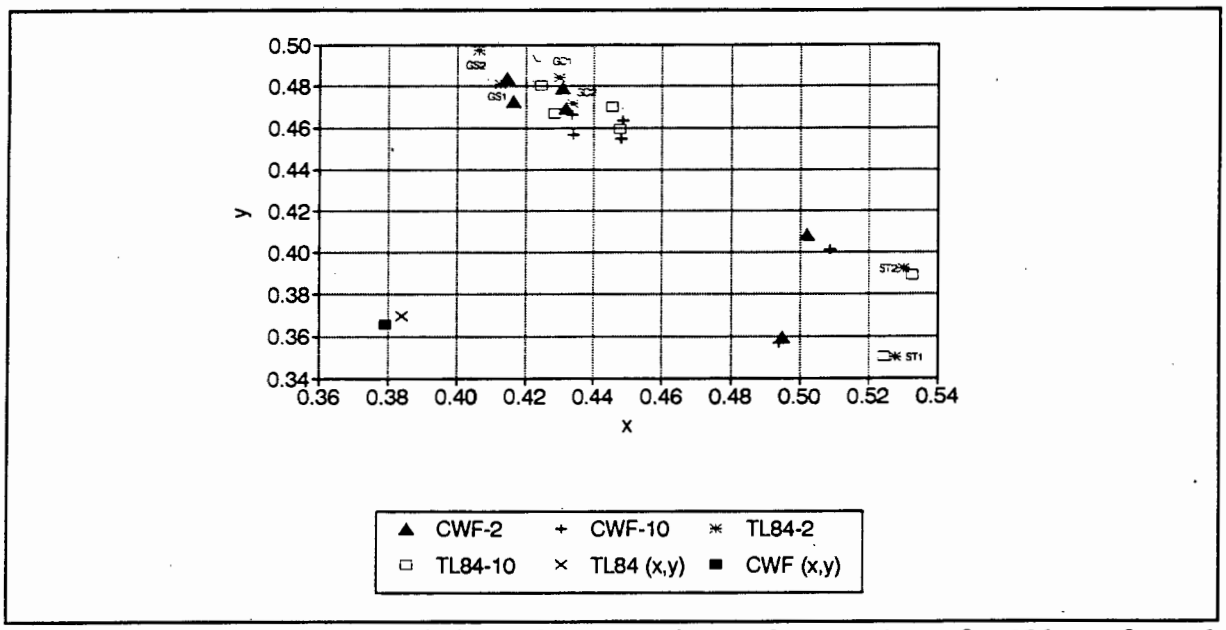


Figure 3.4: The colour positions for the six apples (GS1, GS2, GD1, GD2, ST1 and ST2) on the (x,y) chromaticity plane, for two lighting conditions (CWF and TL84) and two viewing fields (2^0 and 10^0).

Observations

If one can picture the points shown in Figure 3.4 on the chromaticity chart of Figure 2.5, one can see that the green chromaticity coordinates are quite distinct from the red. It is also almost possible to distinguish between the greens of the Granny Smiths (GS1 and GS2) and the Golden Delicious (GD1 and GD2). One can also notice the trend, that the relative positions of the fruit colours remain in a similar pattern for each of the CWF and TL84 lighting conditions, under each of the viewing field angles. A viewing angle from 2^0 to 10^0 tends to shift the greens from lower x values to higher ones and from higher y values to lower ones, whereas the red values of the Starkings (ST1 and ST2) remain almost invariant to the viewing field angle. Note that the chromaticity coordinates of TL84 and CWF illuminants are also shown in Figure 3.4. They are very close on the x,y plane and hence similar in colour, however this fact does not show that TL84 has better colour rendering abilities than CWF. What does show that TL84 renders colour better than CWF is that all colours viewed under TL84 have a wider spread in the x and y directions, than those colours viewed under CWF lighting. From these observations one may conclude that viewing colours under TL84 at a small field angle (2^0) yields a maximum spread between greens and reds.

3.3. Comparative Results of Colour Systems Representing Six Positions on a Granny Smith Apple

This section compares seven colour system representations of the six colour positions shown in Figure 3.2. Colour representation using spectral power distribution curves is first presented. Next, the chromaticity coordinates for each of the six positions are compared. The remaining colour systems to be compared are calculated from the measured XYZ tristimulus values, and they are the: CIE 1960 UCS (u,v) coordinates; L*u*v* values; L*a*b* values; RGB values ;and HSI values. Each colour system is used to represent the positions under two types of lighting (CWF and TL84) and for two observation fields, namely the small field (2^0) and the large field (10^0).

3.3.1. Spectral Power Distribution Curves for Six Positions

Figure 3.5 shows the spectral curves for the six positions A,B,C,D,E and F on a Granny Smith apple (given in Figure 3.2).

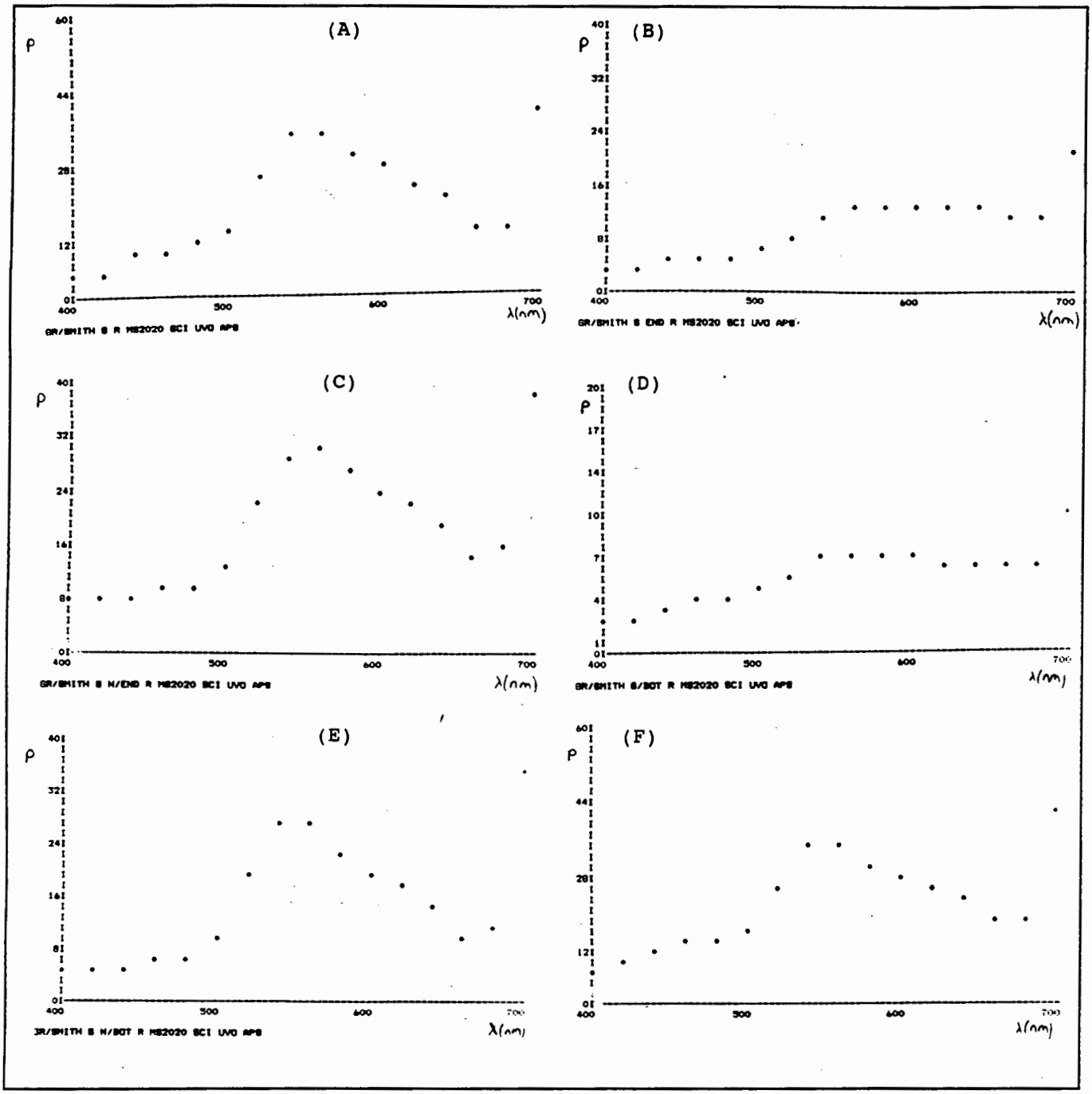


Figure 3.5: Spectral power distribution curves for the six positions: A, B, C, D, E and F on the Granny Smith apple shown in Figure 3.2.

Observations

The curves of Figure 3.5 indicate that the different positions on the fruit analyzed, have differing spectral components to one another. The curves show that positions A,C,E and F have power peaks at about 560nm, hence they are greener than positions B and D. A subtraction between any two curves from one another could be a method to find colour differences between two colours at every wavelength.

3.3.2. CIE 1931 Chromaticity Coordinates for Six Positions

Table 3.2 contains the XYZ tristimulus values and x,y chromaticity coordinates for the six positions on a Granny Smith apple. These values were calculated by normalizing the XYZ tristimulus values produced from the ICS Texicon colour analyzer. The calculation uses equations (2.1). Figure 3.6 uses this table of results to show how the six colour positions vary on the CIE 1931 chromaticity chart. The figure also shows how the lighting conditions and viewing fields shift the coordinates for the six positions.

Table 3.2: Tristimulus values and chromaticity coordinates for the six apple positions shown in Figure 3.2.

Lighting	Position	X	Y	Z	x	y
CWF_2	A	27.56	31.28	7.35	0.416	0.473
	B	12.31	12.46	3.55	0.435	0.440
	C	24.36	26.89	6.76	0.420	0.464
	D	6.87	7.18	2.56	0.414	0.432
	E	20.46	23.60	4.71	0.420	0.484
	F	28.38	31.41	9.38	0.410	0.454
CWF_10	A	28.97	30.51	7.28	0.434	0.457
	B	12.78	12.20	3.57	0.448	0.427
	C	25.53	26.25	6.76	0.436	0.448
	D	7.16	7.06	2.58	0.426	0.420
	E	21.55	22.95	4.67	0.438	0.467
	F	29.75	30.76	9.38	0.426	0.440
TL84_2	A	27.32	31.86	7.03	0.413	0.481
	B	12.55	12.41	3.40	0.443	0.438
	C	24.28	27.24	6.44	0.419	0.470
	D	6.96	7.20	2.46	0.419	0.433
	E	20.15	24.16	4.50	0.413	0.495
	F	28.30	31.86	8.97	0.409	0.461
TL84_10	A	28.49	31.07	6.96	0.428	0.467
	B	12.86	12.18	3.41	0.452	0.428
	C	25.22	26.59	6.43	0.433	0.457
	D	7.17	7.09	2.46	0.429	0.424
	E	21.08	23.48	4.45	0.430	0.479
	F	29.41	31.19	8.96	0.423	0.448

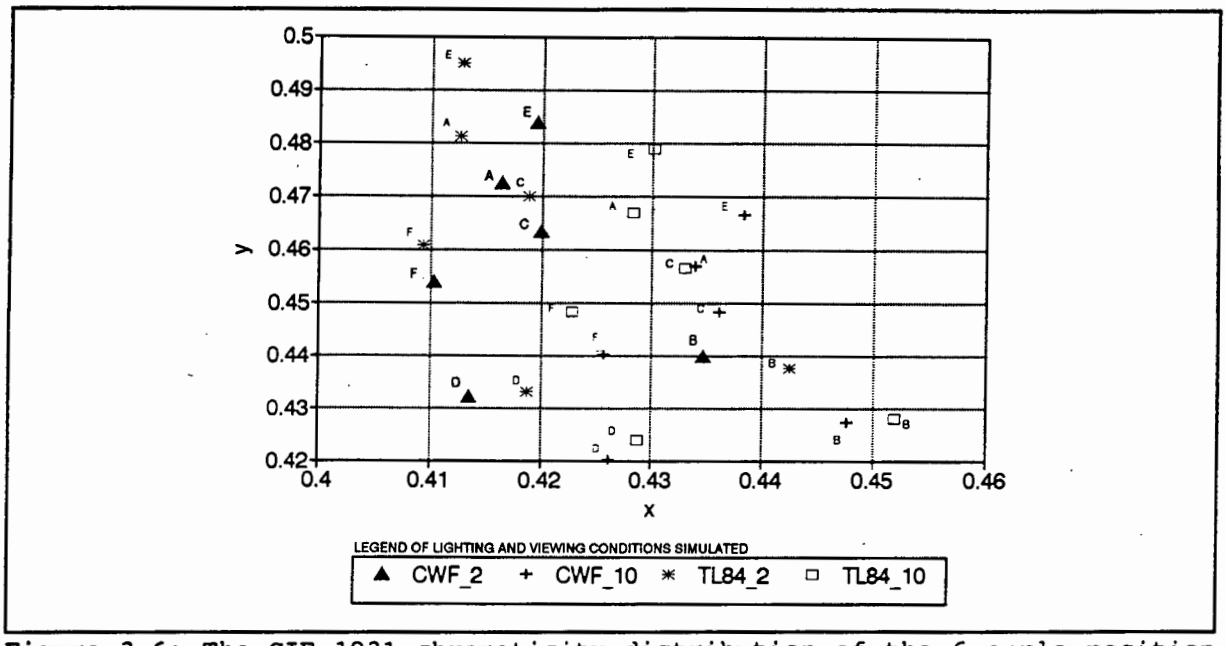


Figure 3.6: The CIE 1931 chromaticity distribution of the 6 apple position colours: A, B, C, D, E and F given in table 3.2, for CWF and TL84 lighting at 2^0 and 10^0 field observations.

Observations

From Figure 3.6 it appears that the relative distribution pattern between the six positions is mostly independent of the lighting conditions. As observed in section 3.2.2, a viewing angle from 2^0 to 10^0 tends to shift the greens from lower to higher x values and from higher to lower y values. Also the relative coordinate position spacing between the six colours tends to increase from the CWF to the TL84 lighting conditions. However, it is possible to deduce by observing the six colour positions on the chromaticity chart, that the colour at E is greenest and that at B is reddest (or brownest), independent of the lighting and viewing conditions. It is difficult to perceptually compare the colours shown at F, A, C, and D using the x,y coordinates.

3.3.3. CIE 1960 Uniform Chromaticity Scale (UCS) Coordinates for Six Positions

The CIE 1960 UCS coordinates (u,v) are a transformed version of the CIE 1931 chromaticity coordinates (x,y). The transformation is given in equation (3.2) (Wyszecki and Stiles (1967)).

$$u = \frac{4x}{-2x+12y+3} \qquad v = \frac{6y}{-2x+12y+3} \tag{3.2}$$

This transformation is aimed at providing a more uniform distribution of colours, by reducing the large green section of the CIE chromaticity chart (Figure 2.5) and enlarging the small yellow and red sections of the chart. Table 3.3 gives the results of this transformation, and Figure 3.7 shows how the six colour positions vary on the CIE 1960 UCS chart.

Table 3.3: The CIE 1960 UCS chromaticity coordinates for the six apple positions shown in Figure 3.2.

Lighting	Position	u	v	Lighting	Position	u	v
CWF_2	A	0.212	0.362	TL84_2	A	0.208	0.363
	B	0.235	0.356		B	0.240	0.356
	C	0.218	0.360		C	0.215	0.361
	D	0.225	0.352		D	0.228	0.353
	E	0.211	0.364		E	0.204	0.366
	F	0.215	0.357		F	0.212	0.359
CWF_10	A	0.228	0.360	TL84_10	A	0.221	0.362
	B	0.248	0.354		B	0.250	0.355
	C	0.232	0.358		C	0.228	0.360
	D	0.237	0.351		D	0.237	0.352
	E	0.227	0.363		E	0.218	0.364
	F	0.229	0.355		F	0.224	0.357

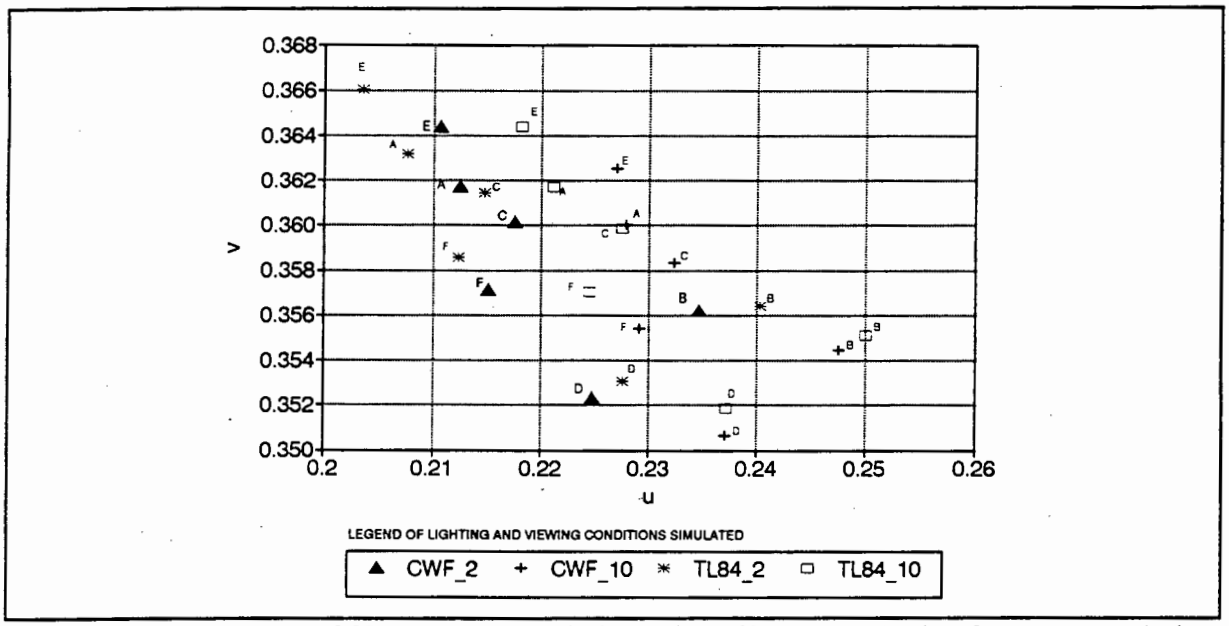


Figure 3.7: The CIE 1960 UCS chromaticity distribution of the 6 apple position colours: A, B, C, D, E and F given in table 3.3, for CWF and TL84 lighting at 2^0 and 10^0 field observations.

Observations

From Figure 3.7 it can be seen that the different lighting conditions and viewing fields shift the six green colours across the u,v plane. For example, position A moves from a higher to a lower v value and a lower to a higher u value from TL84 to CWF and from 2^0 to 10^0 observation fields respectively. However, the relative positions between the colours remain in the same pattern. As u increases colours move from greens to reds. This single variable is able to show that from position E to A to F the colours get less green and more yellow, and the russet part at C lies more yellow than the brown parts of the base (D) and stem (B) of the apple. The v values vary slightly between the colours compared with the u values, since an increasing v shows the change of colours from blue to yellow.

3.3.4. The L*u*v* and L*a*b* Colour Coordinates for Six Positions

Two colours represented in the L*a*b* and L*u*v* systems are compared by measuring the Euclidean distances between them. For the L*u*v* system this can be given as (MacAdam (1981)):

$$\Delta E(L*,u*,v*) = \sqrt{(\Delta L*)^2+(\Delta u*)^2+(\Delta v*)^2} \tag{3.3}$$

where

$$L* = 116(Y/Y_0)^{1/3}-16$$

$$u* = 13L*\left(\frac{4X}{X+15Y+3Z}-\frac{4X_0}{X_0+15Y_0+3Z_0}\right) \tag{3.4}$$

$$v* = 13L*\left(\frac{9Y}{X+15Y+3Z}-\frac{9Y_0}{X_0+15Y_0+3Z_0}\right)$$

The constants X_0, Y_0 and Z_0 are the tristimulus values corresponding to the standard illuminating source. Since the ICS Texicon manuals available did not provide such values for the CWF and TL84 lighting sources, an estimated calculation was made for the values[1].

[1] The calculation used the values of L*, a*, b*, X, Y and Z which were given by the ICS Texicon colour analyzer. By using the L*a*b* formula and these values for colour samples under CWF, and TL84 lighting, the corresponding X_0, Y_0, and Z_0 values could be derived for the respective illuminant.

These values are estimated as:

$X_0 = 132.97 \quad Y_0 = 128.64 \quad Z_0 = 89.63$ for CWF and

$X_0 = 134.00 \quad Y_0 = 128.96 \quad Z_0 = 85.87$ for TL84.

The variables X,Y and Z are the tristimulus values of the colour samples, illuminated by the same source. The differences between their values of L*, u*, and v* are designated ΔL*, Δu*, and Δv*.

In terms of the CIE Luv formula, the metric hue angle is given by:

$$H_{uv}^0 = \tan^{-1}(v*/u*) \tag{3.5}$$

The L*a*b* formula for colour differences is written in terms of the same constants X_0, Y_0, Z_0, variables X, Y, Z and L*,

$$\Delta E(L*,a*,b*) = \sqrt{(\Delta L*)^2+(\Delta a*)^2+(\Delta b*)^2} \tag{3.6}$$

where

$$\begin{aligned} a* &= 500\left[\left(\frac{X}{X_0}\right)^{1/3} - \left(\frac{Y}{Y_0}\right)^{1/3}\right] \\ b* &= 200\left[\left(\frac{Y}{Y_0}\right)^{1/3} - \left(\frac{Z}{Z_0}\right)^{1/3}\right] \end{aligned} \tag{3.7}$$

In terms of the CIE Lab formula, the metric hue angle is:

$$H_{ab}^0 = \tan^{-1}(b*/a*) \tag{3.8}$$

Table 3.4 gives the L*, u*, v*, H_{uv}^0, a*, b*, and H_{ab}^0 values for the six apple positions under the 2 different types of lighting and viewing field conditions. Figures 3.8 and 3.9 give the corresponding positions of the values in the u*, v* plane and the a*, b* plane respectively.

Table 3.4: The L*u*v*, L*a*b* and corresponding hue values for the six apple positions shown in Figure 3.2

Lighting	Position	L*	u*	v*	a*	b*	H^0_{uv}	H^0_{ab}
CWF-2	A	56.40	-11.52	33.78	-16.21	37.95	-71.18	-66.87
	B	37.27	3.12	18.32	-3.46	23.68	80.33	-81.69
	C	52.84	-7.34	29.99	-12.80	34.20	-76.24	-69.49
	D	28.33	-1.26	11.80	-4.88	15.30	-83.93	-72.32
	E	49.91	-11.41	32.47	-16.21	38.74	-70.64	-67.30
	F	56.50	-9.59	28.78	-13.74	30.77	-71.57	-65.94
CWF-10	A	55.80	-0.21	31.55	-8.67	37.20	-89.62	-76.88
	B	36.90	9.29	16.88	0.99	22.91	61.17	87.52
	C	52.29	2.81	27.82	-5.95	33.26	84.24	-79.86
	D	28.08	3.25	10.75	-1.22	14.72	73.19	-85.25
	E	49.30	-0.79	30.30	-8.90	37.90	-88.50	-76.79
	F	56.00	0.70	26.62	-6.84	29.90	88.49	-77.12
TL84-2	A	56.79	-16.82	33.84	-19.45	38.66	-63.57	-63.29
	B	37.16	4.78	17.24	-2.06	23.49	74.51	-85.00
	C	53.08	-10.80	29.79	-14.83	34.77	-70.08	-66.90
	D	28.34	-1.05	11.31	-4.54	15.25	-84.69	-73.41
	E	50.38	-17.62	32.77	-20.21	39.60	-61.74	-62.97
	F	56.79	-13.34	28.71	-15.97	31.31	-65.07	-62.97
TL84-10	A	56.18	-6.80	31.81	-12.69	37.90	-77.93	-71.49
	B	36.83	9.36	16.14	1.22	22.85	59.89	86.94
	C	52.53	-1.97	27.86	-8.84	33.86	-85.96	-75.37
	D	28.11	2.49	10.53	-1.71	14.86	76.71	-83.45
	E	49.75	-7.97	30.78	-13.47	38.80	-75.48	-70.85
	F	56.27	-4.37	26.77	-9.91	30.46	-80.73	-71.97

Observations

The u*v* and a*b* planes, which are modifications of the CIE 1960 UCS u,v plane, illustrate more clearly what was observed in the previous section on the u,v plane. There is still the trend that a 2^0 to a 10^0 field view shifts the colours to a redder tinge (i.e. in this case by shifting points from lower to higher u* or a* values and from higher to lower v* or b* values respectively). Also, in these two colour spaces the colours observed under TL84 lighting give a wider spread than those viewed under CWF lighting. For most conditions the single variable u* or a* shows that E is greenest, and from A to F to C colours get more yellow and that at D the colour is less brown than B.

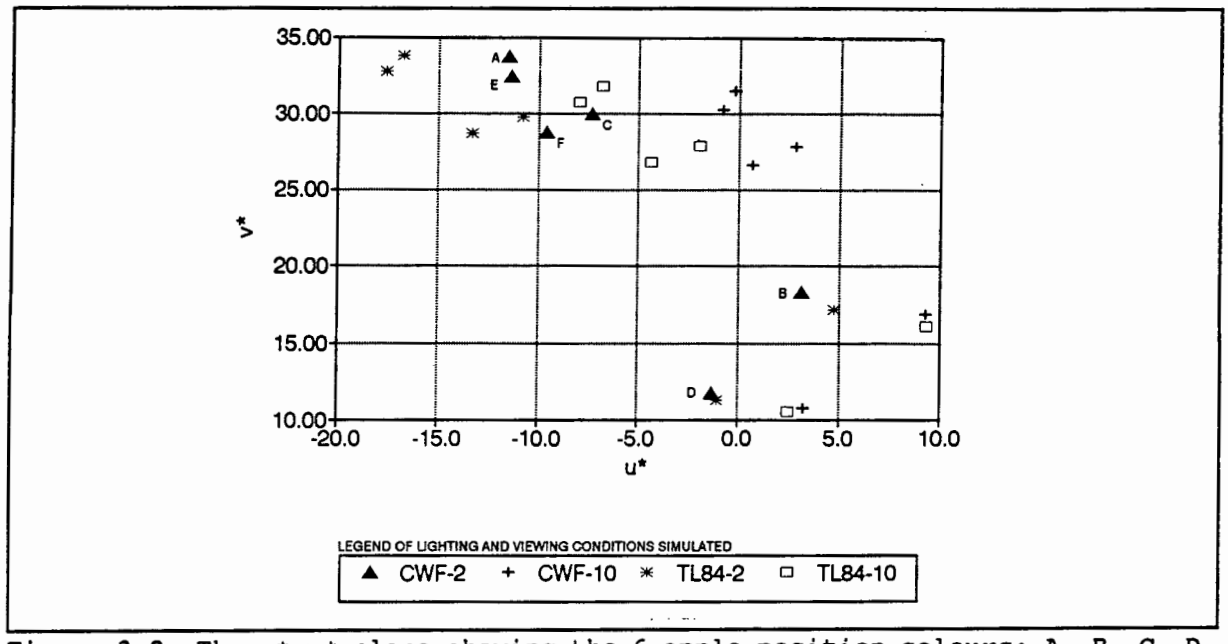


Figure 3.8: The u*,v* plane showing the 6 apple position colours: A, B, C, D, E and F given in table 3.4, for CWF and TL84 lighting at 2^0 and 10^0 field observations.

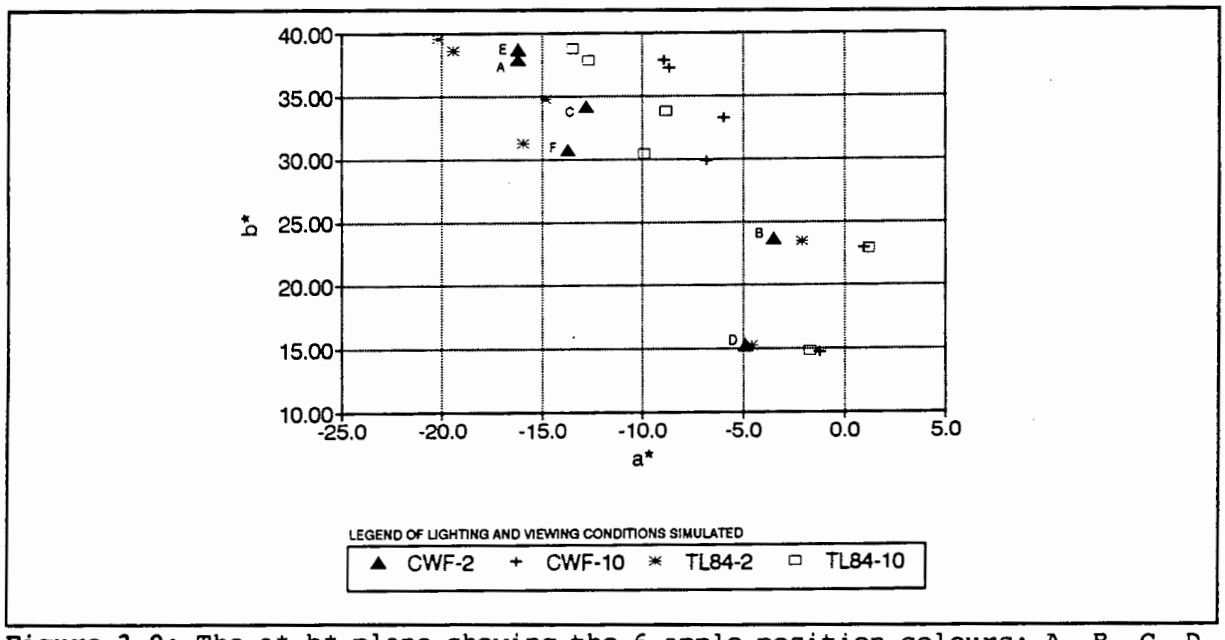


Figure 3.9: The a*,b* plane showing the 6 apple position colours: A, B, C, D, E and F given in table 3.4, for CWF and TL84 lighting at 2^0 and 10^0 field observations.

However, the H^0_{uv} or H^0_{ab} values shown in table 3.4 give a more accurate account of the relative colour positions. This is because, for example, u* will detect that A is greener than E in the CWF_2^0 condition which is not true, but by H^0_{uv} for A being less than H^0_{uv} for E, the H^0_{uv} value correctly accounts for E being greener than A. The maximum hue angle difference is between positions E and B under condition TL84_10^0. This angle difference is 44.7^0 in the u*v* plane and only 22.2^0 in the a*b* plane. In general the spread between colour positions

is much larger on the u*v* plane, than on the a*b* plane. This could be because the u* and v* formulae are dependent on the luminance value L* whereas a* and b* are independent of L*. Table 3.4 provides the information that with the L* variable, lightness increases from positions D, B, E, C, A to F, which is perceptually correct. A point to notice is that the u*, v*, a* and b* values all account for the lighting sources tristimulus values X_0, Y_0 and Z_0 in their formulae. It can be seen, however, that this fact does not make the u*, v*, a*, and b* values eliminate the effects of the different lighting sources.

3.3.5. The RGB Values for Six Positions

RGB values are those used to represent each pixel in an image displayed on a colour monitor. However, one can transform to and from the XYZ tristimulus values and RGB values, if the chromaticities of each of the Red, Green, and Blue phosphors are known. For example, the chromaticities of the NTSC colour television system standard (and also the PAL system which is derived from NTSC) are: $x_R = 0.67 \quad y_R = 0.33$

$$x_G = 0.21 \quad y_G = 0.71$$

$$x_B = 0.14 \quad y_B = 0.08$$

The luminance signal Y is formed from the outputs of the camera corresponding to the red, green and blue components of the scene and is equivalent to their sum in the following proportions (PAL Colour Tv (1967)):

$$Y = 0.31R + 0.59G + 0.11B$$

Since by definition $x/X = y/Y = z/Z,$

hence $X = (x/y)Y \quad \text{and} \quad Z = (z/y)Y$

If the chromaticity ratios are defined as: $\kappa = x/y$ and $\xi = z/y,$

then

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} \kappa_R & \kappa_G & \kappa_B \\ 0.31 & 0.59 & 0.11 \\ \xi_R & \xi_G & \xi_B \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{3.9}$$

hence by matrix inversion and substitution of the RGB phosphor chromaticities into the above equation, the RGB values can be given as:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 0.62 & 0.17 & 0.18 \\ 0.31 & 0.59 & 0.11 \\ 0 & 0.066 & 1.02 \end{pmatrix}^{-1} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{3.10}$$

One is mainly interested in observing the relative positions of the RGB represented colours. From the above equations, each primary is dependent on the magnitudes of the X, Y and Z values. Since the magnitudes of the XYZ values are quite arbitrary, the RGB values of table 3.5 would not be a true representation of the RGB values for a 24 bit colour monitor. A multiplying factor (of say 2) would be needed to raise the RGB table values to a suitable level of colour representation on an RGB monitor. Table 3.5 gives the R, G and B values for the six positions on the Granny Smith apple viewed under CWF and TL84 lighting at 2^0 and 10^0 observation fields. Figure 3.10 shows the values of table 3.5 (not corrected for RGB monitor display) on the GR plane and Figure 3.11 shows the values on the GB plane.

Table 3.5: The R, G, and B values for the six apple positions shown in Figure 3.2.

Lighting	Position	R	G	B
CWF-2	A	33.56	34.46	4.98
	B	15.68	12.38	2.68
	C	29.97	28.94	4.75
	D	8.48	7.34	2.04
	E	24.93	26.36	2.91
	F	34.47	33.82	7.01
CWF-10	A	36.60	31.53	5.10
	B	16.69	11.40	2.76
	C	32.48	26.51	4.91
	D	9.08	6.81	2.09
	E	27.30	23.99	3.03
	F	37.35	31.17	7.18
TL84-2	A	32.91	35.86	4.57
	B	16.20	12.05	2.55
	C	29.74	29.73	4.39
	D	8.66	7.29	1.94
	E	24.12	27.79	2.61
	F	34.20	34.81	6.54
TL84-10	A	35.51	33.13	4.68
	B	16.89	11.28	2.61
	C	31.82	27.50	4.52
	D	9.11	6.86	1.97
	E	26.22	25.52	2.71
	F	36.62	32.38	6.69

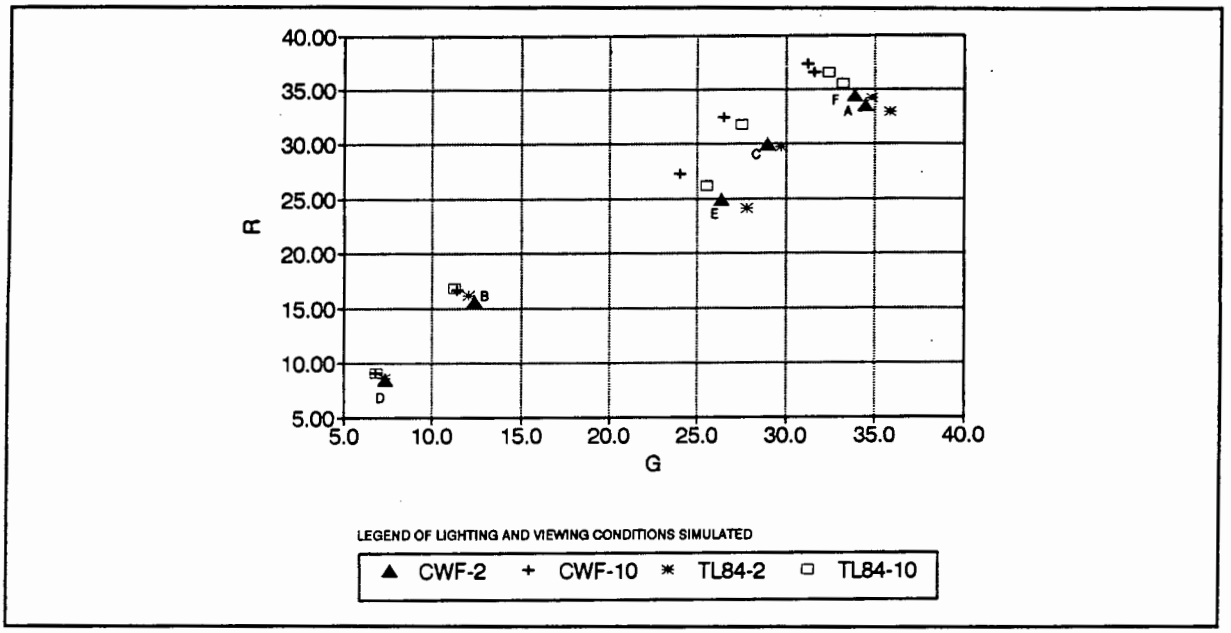


Figure 3.10: The R,G plane showing the 6 apple position colours: A, B, C, D, E and F given in table 3.5, for CWF and TL84 lighting at 2^0 and 10^0 field observations.

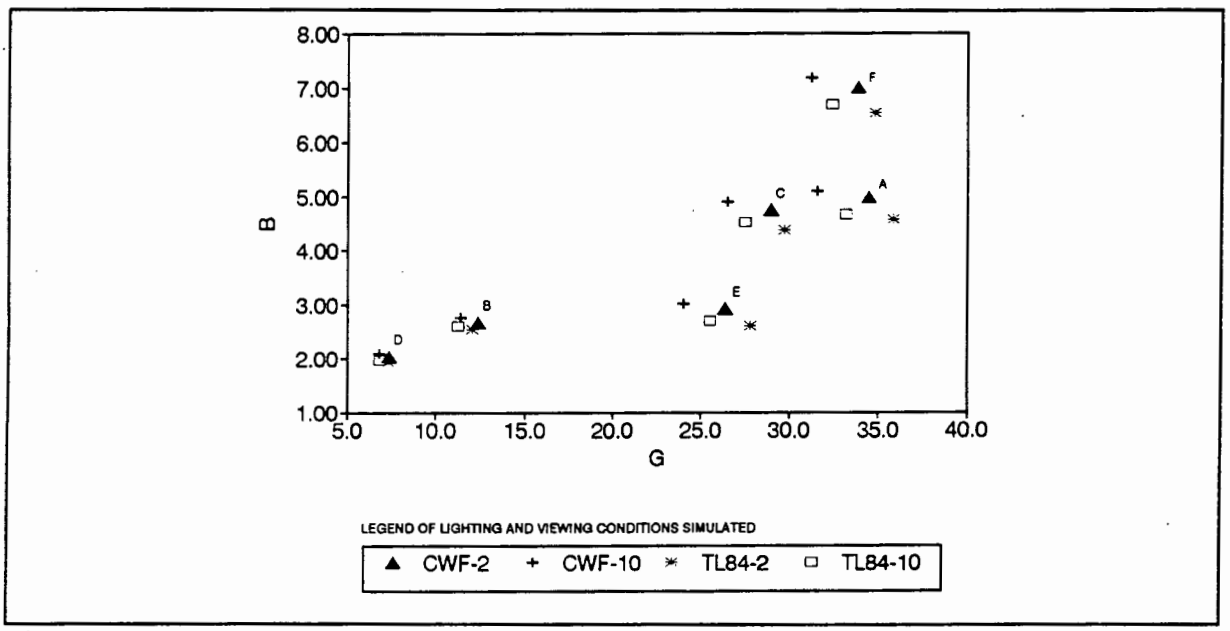


Figure 3.11: The B,G plane showing the 6 apple position colours: A, B, C, D, E and F given in table 3.5, for CWF and TL84 lighting at 2^0 and 10^0 field observations.

Observations

As in the previous colour space observations, the relative colour positions maintain the same pattern under the different lighting and viewing field conditions. The 2^0 and TL84 illuminated colour positions both tend to have less red, more green and slightly less blue components than the 10^0 field and CWF illuminated observed positions respectively. This is to be expected

since the TL84 spectral power curve has a large peak at the green wavelength (Burke (1992)).

It is difficult to perceptually realize the colour positions using only one primary variable. For example, the green variable G suggests position A is greenest and D is the least green, which is not true in terms of actual colour. However, the R and G components together suggest that E has the most green component over red, i.e. the largest green/red ratio, compared with any other position, hence E must be the greenest. Similarly by using the green/red ratio the remaining positions can be correctly identified. If one uses the green/blue ratio, the positions can (as with green and red) be correctly identified perceptually.

3.3.6. HSI Values for Six Positions

In chapter 2 a matrix transformation was given (equation (2.2)) that used Niblack's (1986) formula to convert from RGB to HSI. In this experiment Lehar and Stevens' (1984) matrix transformation is used, as this gave a regular hexagon of colour distribution in the H,S plane, whereas Niblack's was slightly irregular. Chapter 4 will illustrate in more detail the differences between the various HSI derivations. The vectors m1 and m2 (which make up the hue (H) and saturation (S) values) and the intensity (I) are derived from:

$$\begin{pmatrix} I \\ m1 \\ m2 \end{pmatrix} = \begin{pmatrix} 1/3 & 1/3 & 1/3 \\ 2/\sqrt{6} & -1/\sqrt{6} & -1/\sqrt{6} \\ 0 & 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{3.11}$$

where, after amendments to Lehar and Stevens (1984) incorrect definitions:

$$H = \tan^{-1}(m1/m2)$$
$$S = \sqrt{m1^2+m2^2} \tag{3.12}$$

In the observations stated in section 3.3.5 for RGB values, it was noted that the green/red ratio or green/blue ratio, was capable of perceptually classifying the six colour positions. This HSI system in fact makes it easier to compare RGB components by rotating the RGB cube with the above transformation. The Hue value, in this case, now does the task of comparing the amount green over both the red and blue components simultaneously.

Table 3.6 gives the m1, m2, H, S and I values for the six positions on the Granny Smith apple observed under CWF and TL84 lighting and at 2^0 and 10^0 observation field angles. Figure 3.12 illustrates how the colour positions vary on the H, S plane by using the m1 and m2 coordinates.

Table 3.6: The vectors m1 and m2 and derived HSI values for the six apple positions of Figure 3.2.

Lighting	Position	m2	m1	H	S	I
CWF_2	A	20.85	11.30	28.47	23.71	24.33
	B	6.86	6.66	44.15	9.56	10.25
	C	17.10	10.72	32.08	20.18	21.22
	D	3.75	3.10	39.57	4.86	5.95
	E	16.58	8.40	26.87	18.59	18.07
	F	18.96	11.47	31.18	22.16	25.10
CWF_10	A	18.69	14.93	38.62	23.92	24.41
	B	6.10	7.84	52.11	9.94	10.28
	C	15.27	13.69	41.89	20.51	21.30
	D	3.34	3.78	48.54	5.04	5.99
	E	14.82	11.26	37.23	18.62	18.11
	F	16.96	14.84	41.18	22.54	25.23
TL84_2	A	22.12	10.36	25.10	24.43	24.45
	B	6.71	7.26	47.26	9.89	10.27
	C	17.92	10.35	30.02	20.69	21.28
	D	3.78	3.31	41.15	5.02	5.96
	E	17.80	7.28	22.26	19.23	18.17
	F	19.99	11.04	28.92	22.84	25.18
TL84_10	A	20.12	13.56	33.97	24.26	24.44
	B	6.13	8.12	52.94	10.17	10.26
	C	16.25	12.91	38.46	20.75	21.28
	D	3.46	3.83	47.93	5.17	5.98
	E	16.13	9.88	31.50	18.91	18.15
	F	18.17	13.95	37.52	22.90	25.23

Observations

Figure 3.12 shows that a 2^0 field gives values that are larger in m1 and smaller in m2, than 10^0 field view values (this is similar to the previous colour space observations). Colour positions are more spread out over the m1m2 plane when viewed under TL84 lighting compared with those values viewed under CWF lighting. Neither m1 nor m2 singularly

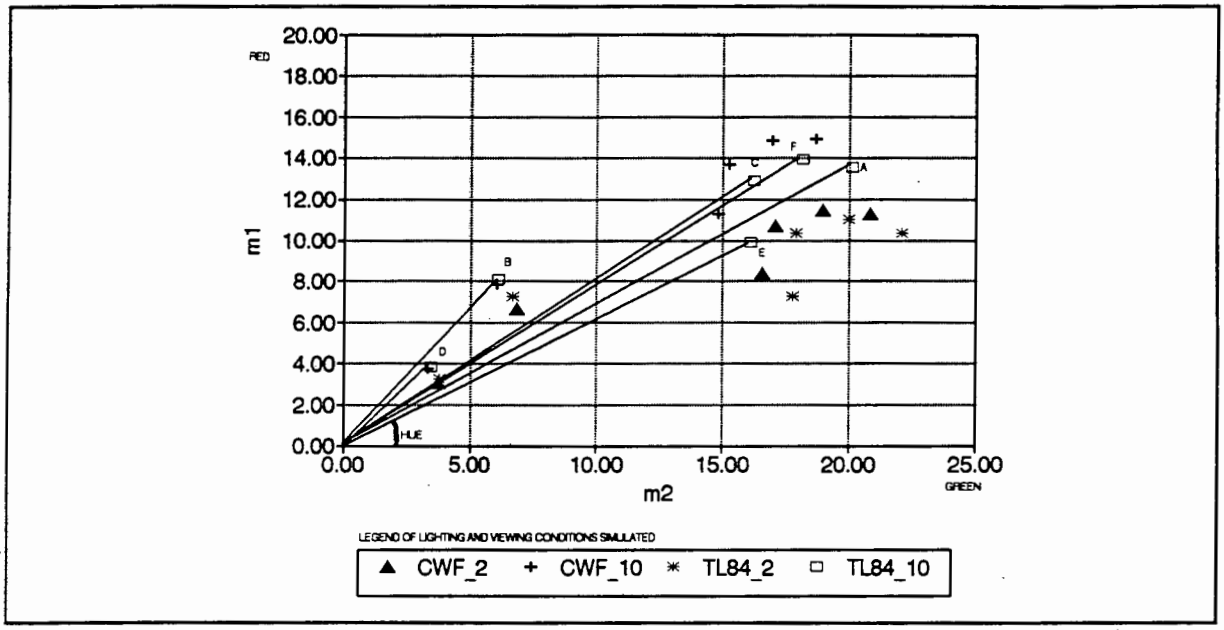


Figure 3.12: The m1,m2 plane shows the 6 apple position colours given in table 3.6, for CWF and TL84 lighting at 2^0 and 10^0 fields. The radial lines produce hue angles with m2.

describe the colour positions, however, the hue value gives a sufficiently accurate account that position E is greenest and from A, F, C, D, to B the colour gets less green and more brown. From table 3.6 the maximum hue angle difference of 21.4^0 is found between positions E and B under condition TL84_10^0. Saturation, which is the length of the radial lines shown in Figure 3.12, shows the purity of each colour position, with position A the most pure (or saturated) and D the least. The intensity value I in table 3.6, as with L* in table 3.4 suggests that lightness increases from positions D, B, E, C, A to F. The perceptual values of H, S and I thus agree with the human ability to perceptually distinguish between similar colours.

3.4. Summary and Discussion of the Results Shown by The Various Colour Systems

Figure 3.13 is a chart that shows the relationships between all the colour systems discussed in chapter 2 and in this chapter. One can see that colours are represented by spectral distributions using a spectrophotometer instrument. The XYZ tristimulus values are usually the result of colour matching using a colorimeter, and the RGB colour components are the output of a colour RGB video camera (or colour CCD). Each connecting line in the figure represents a mathematical operation that must be carried out in order to achieve the required colour system.

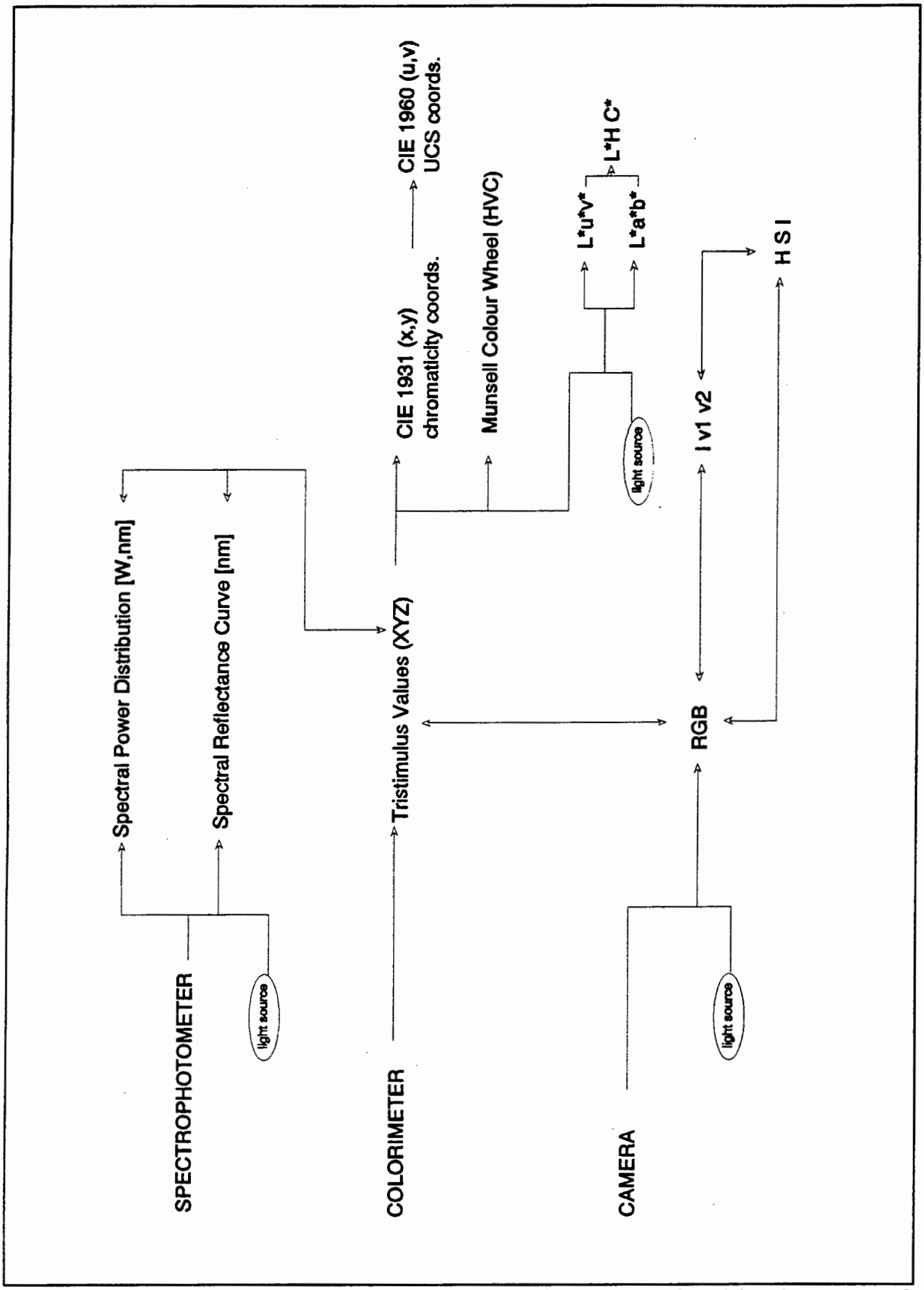


Figure 3.13: Summary diagram representing the inter-relationships between all colour systems discussed. Each connecting line represents a mathematical transition.

Lighting conditions and viewing angles shift the values of observed colour positions, however, the pattern of their positions relative to each other remains almost invariant to such changing conditions. A lighting system that allows for good colour rendering (TL84), as opposed to one with bad colour rendering (CWF), will increase the relative separation between values on the same colour space plane. A 2^0 viewing angle, as opposed to a 10^0 field tends to allow the colour positions to become more green in appearance. This can be explained by the photopic response curve of the eye (Figure 1.1), with a 2^0 field giving a larger peak at the green wavelengths than that of a 10^0 field.

All the colour representation systems used in the two experiments show that each is capable of uniquely distinguishing colours. In the case of spectral curves, one variable - the spectral power, is needed to describe a single colour at all visible wavelengths. The other colour spaces require at most, two variables to distinguish between colours. Some colour spaces like the CIE 1960 UCS (u,v), the L*u*v* and the L*a*b* systems could distinguish between green and red with only one variable, namely the u, u*, or a* value respectively. The L*u*v*, L*a*b* and HSI colour spaces could enable one to recognise any colour using the single variable of hue. Lightness or intensity and saturation were further variables that could be used to adequately distinguish between colours as humans perceive the colours.

From the experimental results given in this chapter any of the colour systems described could be used for the proposed automated fruit sorter. The next chapter will give reasons why the HSI perceptual colour system was chosen for the automatic fruit sorting task. The chapter will also give a comparison between the various RGB to HSI transformations, and the development of a new HSI system based on previous transformations, will be shown. The new HSI system is aimed at being ideal for the automatic fruit sorting device.

CHAPTER 4

Further Experimentation With The HSI Colour Systems

4.1. Introduction

In chapter 3 experimental results showed that generally any of the colour systems described, are capable of distinguishing similar colours (namely the greens on an apple) and colours which are quite different (namely red and green apples). It was also found that the better colour spaces were those which produced a hue value, since these spaces allowed for a perceptual understanding how the actual values related to the colours they represented. The L*a*b* and the L*u*v* colour spaces were among the best systems for representing similar greens, since their hue values covered a large range. However, the HSI system was equally capable of distinguishing the greens even though the hue range was narrower. The HSI system was chosen for further experimentation, with the intention of using it in the eventual hardware that will be used to automatically sort fruit. The following are reasons why the HSI system was chosen over the other colour systems:

1) The HSI system is able to distinguish between similar colours;

2) The operations to convert from RGB to HSI involve simple multiplications of values to constants (in a matrix or equation form). If each of the H, S, and I values are represented by a byte (i.e. 256 different levels for each value) then the time to convert from RGB to HSI is at its minimum, since most floating point operations are removed. Systems such as L*a*b* and L*u*v* require a conversion from RGB to XYZ and then, eventually a conversion to their system values, using cube root operations as well. This means transformations to such colour systems are slower than to the HSI system. Speed is a major criterion for automated sorting in real time, hence the HSI system would be a more viable colour system to use.

3) The HSI system stems from the human ability to describe colour perceptually, thus automated fruit sorting can be made to simulate how humans sort fruit.

4) The HSI system is not rigid, hence by manipulating the matrix values for the transformation, certain colour regions can be made larger than others. This is good if say the green hue range needs to be made more spread out.

The aim of this chapter is to formally compare various RGB to HSI transformations, with the intention of using the best HSI system in an experiment to represent fruit colours. This chapter also intends to illustrate fruit skin colours represented by various combinations of the H, S and I components. This chapter begins by describing the theory involved in the several RGB to HSI transformations. The experimental set up for capturing and displaying the colour scenes is then described. From the theory several colour feature components are chosen to represent three apples in a colour analysis experiment. The aim of the experiment was to find the fewest and the best features that best describe the colour variations on the fruits. A final overview and discussion of the experimental results concludes this chapter.

4.2. The Various RGB to HSI Transformations Used for Experimentation

From the literature researched there has been no formal comparison between the various methods of converting colours represented by RGB to the perceptual HSI systems. It appears that if a certain colour transformation produced adequate results then no other transformation was sought out to yield a better result.

In this section several RGB to HSI transformations will be compared, from which one will be derived for use in further experimental colour feature analysis. There are seven RGB to HSI transformations used in this section which are based on those given by Niblack (1986), Ramirez (1986), Lehar and Stevens (1984), Benson (1987), and Morrissey-Golas (1989). These transformations were introduced in section 2.3.5. The transformation equations given below were used in a software program to produce the colour figures given in this section (Function drawdisc() in the program listing given in Appendix A). The program calculated and plotted the v1 and v2 components for all combinations of the 256 levels of the R, G and B colour values. The result was an H-S plane with intensity increasing perpendicularly from the plane outwards (see Figure 2.12a).

For the following transformations the definitions of H and S are common, where:

$$H = \tan^{-1}\left(\frac{v2}{v1}\right)$$

$$S = \sqrt{v1^2+v2^2} \qquad (4.1)$$

Were H is in the range 0^0 to 360^0. Hue (H) and Saturation (S) are in fact the polar coordinates of the vector components v1 and v2. The vector components and the intensity I are the variables that are shaped according to different linear combinations of the primary colours (R, G, and B). The seven RGB to HSI matrix transformations are given below.

From Niblack (1986):

$$\begin{pmatrix} I \\ v1 \\ v2 \end{pmatrix} = \begin{pmatrix} 1/3 & 1/3 & 1/3 \\ -1/\sqrt{6} & -1/\sqrt{6} & 2/\sqrt{6} \\ 1/\sqrt{6} & -2/\sqrt{6} & 0 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{4.2}$$

Figure 4.1 shows the v1 and v2 components of this transformation forming the irregular hexagonal shape of the H-S plane.

From Ramirez in Niblack (1986):

$$\begin{pmatrix} I \\ v1 \\ v2 \end{pmatrix} = \begin{pmatrix} .3 & .59 & .11 \\ -.105465 & -.207424 & .312889 \\ .445942 & -.445942 & 0 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{4.3}$$

Figure 4.2 shows the v1 and v2 components of this transformation forming a version of the H-S plane that is aimed at making the intensity perceptually more uniform over such a plane. This transformation tilts that of Figure 4.1 to try to achieve a uniform intensity over the H-S plane. According to Niblack Figures 4.1 and 4.2 show that the tilted plane version more nearly corresponds to the idea of constant intensity. The difference between calculating I as in (4.2) compared with (4.3) will be shown more clearly in section 4.4.

From Benson (1986) in Slaughter and Harrell (1987):

$$\begin{pmatrix} I \\ v1 \\ v2 \end{pmatrix} = \begin{pmatrix} .299 & .587 & .114 \\ .596 & -.275 & -.321 \\ .212 & -.523 & .311 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{4.4}$$

Figure 4.3 shows the v1 and v2 components of this transformation forming an irregular hexagonal H-S plane with the intensity component aimed at being perceptually uniform over the plane.

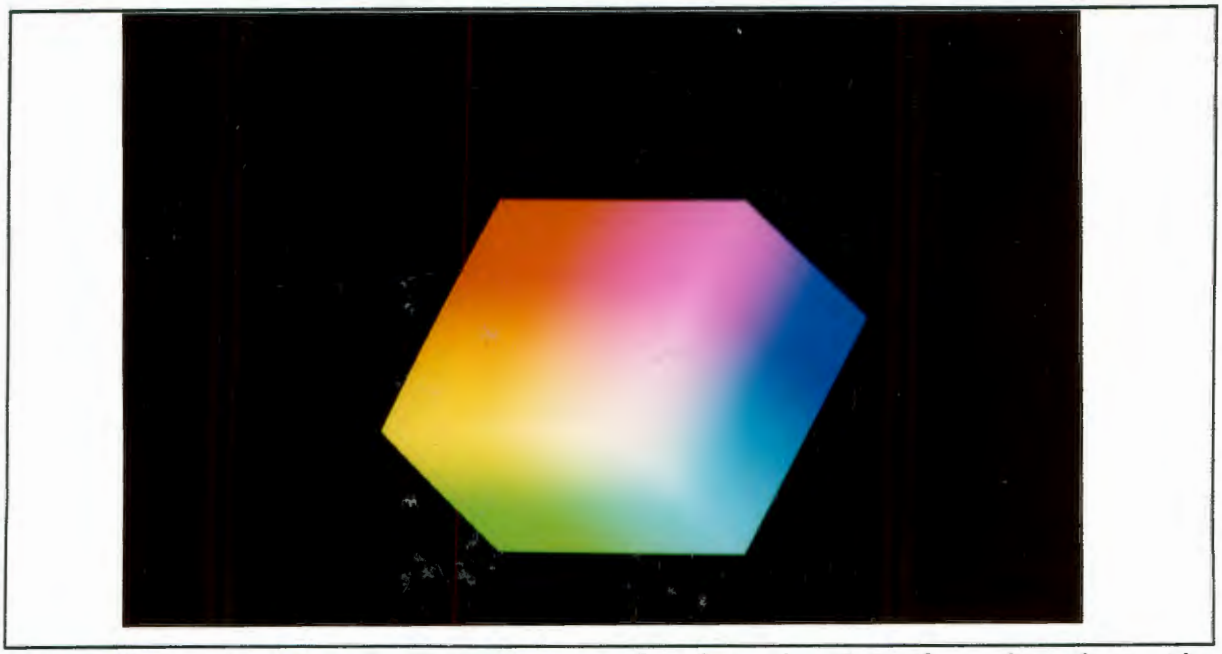


Figure 4.1: The v1 and v2 components forming the H-S plane based on the transformation given by Niblack (1986).

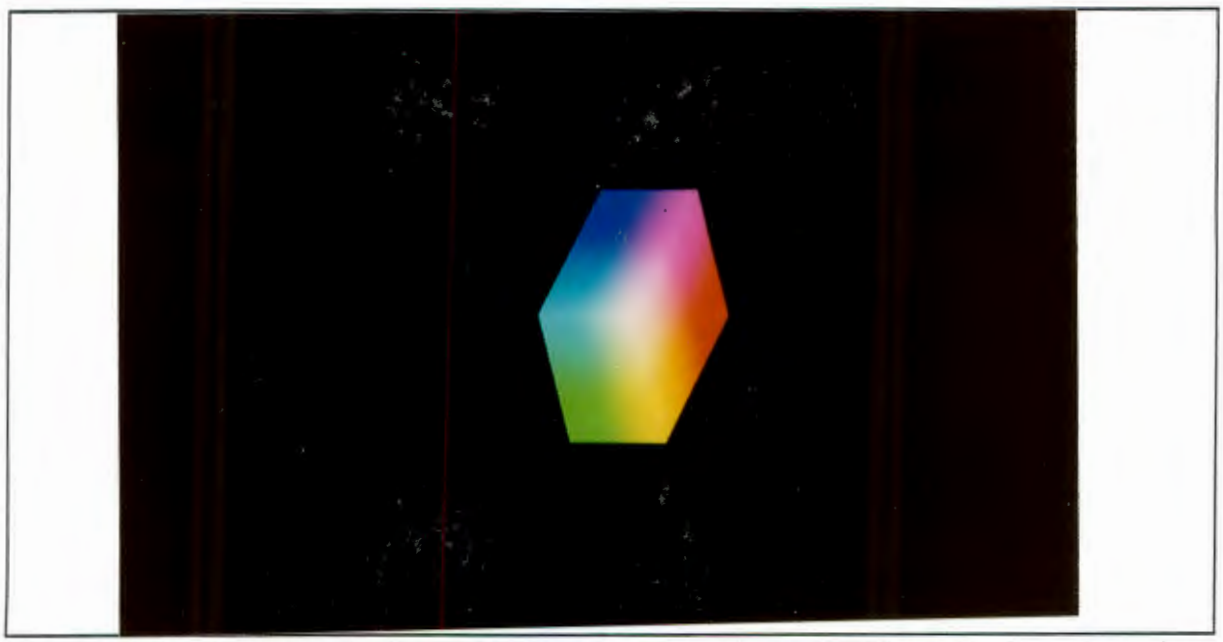


Figure 4.2: The v1 and v2 components forming the 'tilted' H-S plane based on the transformation given by Ramirez in Niblack (1986).

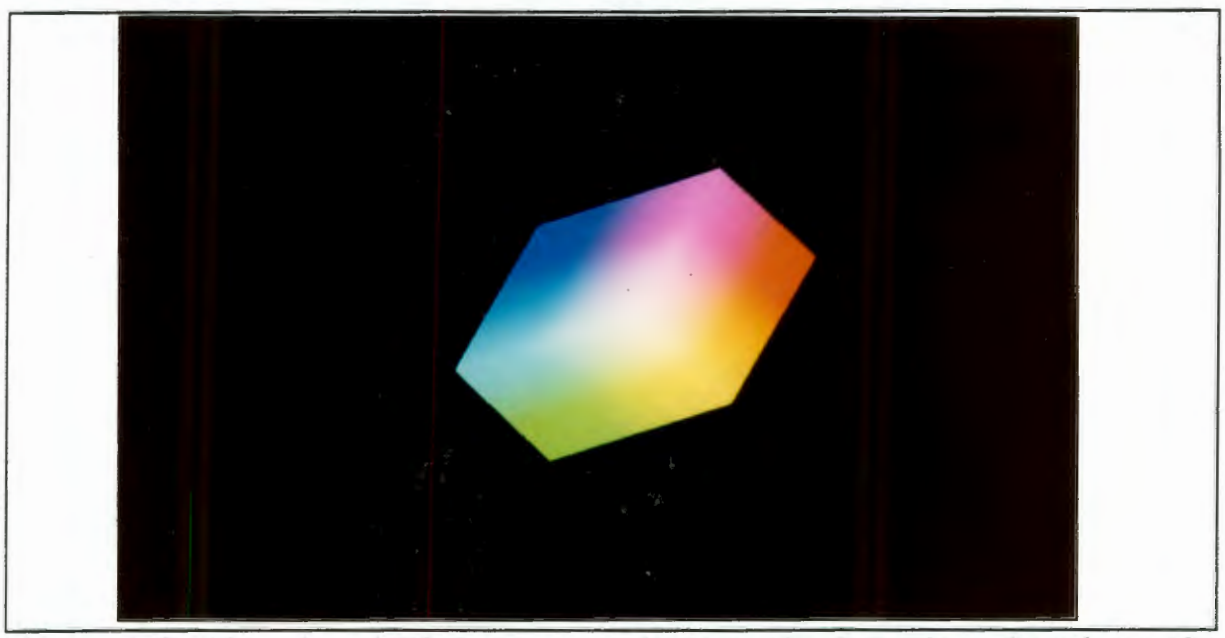


Figure 4.3: The v1 and v2 components forming the H-S plane based on the transformation given by Benson in Slaughter and Harrell (1987).

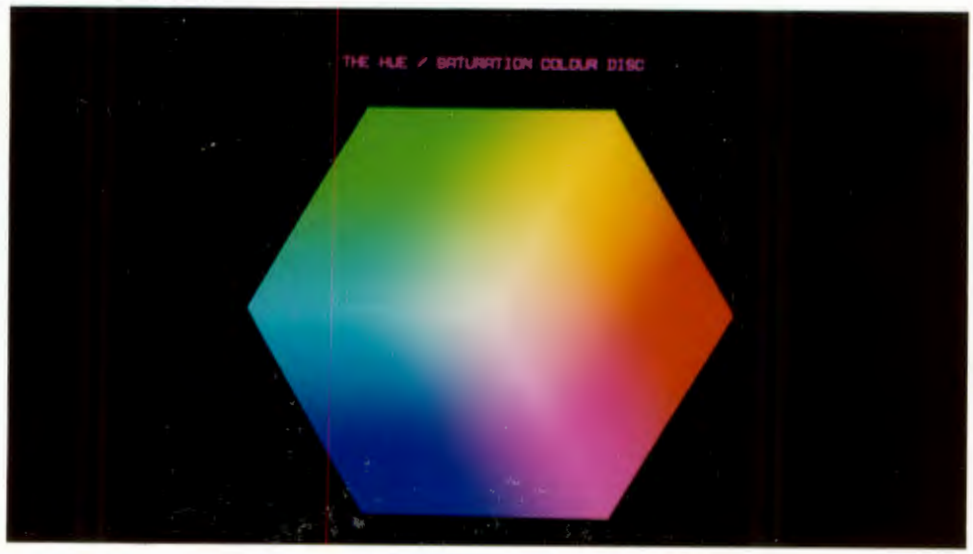


Figure 4.4: The v1 and v2 components forming the H-S plane based on the transformation given by Lehar and Stevens (1984).

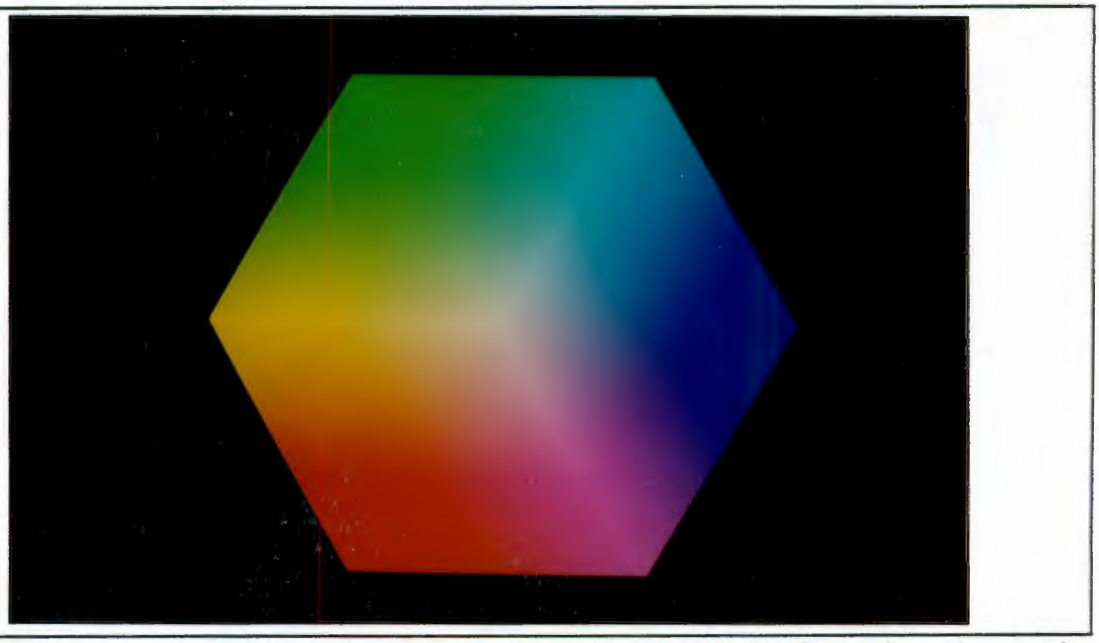


Figure 4.5: The v1 and v2 components forming the H-S plane based on the transformation given by equation (4.8) and (4.1).

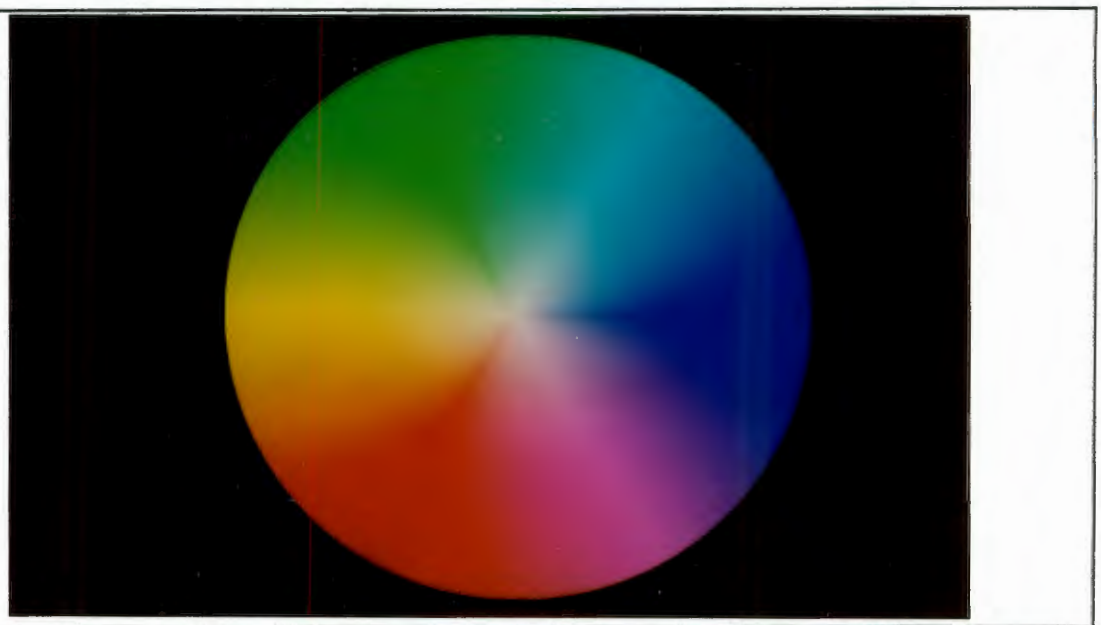


Figure 4.6: The V1 and V2 components forming the H-S plane using equations (4.8), (4.1) for H, (4.6) for S and (4.9) for V1 and V2.

Figure 4.7: The V1 and V2 components forming the 'tilted' H-S plane based on the transformation given by Ramirez in Niblack (1986).

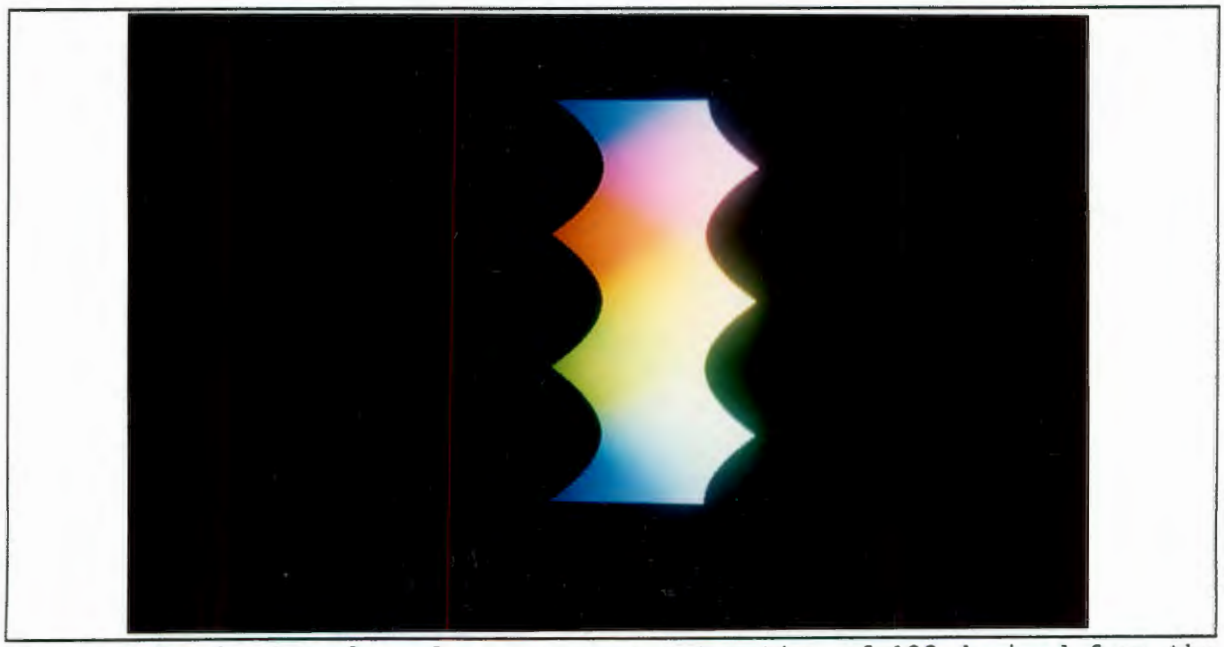


Figure 4.8: The H-I plane for a constant saturation of 128 derived from the transformation given by equations (4.8) and (4.1).

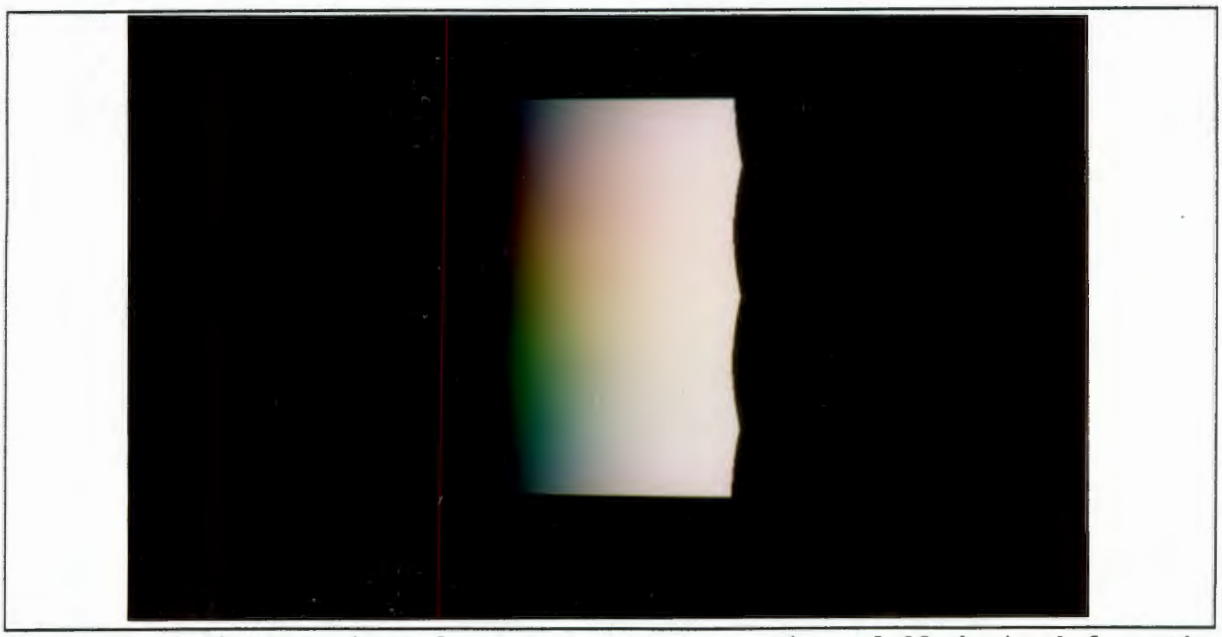


Figure 4.9: The H-I plane for a constant saturation of 20 derived from the transformation given by equations (4.8) and (4.1).

From Lehar and Stevens (1984):

$$\begin{pmatrix} I \\ v1 \\ v2 \end{pmatrix} = \begin{pmatrix} 1/3 & 1/3 & 1/3 \\ 2/\sqrt{6} & -1/\sqrt{6} & -1/\sqrt{6} \\ 0 & 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{4.5}$$

Figure 4.4 shows the v1 and v2 components of this transformation forming the regular hexagonal shape of the H-S plane.

From the Data Translation Manual and Morrissey-Golas (1989) there are three equations that describe the operation of the DT7910 RGB to HSI chip converter. The equations can be interpreted as:

$$I = \frac{R+G+B}{3}$$

$$S = 1 - \frac{\min(R,G,B)}{I} \tag{4.6}$$

$$H = \frac{1}{360^0} * (90^0 - atan(F/\sqrt{3}))$$

where $F = (2R - G - B) / (G - B)$

and H is a 10 bit value from 0 to 1, and S is a 10 bit value from 0 to 1.

It can be shown that if $v1 = (2R - G - B) / \sqrt{3}$

and $v2 = G - B$

then H can be interpreted as being similar to that in equation (4.1). From these equations a matrix transformation can be written as:

$$\begin{pmatrix} v1 \\ v2 \end{pmatrix} = \begin{pmatrix} 2/\sqrt{3} & -1/\sqrt{3} & -1/\sqrt{3} \\ 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{4.7}$$

If a scaling factor of $1/\sqrt{2}$ is used on the matrix of equation (4.7) then v1 and v2 become equal to the v1 and v2 of equation (4.5) (Lehar and Stevens (1984)). In fact any scaling factor

would only affect the saturation value if the saturation was calculated as a vector distance with $S = \sqrt{(v1^2 + v2^2)}$.

If a scaling factor of $\sqrt{3}/2$ is used on the matrix of equation (4.7) then S as a vector sum would allow a maximum saturation of 255, as opposed to 294 with no scaling factor, and 209 with the scaling factor of $1/\sqrt{2}$. Hence if maximum saturation were to be represented with an 8 bit integer, the effect of $\sqrt{3}/2$ on equation (4.7) and a combination of the intensity value would give:

$$\begin{pmatrix} I \\ v1 \\ v2 \end{pmatrix} = \begin{pmatrix} 1/3 & 1/3 & 1/3 \\ 1 & -.5 & -.5 \\ 0 & \sqrt{3}/2 & -\sqrt{3}/2 \end{pmatrix} \begin{pmatrix} B \\ G \\ R \end{pmatrix} \tag{4.8}$$

Where H and S are given in equation (4.1).

Figure 4.5 shows the v1 and v2 components of this transformation (4.8). The regular hexagon is larger than Figure 4.4 produced by equation (4.5), because of the effect of the scaling factor chosen. It is important to note that by interchanging B and R from (4.5), equation (4.8) allows for blue to be the 0^0 hue instead of red. In this HSI system all fruit ranging from bluish green to purplish red could lie between the hue angles of 52^0 and 308^0 respectively. This means that the 360 levels of hue can be mapped to 256 levels. This allows the hue value to be represented by an 8 bit integer, without reducing the accuracy obtainable with 360 different hues. The intention of the new derived RGB to HSI transformation equation (4.8) is that if it is implemented in hardware, it will be faster than the DT7910 chip.

All the above RGB to HSI transformations use the saturation value as a vector distance. The figures show that with saturation evaluated in this form, the H-S plane is generally hexagonal. The transformations can be conceptualized as different rotations and warpings of the RGB cube. It also appears that arbitrary colours can be chosen for the 0^0 hue starting angle. Some of the transformations represent intensity (I) as a simple average of the RGB values, other transformations weight the RGB values in the form 0.3: 0.59: 0.11. The effect of calculating intensity by these two methods will be shown by the results given from the experiments

covered later in this chapter.

An alternative way of calculating saturation, instead of using a vector sum, is given in equation (4.6). This is the method used by the DT7910 RGB to HSI chip converter. This saturation is calculated by dividing the minimum value of red, green and blue by the intensity I, and subtracting the result from 1. If this saturation value is multiplied by 255 then an 8 bit integer could still represent the S value. Saturation calculated in this way yields a cylinder instead of a rotated RGB cube. The new vector components V1 and V2 can be calculated using the new saturation value S and the hue value H from (4.8) and (4.1) :

$$V1 = S\cos(H)$$
$$V2 = S\sin(H) \qquad (4.9)$$

Figure 4.6 shows the H-S plane as a disc made up of the V1 and V2 vector components, using the transformation equation (4.9). Figure 4.7 shows another H-S plane made up of the V1 and V2 components of (4.9), where H is derived from equation (4.3) with (4.1) and S is from (4.6) - this transformation aims at accounting for an even intensity over the H-S plane. It should be noted that saturation will always be at a maximum when any of the R, G or B values is zero, whereas in the previous vector sum calculation of S, saturation is maximum only at the six facets of the hexagon. The experimental results given later in this chapter aim to decide which of the S calculations best represents the saturation value.

The H-I plane is another way of viewing how the above transformations manipulate the RGB values of colour pixels. Only the H-I plane resulting from equations (4.8) and (4.1) is given here, since the other transformations yield a similar effect. Figure 4.8 shows the H-I plane for a constant saturation (S) of 128. The result is effectively a two dimensional development of a cylinder of radius 128 (pixels) projected into the centre of the H-S plane of Figure 4.5. Note that if the constant saturation was chosen close to zero, then the I range would increase. Figure 4.9 shows how the I range increases for the smaller constant saturation value of 20. The maximum intensity range of 0 to 255 is when S is 0.

4.3. The Experimental Set Up for Capturing Fruit Images

In order to test and experiment with the RGB to HSI transformations discussed in section 4.2, an image capturing set up was needed.

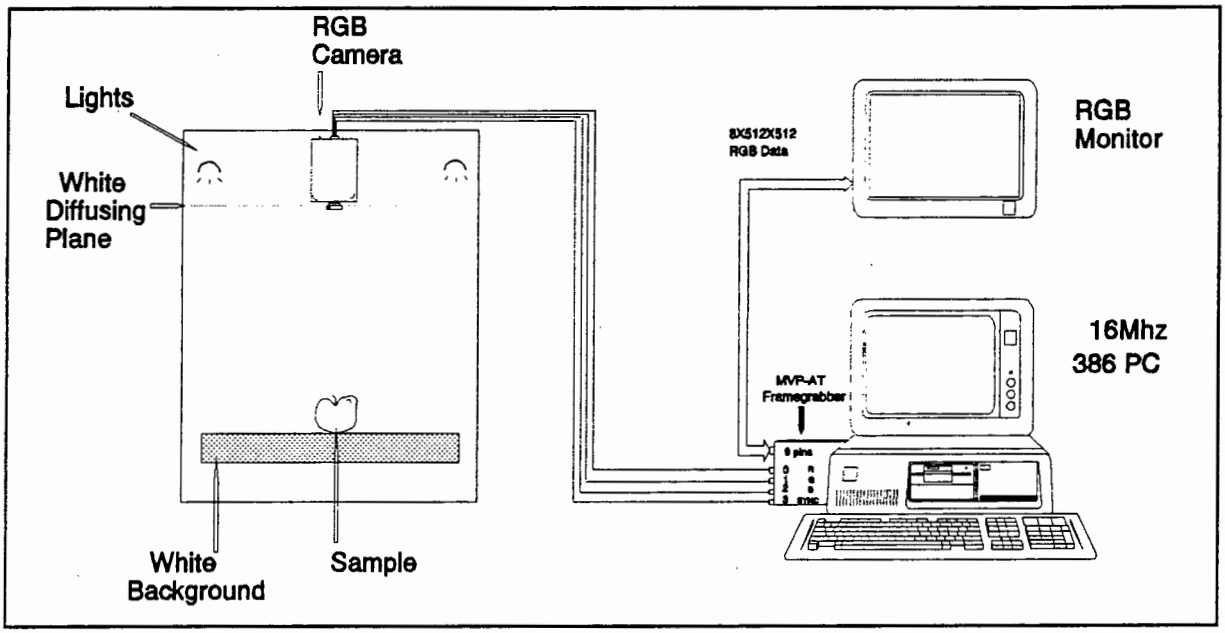


Figure 4.10: The experimental set up for capturing and displaying the sample scene.

Figure 4.10 illustrates the image capturing set up used for laboratory experimentation. The lay out to set up the scene for image capture consists of:

- an RGB colour camera with an adjustable red/blue tint knob. The JVC TK-870E colour video camera was used. This camera is equipped with a solid state CCD (Charge Coupled Device) image pickup element (JVC Instruction Book);
- sufficient lighting to provide good colour rendering. This consisted of four lighting elements, namely two 12v, 50w tungsten halogen lamps and two 230v, 60w incandescent light bulbs, placed equidistantly around the camera;
- a white paper diffusing plane to remove any gloss patches that may have been caused by the lighting elements;
- a white background for the real fruit samples. Generally the fruit samples were analyzed from standard colour charts which had colour pictures of fruit grades on a white background.
- a black box enclosing the camera and scene set up, so that ambient light would not

affect the colour of the samples.

- and a Colour Analysis Prototype Package which was developed to interact between the MVP-AT colour frame grabber, the 16MHz 386 computer, and the Philips RGB colour monitor. This software which was developed for general colour analysis is given as a program listing in Appendix A.

The fruit samples used for two of the experiments described in this chapter, were taken from the Unifruit Co. colour charts for Granny Smith, Golden Delicious and Red Starking apples. The image captured and displayed by the MVP-AT board used four frame buffers. Each buffer was 512x512 pixels and each pixel consisted of a 24 bit colour value, namely: 8 bits for each pixel in the red buffer (buffer 0); 8 bits for each pixel in the green buffer (buffer 1); and 8 bits for each pixel in the blue buffer (buffer 3). Buffer 2 contained an overlay image which could be used to show (by using an overlay mask) the region on the image being processed. When analyzing the colour features of the fruit it was found that to segment the fruit from its background a simple thresholding technique was sufficient. If the background of the fruit was white then the blue frame buffer was used for thresholding.

In the proposed industrial colour inspection system, an accurate boundary of the fruit will be provided, using a more sophisticated method than just thresholding the fruit from its background. In the laboratory set up (Figure 4.10) the colours in the fruit sample could be analyzed by sampling every colour pixel in the fruit image (i.e. line by line) or by taking a specified number of lines in the fruit so that a smaller sample set of pixels could be chosen to represent the colours in the fruit. In the experiments that follow the number of alternate lines used for sampling the colour pixels in the fruit images will be mentioned.

4.4. Analyzing the Colour Features of Three Different Apples

4.4.1. The Colour Features to be Used For Experimentation

In this section the three fruit to be analyzed are: a green Golden Delicious, a yellow/green Golden Delicious with russeting, and a green Granny Smith with a large red blemish (see Figure 4.11). In this figure the image of apple (a) was captured from the greenest Golden Delicious shown on the Unifruit Co. colour chart for Golden Delicious colours (shown later

in Plate 1). Note that the glossy highlights that are seen on the fruit were present on the actual picture sample and is not due to the experimental lighting set up. The image of apple (b) was captured from a real Golden Delicious with russeting, the stalk end is shown in the figure. The image of apple (c) was captured from the Unifruit Co. colour chart for various pink blush sizes on a Granny Smith apple.

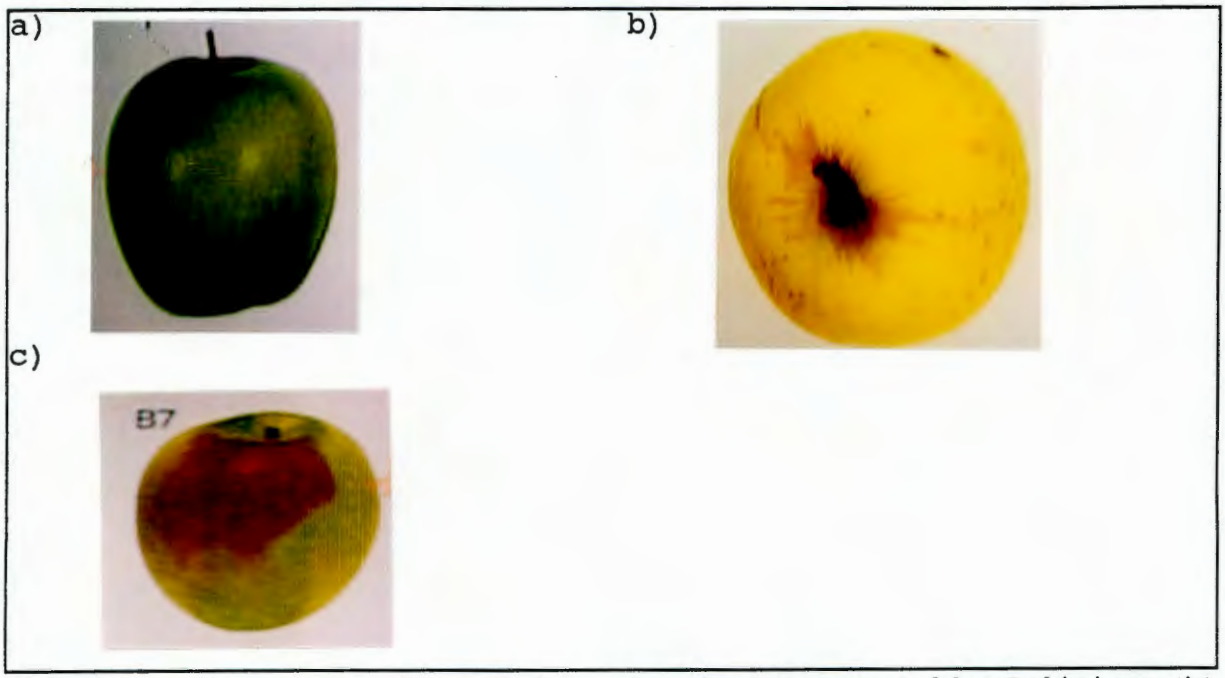


Figure 4.11: The fruit colours to be analyzed: (a) green Golden Delicious (b) yellow Golden Delicious with russeting (c) green Granny Smith with large blemish.

The aim of the experiment is to find which feature or combination of features best represent the colours on the fruit. The intention is to find the fewest features to accurately describe the fruit colour. The features that were investigated are:

the red plane (R),

the green plane (G),

the blue plane (B),

the hue plane (H) from equations (4.8) and (4.1),

the hue histogram (H vs number of pixels in the plane),

the saturation plane (S1) from equations (4.8) and (4.1),

the saturation plane (S2) from equation (4.6),

the intensity plane (I1) from equation (4.8),

and the intensity plane (I2) from equation (4.3).

The feature combinations that were investigated are:

the H-S1 plane made up of the v1 and v2 components from equations (4.8) and (4.1),
the H-S2 plane made up of the V1 and V2 components from equation (4.9),
and the H-I1 plane from equations (4.8) and (4.1).

For the purpose of displaying the single feature planes of R, G, B , H, S1, S2, I1 and I2 each of the features were made up of a grey level from 0 to 255 (0 being black and 255 being white). The plane consisted of all the pixels in the region of the image being analyzed. Hence, for example, a dark region on the plane for the red feature indicated there was not much red in that region, whereas a light region on the plane for the red feature indicated a good concentration of red component in that region. For hue, a dark region indicated blue, thus 0 is blue, 60 is cyan, 120 is green, 180 is yellow, 240 is red, and 255 is a purplish red. The hue histogram was also used in the experiment since it was a method of observing the frequency of pixels in the region of interest, having the same hue values.

4.4.2. Results of the Single Colour Features and Colour Feature Combinations for the Three Apples

The following figures and observations refer to the colour features of the three apples shown in Figure 4.11. In some cases the grey level or colour reproduction of the image feature planes presented in the figures is not of the same high quality as the same images observed on the RGB monitor. In such cases the described observations should provide adequate descriptions of the images.

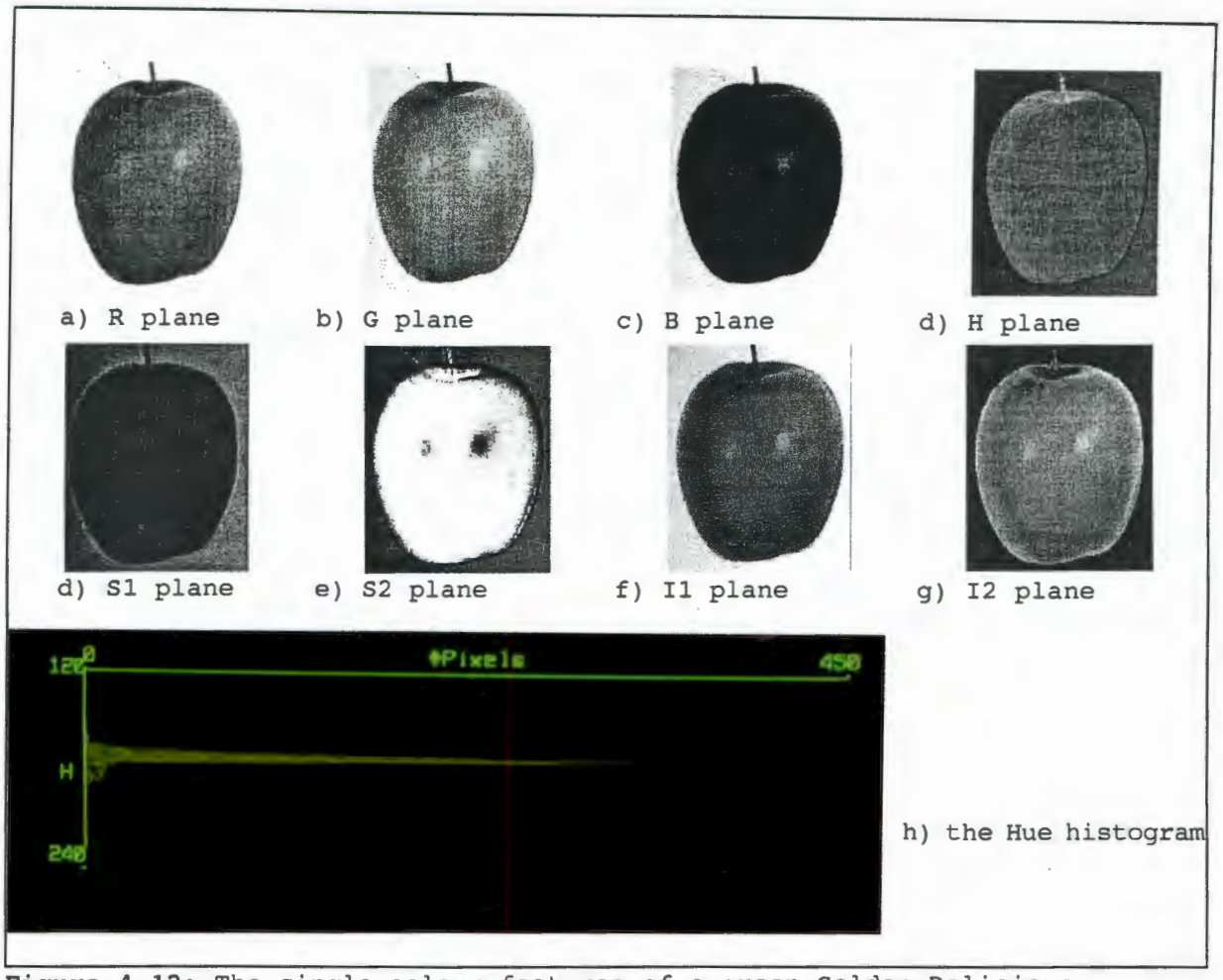


Figure 4.12: The single colour features of a green Golden Delicious

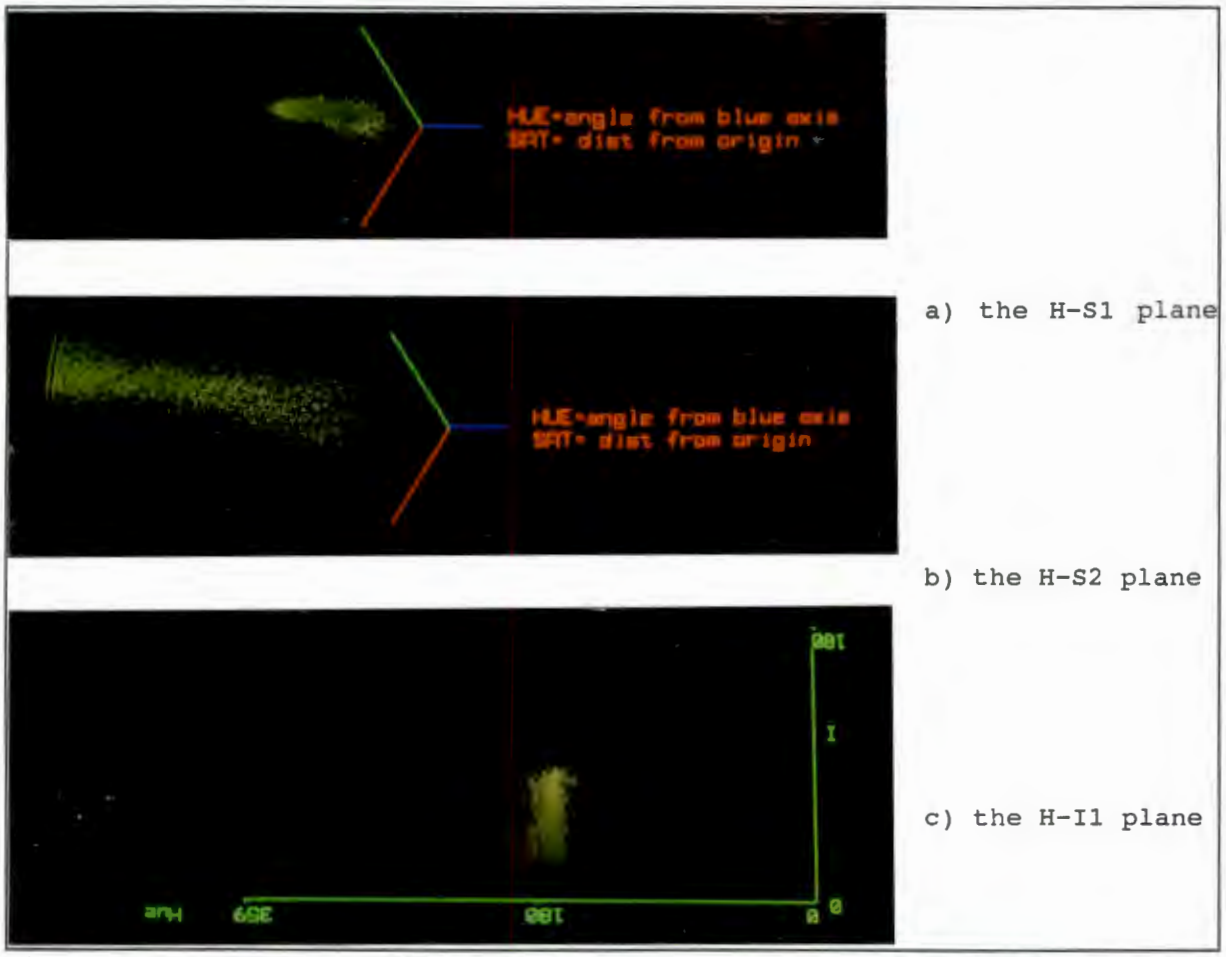


Figure 4.13: The colour feature combinations of a green Golden Delicious

Observations on the green Golden Delicious

The colour image used in this experiment is shown in Figure 4.11 (a). From Figure 4.12, the green plane is slightly brighter than the red plane, with virtually no blue component, indicating a green fruit. The hue plane is constant, except at the stalk end, where the lighter region suggests the colour is more red (or brown) than the rest of the fruit. The hue histogram was obtained by sampling and displaying every 10 lines in the fruit region, whereas all other images represented in Figures 4.12 and 4.13 show all the pixels in the fruit region. The histogram shows that hue ranges from 152^0 to 239^0 and peaks at 170^0. The high concentration of colours near the hue peak suggest the uniformity of colour over the fruit.

The S1 plane (saturation evaluated as a vector sum) is fairly uniform over the fruit except at the regions on the fruit where the gloss regions are lightest. The S1 plane shows these gloss regions as darker than the surrounding fruit region. The S2 plane is similar to the S1 plane, however the contrast in the S2 plane is noticeably much more vivid. The I1 plane (intensity evaluated as the average of the RGB components) shows what one would expect to see if the colour image of the fruit were to be displayed in black and white. Here the gloss and the darker region by the stalk all appear in this plane. The I2 component (intensity calculated as a weighted sum of each of the RGB components) is similar over the fruit image plane to the I1 component, however the contrast on the I2 plane is slightly better than the I1 plane.

From Figure 4.13 pixels on the H-S1 plane are in a more concentrated cluster than the pixels on the H-S2 plane. This shows why the contrast on the S2 plane is better than on the S1 plane in Figure 4.12. It can still be seen, however, that no matter which S plane is used, the actual hue colour spread for the fruit is still the same - narrow. The base of the clusters show the browner parts of the stalk, but the saturation values for this region are fairly intermediate (see H-S1 plane) or mainly saturated (as in the H-S2 plane).

The H-I1 plane shows that for the lower intensities by the stalk the hue value is larger (browner). The colour plot does not show the low intensities very well though. The H-I1 cluster for the green of the fruit is fairly tight, indicating the uniformity of the green over the fruit.

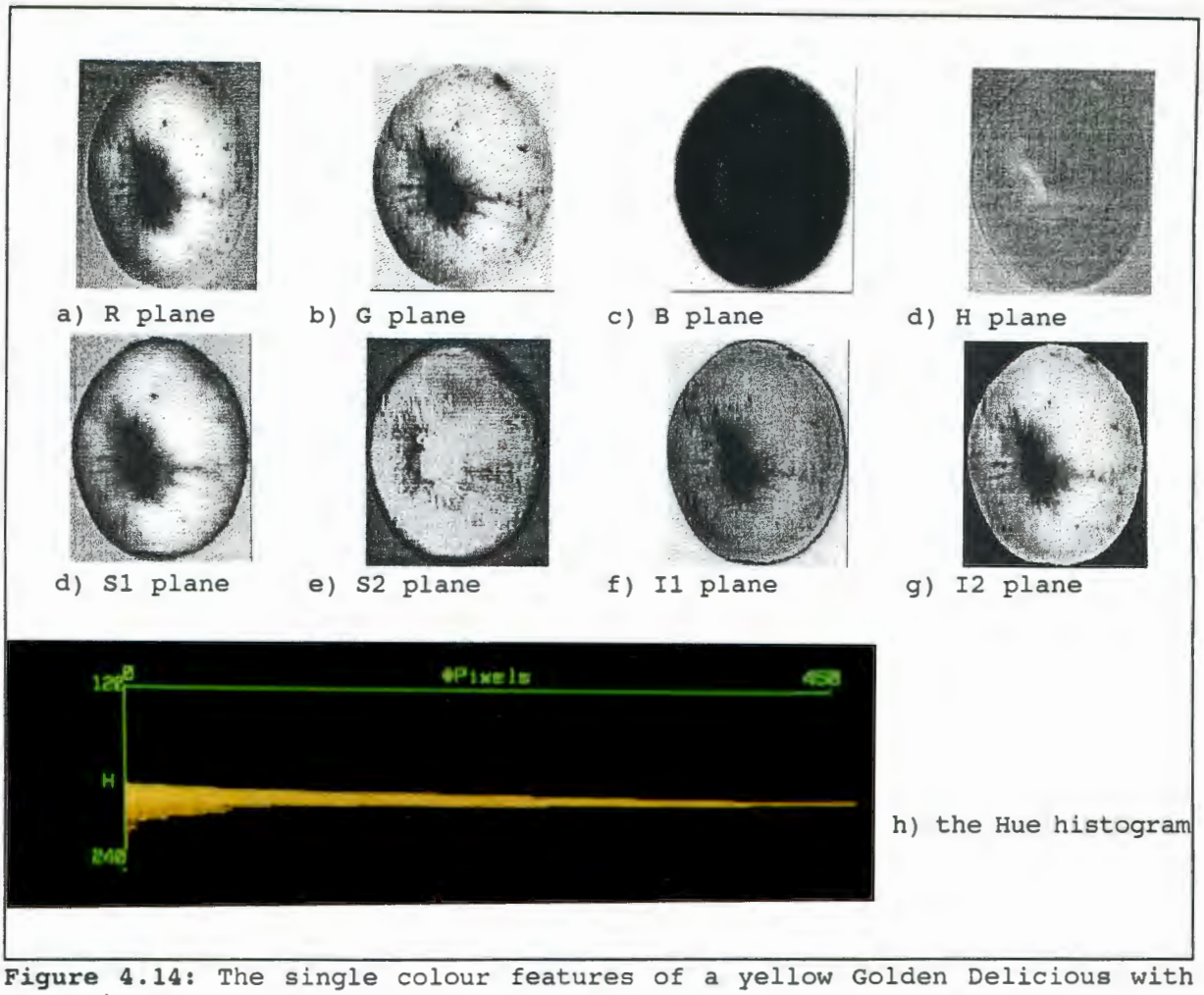


Figure 4.14: The single colour features of a yellow Golden Delicious with russeting

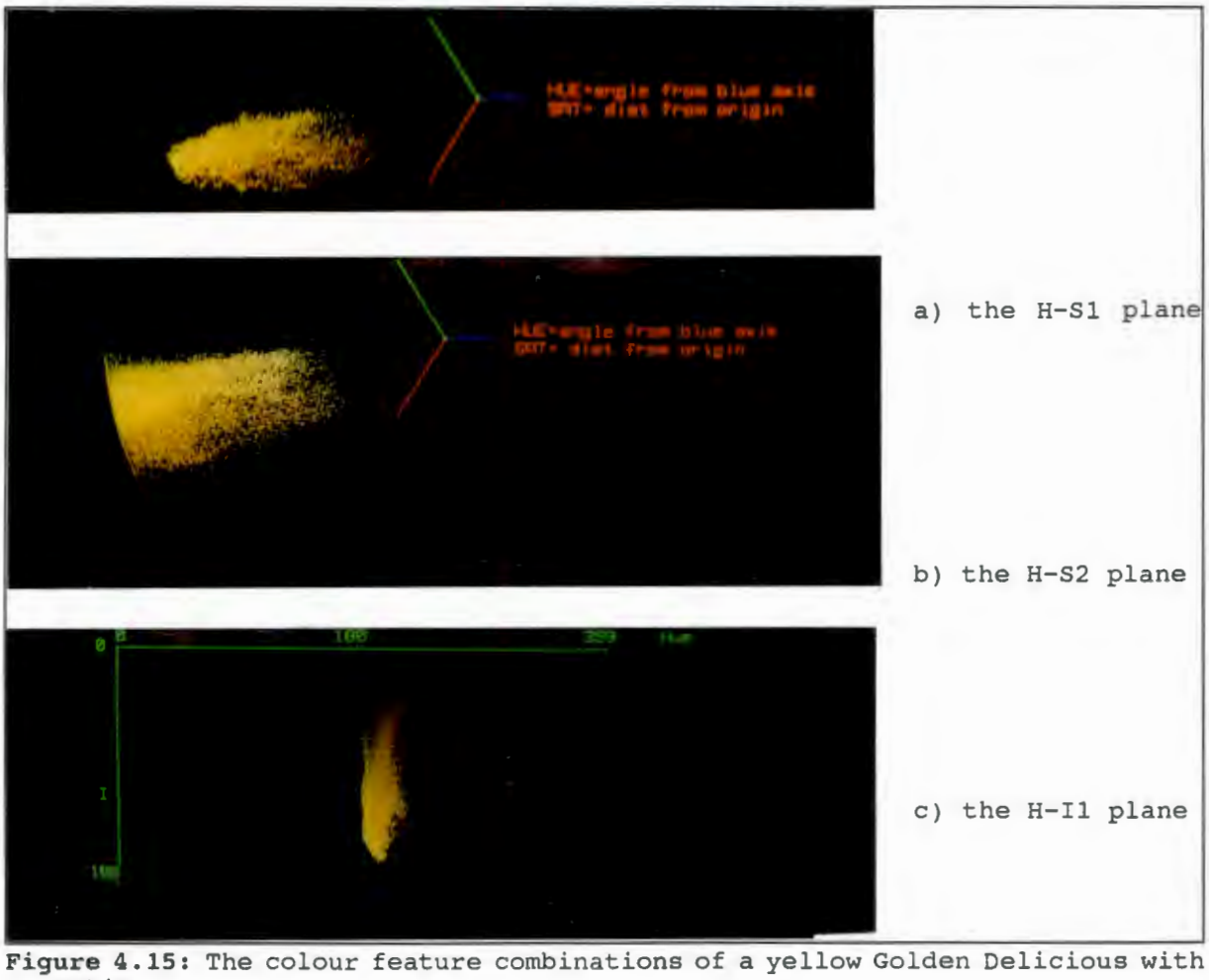


Figure 4.15: The colour feature combinations of a yellow Golden Delicious with russeting

Observations on the yellow Golden Delicious with Russeting

The fruit used in this experiment was a real Golden Delicious with russeting and several small brown spots (Figure 4.11 (b)). From Figure 4.14 the green plane shows the edge of the russeting more clearly than the red plane. Both the red and green plane are similar in grey level and together with the darkness of the blue plane, suggest the apple is yellow and the russeting is a darker yellow. The hue plane, although not shown clearly in Figure 4.14 (e), is lightest at the stalk region, less light at the russeting and fairly uniform over the rest of the apple. The hue histogram contains pixels sampled every 11 lines in the fruit image and shows that hue values spread from 183^0 to 239^0. The main peak is at 192^0, and a smaller less obvious peak for the russeting is at 210^0. All other images represented in Figures 4.14 and 4.15 show all the pixels in the fruit region.

The S1 plane shows a large saturation range in Figure 4.14 (f) with the russeting boundary not clearly defined. The S2 plane, however tends to show that most of the darker areas around the stalk and russet are totally saturated. The reason for grey level distribution showing mainly maximum saturation in S2 is because the blue values in most of the apple are zero (see equation (4.6) for calculation of S2). The I1 and I2 planes show equally well the intensity over the fruit, however the contrast between grey levels is slightly better on the I2 plane than on the I1 plane. As expected the intensity drops from the unblemished fruit skin to the russeting and brown markings, and intensity is lowest at the stalk.

From Figure 4.15 pixels are more concentrated on the H-S1 plane compared to those on the H-S2 plane. However, both planes show that maximum saturation is observed on this fruit image. This could mean that the lighting was too bright when this particular image was captured. Both planes also show the slightly redder pixel spread referring to the russeting.

The H-I1 plane shows the narrowness of the hue spread over the fruit, but with the intensity component, the darker regions of the russeting and stem appear to cluster beside the main yellow components of the fruit.

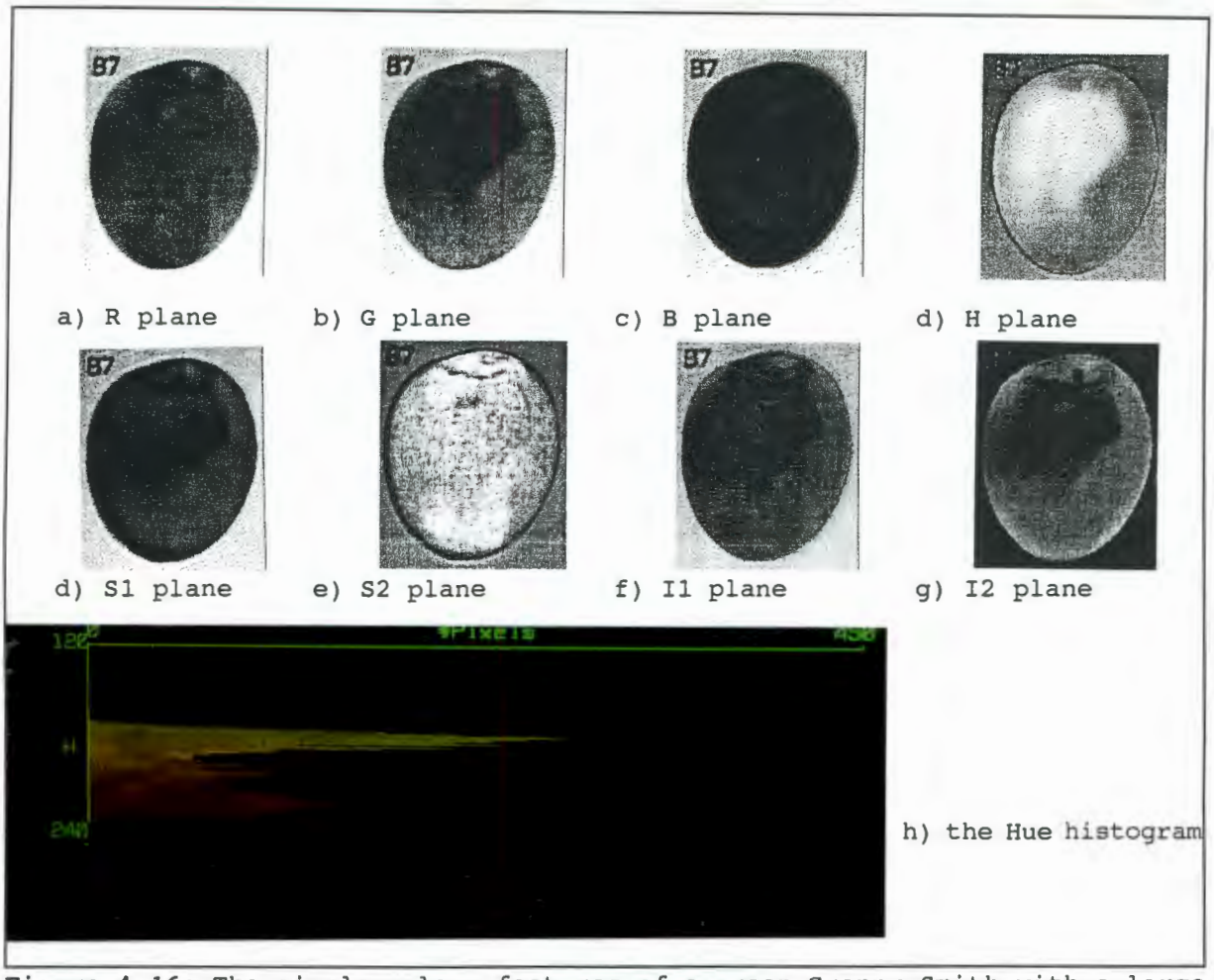


Figure 4.16: The single colour features of a green Granny Smith with a large blemish

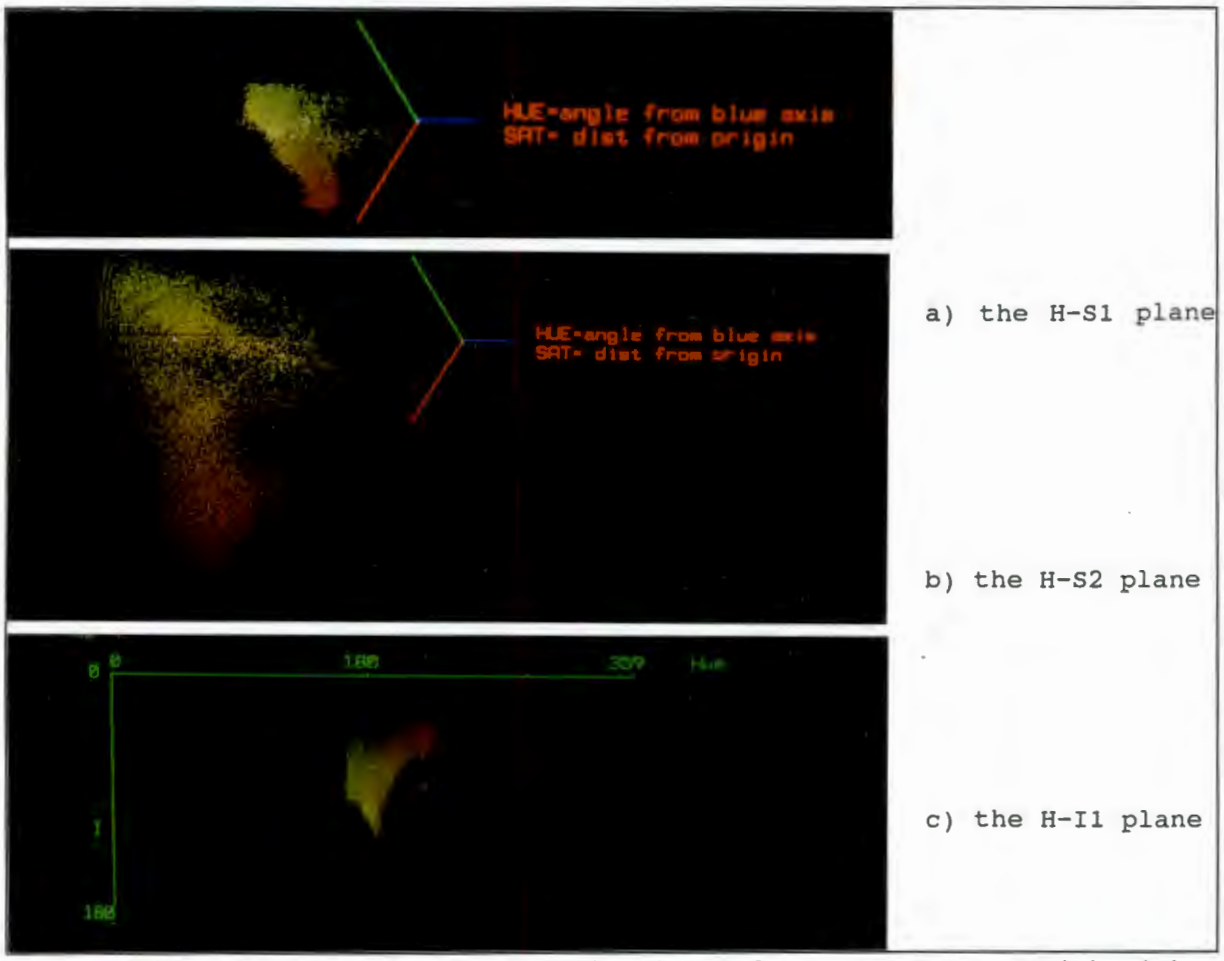


Figure 4.17: The colour feature combinations of a green Granny Smith with a large blemish.

Observations on the green Granny Smith with a large blemish

The fruit used in this experiment has a large red/brown blemish called pink blush and is shown in Figure 4.11 (c). From Figure 4.16, the red plane is fairly uniform and does not show the region of pink blush. The green plane however, has less green component where the blemish is present. The blue plane, as expected for the green/red fruit is dark indicating little or no blue component. The hue plane clearly distinguishes the red blemish from the rest of the green apple, this is indicated by the light shading on the darker shading of Figure 4.16 (d). The hue histogram (pixel samples taken every second line in the fruit image) shows that hue ranges from 159^0 to 229^0 with three distinct peaks at hue values of 174^0, 196^0 and 213^0.

The S1 plane shows that the blemish region is slightly less saturated than the rest of the fruit. This is shown by the dark region on the slightly less dark region of Figure 4.16 (e). In the case of the S2 plane the image is as if the whole S1 plane has been made brighter. The S2 plane makes it difficult to distinguish the blemish from the surrounding fruit.

As in the previous two experiments the I1 plane shows a black and white version of the colour fruit and the I2 plane is a slightly better contrast enhanced version of the I1 plane.In both cases the blemish is darker than the surrounding fruit.

From Figure 4.17 as before, the pixels on the H-S1 plane are more concentrated in the S1 direction, than the pixels in the S2 direction on the H-S2 plane. These planes show (more clearly than the grey level planes of S1 and S2 in Figure 4.16) that the red of the fruit is only slightly less saturated than the green. The H-I1 plane, however shows that the red blemish is of decidedly less intensity than the remaining green of the fruit.

4.5. Overview and Discussion

Any of the RGB to HSI transformations given in section 4.2 could have been used for the experiment described in this chapter. However, one such transformation (using equations (4.1) and (4.8)) along with secondary variations on the S and I formulae, were used to identify the colour features of fruit images. The results from the experiments shown in this

chapter reveal that certain single colour features are capable of identifying certain aspects of the colour on a fruit.

The hue value gives the actual colour distributions on the fruit and is capable of distinguishing the red blemishes on a green fruit. The I1 plane shows less contrast than the I2 plane between the varying intensities of the fruit. However, the I1 plane is still capable of identifying the blemish regions as darker areas in the fruit. The pixel saturations in the S1 plane are not as well spread as those in the S2 plane. Both saturation plane values, however, appear to be dependent on the position and colour of the lighting on the fruit. Saturation appears to be only sightly dependent of the actual colour distribution on the fruit, but in general the saturation component is quite variable over a uniformly illuminated sample.

From the above overview, one may conclude that since saturation is so lighting dependent, that the H and I features best describe the colour distribution over the fruit image. However, the v1 and v2 or V1 and V2 vector components may also be capable of classifying fruit colours, since the so called H-S plane appears to form identifiable clusters using the vector components (v1 and v2). In the next chapter clustering and classification theory will be discussed using mainly the H and I1 feature combinations given in this chapter. In chapter 6 experimental results based on some of this classification theory will be given, using most of the features and feature combinations given in this chapter. The eventual aim is to confirm whether or not H and I are suitable features to be used in the grading of fruit.

CHAPTER 5

Clustering and Classification Methods For Colour Sorting

5.1. Introduction

This chapter sets out to describe some of the clustering and classification techniques available for image data. The data to be used will be derived from colour pixels in fruit images. The ultimate goal is to classify the fruit into an appropriate grade.

For most green and red apple samples experimented with, it seems that a natural cluster forms within a certain region of the Hue, Saturation, Intensity (HSI) cylinder. For green apples (Granny Smiths and Golden Delicious) the hue of image pixels vary from 120° to 240° and the intensity of image pixels vary from 0 to 255. The fruit pixel's position within this boundary is very dependent on the colour and positioning of the lighting.

The objective in statistical image classification is : *Given a set of objects, assign each object to one of a set of classes.* In terms of fruit classification the objects are the fruit and the classes are the colour grades of the fruit. Alternatively, the objects could be the pixels in the fruit image, and the classes are the various types of markings and colorations of the fruit image. This latter alternative will be used in describing the supervised and unsupervised algorithms.

Why cluster and classify? It has already been shown that fruit may be classified without clustering. The number of pixels with hues within a user specified hue range, over the whole fruit pixel area, gives a percentage coverage of specified hue on the fruit image. The clustering of fruit image pixels on a hue-intensity (H-I) 2-D plane gives almost immediate classification of the fruit - to the human eye. That is, a tight cluster with a narrow hue range and narrow intensity range, suggests that the fruit is of high grade since it has almost uniform colour (Figure 5.1a). An H-I plane with almost two clusters visible suggests the fruit is not

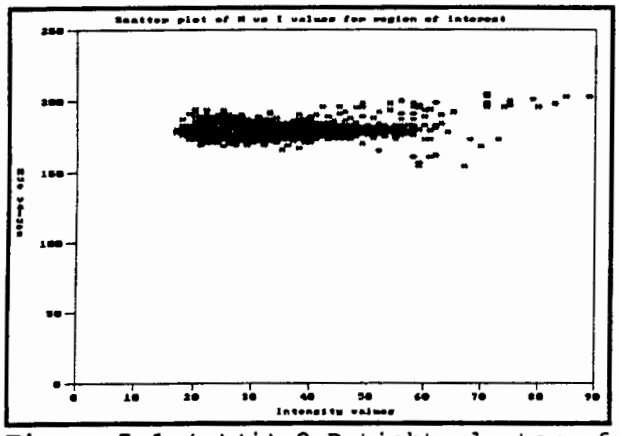
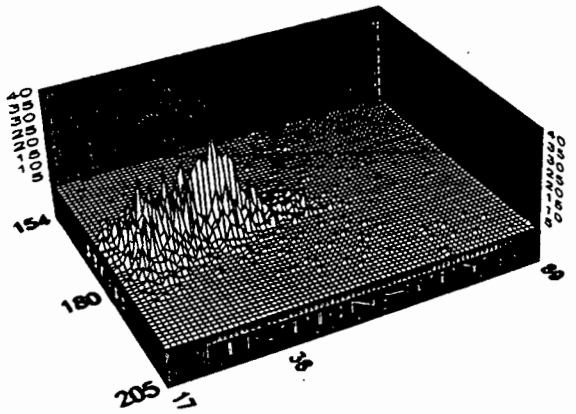


Figure 5.1:(a)(i) 2-D tight cluster of all pixels in a green apple image



(ii) 3-D histogram of plot in 5.1(a)(i)

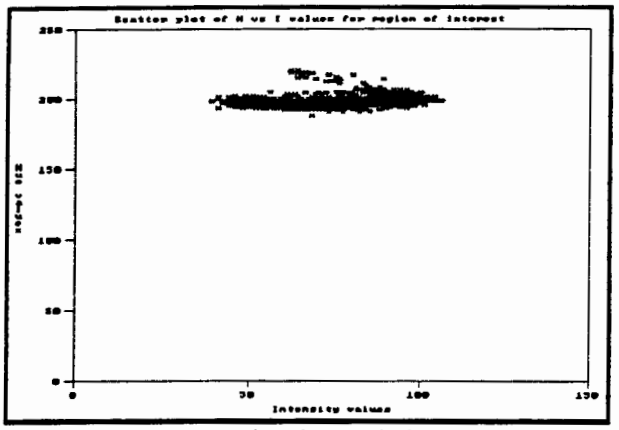
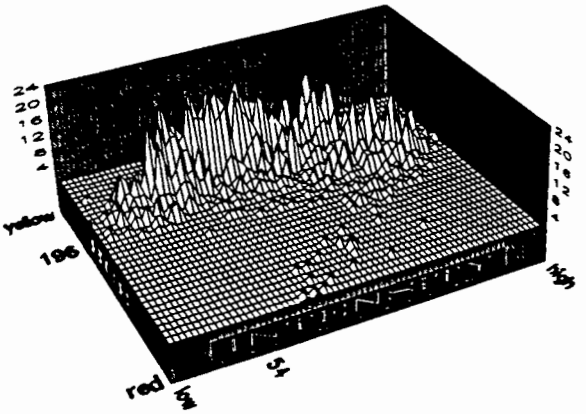


Figure 5.1:(b)(i) 2-D tight cluster of all pixels in a yellow apple image.



(ii) 3-D histogram of plot in 5.1(b)(i)

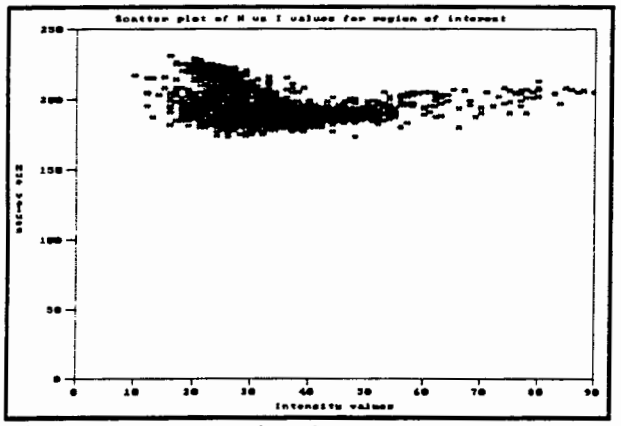
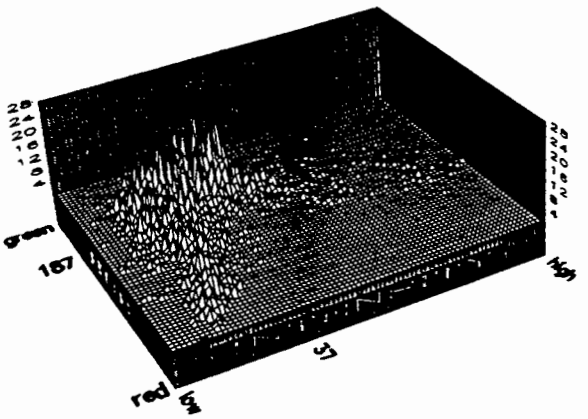


Figure 5.1:(c)(i) 2-D cluster of all pixels in a green apple with small blemish.



(ii) 3-D histogram of plot in 5.1(c)(i)

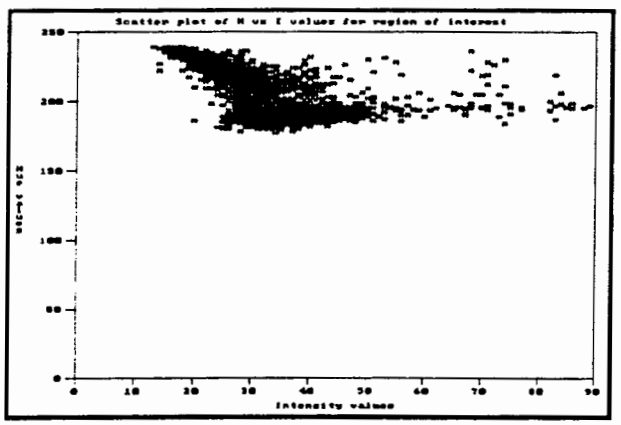
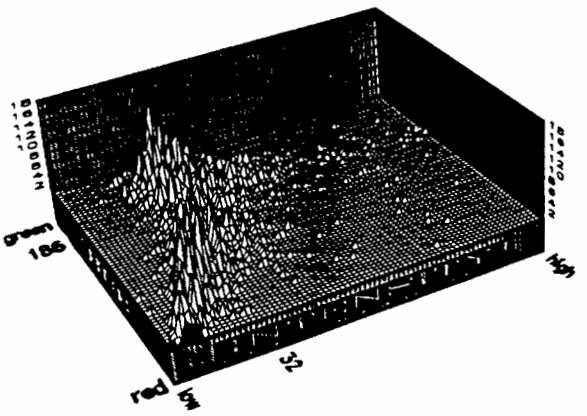


Figure 5.1:(d)(i) 2-D cluster of all pixels in a green apple with large blemish.



(ii) 3-D histogram of plot in 5.1(d)(i)

uniformly one colour but may contain a blemish (see Figures 5.1c, 5.1d). The upper and lower limits of the hue and intensity axes shown in Figures 5.1b(ii),c(ii) and d(ii) are represented, for clarity, as either a colour or high and low, respectively. The actual values of these limits can be read off the corresponding 2-D plots. The values which are shown within the upper and lower limits of the H and I axis in the 3-D plots corresponds to the peak colour in each scatter plot.

The human eye is able to recognise the clusters and hence classify the fruit in real time. It is thus hoped that the use of clustering and classification algorithms on the fruit image will achieve the goal of accurate classification in as near to real time as possible.

In the first sections a pixel p will have associated value v, where this value is considered as multi-band data, written as a vector:

$$v = \begin{bmatrix} v_1 \\ v_2 \\ . \\ . \\ v_n \end{bmatrix} = \begin{bmatrix} H \\ I \end{bmatrix}$$

The elements (v_1 and v_2) of v are features of hue (H) and intensity (I). Note that v_1 and v_2 should not be confused with the vector components v1 and v2 which are used to derive the H and S values from RGB, as shown in the previous chapter. The H and I features were preliminary choices which, by experimentation shown in chapter 4, suggested they were among the best representation of pixel values in the fruit. Saturation was another feature considered, as were the vector components v1 and v2. This chapter will only explain clustering and classification theory and examples, by using the H and I features. This chapter also shows that the mean hue (*Have*), hue variance (*Hvar*), mean intensity (*Iave*), and intensity variance (*Ivar*) for all pixels in a fruit can also be used in the fruit classification process. A fruit F can be represented as the vector :

$$F = \begin{bmatrix} Have \\ Hvar \\ Iave \\ Ivar \end{bmatrix}$$

There are two fundamental categories of classification methods : *supervised* and *unsupervised.* In the supervised methods, the user "supervises" the process by initially selecting some pixels from each class. From these, the classification algorithm determines what each class "looks like", and then assigns each pixel of the image to one of the classes. In the unsupervised methods, the classes are determined within the algorithm by locating clusters in pixel space (eg the H-I plane), and assuming each cluster corresponds to a class. The problem becomes one of cluster identification. A final step in the unsupervised case, which must be done by the user, is to decide which clusters represent which physical classes such as good parts of an apple, blemishes, stalk, etc.

Sections 5.2 and 5.3 cover the theory behind selected supervised and unsupervised classification methods, respectively. Section 5.4 uses this theory in examples that predict the outcome of such classification methods used on fruit images. This chapter ends with an overview and discussion.

5.2. Supervised Classification

In this section two main supervised classification methods will be discussed. They are the complex parametric Bayesian classification and the simplistic minimum distance classification methods.

5.2.1. Parametric Bayesian Classification

Parametric Bayesian classification is derived from Bayesian Maximum Likelihood Classification (Niblack (1986b), Duda and Hart (1973), van Ryzin (1977)). Consider a blemished green apple image. If the one band data of hue only is used then a simple decision rule based on the hue histogram can be used:

Assign all pixels above a certain hue as class blemish.
Assign all below to unblemished.

Here the decision rule is just a threshold. The problem becomes more interesting as the number of classes and bands increases. The Bayesian Maximum Likelihood method is one

way to derive the decision rules. As one looks again at the apple image, several areas are known to be blemishes and several known to be unblemished. These are called training areas, because they will be used to "train" the classifier. A normalised histogram based on hue could be made for each training area. For example $Hhist_u$ and $Hhist_b$ for the unblemished and blemished histograms respectively.

In theory, each training area will have an associated probability with it. For example, assume that fruit being sorted by an automatic colour fruit sorter has 20% chance *q(b)* of having a blemish. This probability would thus weight all blemish pixels in a blemish histogram. In addition the 80% probability *q(u)* weighting to unblemished pixels would reduce the unblemished histogram.

By including the a priori probabilities to weight the histogram curves, the following decision rule is made :

> For v the value of pixel p, assign p to blemished if $q(b)Hhist_b$ is greater than $q(u)Hhist_u$. Assign p to unblemished otherwise.

This is the basic maximum likelihood decision rule. It can be extended to multiple classes, and multiple bands . The two bands of hue and intensity give the 2-dimensional histograms in Figure 5.1(ii) which are seen as 3-D surface plots.

In using Bayes Decision rule, the result of Bayes Formula is used:

$$q(i \mid v) = \frac{q(v \mid i)q(i)}{q(v)}$$

The terms used are:

q(i): The a priori probability, assumed known before classification starts, that any pixel belongs to class i. Namely *q(u)* and *q(b)* of values 80% and 20% respectively in the above example.

q(v | i): The probability of v given i. This is estimated from the normalised histogram

of class i. It is the class conditional probability density function for class i, or the probability that a pixel from class i has value v.

$q(i \mid v)$: The probability of i given v. It is the probability that a pixel with value v is in class i.

$q(v)$: The sum of the $q(v \mid i)$ over all i. This is simply a normalization factor and is not a probability. Since it is the same for all i it can be ignored.

Bayes maximum likelihood rule can now be stated as :

Assign pixel p to class i for which $q(i \mid v)$ is maximum.

By ignoring the constant $q(v)$ Bayes rule can be rewritten as:

$$g(i \mid v) = q(v \mid i)q(i)$$

The conceptual steps thus needed to classify an image are:

1. Select training areas for each candidate class. These are areas known to contain pixels in the class. In general the more training data per class the better, since this should give a truer representation of the class histogram.
2. Compute the n dimensional histogram for the training data for each class, where n is the number of features, and normalise the histograms, giving $Hhist_i(v)$. Use these as estimates of the conditional probability density functions $q(v \mid i)$ giving the probability that a pixel has value v given it belongs to class i. There is one $q(v \mid i)$ for each of the classes.
3. Estimate the a priori probabilities $q(i)$ and use them to scale the $q(v \mid i)$.
4. Classify each pixel in the image by computing, for each class i, $g(i \mid v) = q(v \mid i)q(i)$. To classify the pixel, assign it to the class i for which $g(i \mid v)$ is maximum.

Steps 1, 2, and 3 are done once, then step 4 is applied to each pixel in the image to produce the output or segmented image.

Implementing $q(v \mid i)$ on a computer can done two ways. Firstly it can be estimated from the normalised n dimensional histogram over the training data. If there are m possible pixel values per band and n bands, each $q(v \mid i)$ requires a table of size m^n and a c class problem requires tables of size cm^n. For the fruit classing problem let there be 4 classes, 2 bands (hue and intensity) with pixel values from 0 to 255 in each band then about 32K of computer memory is needed ($4x256^2$). Memory partitioning becomes more cumbersome as the number of bands and classes increases.

The second method to compute $q(v \mid i)$ is to assume that it is a Gaussian distribution. Hence in the one dimensional case (eg hue) the parameters that must be estimated are the mean and standard deviation of the distribution, for each class i of training data. In the n dimensional case, to perform the classification of an image using this parametric form, the training data must be used to compute the n dimensional mean and the $n \times n$ covariance matrix Σ_i for each class (Gleb (1982) gives a useful definition of the covariance matrix).

It can be shown that using Gaussian distributions and some simplifications that $g(i \mid v)$ can be expressed in the following form: (Niblack (1986b))

$$g(i \mid v) = \ln q(i) - 0.5(\ln|\Sigma_i|) - 0.5(v - m_i)^T\Sigma_i^{-1}(v - m_i) \qquad (5.1)$$

The expression: $(v - m_i)^T\Sigma_i^{-1}(v - m_i)$ is usually referred to as the Mahalanobis distance, namely the distance from v to m_i weighted by Σ_i^{-1}.

Equation (5.1) is the form normally used in computer implementations, and in this case, p is assigned to the class for which $g(i \mid v)$ is minimum. This now covers the complete method for doing a parametric Bayesian classification.

5.2.2. Minimum Distance Classification

The Bayesian maximum likelihood classifier of the previous section is probably the most common method used to classify multi-band image data. It is fairly computationally expensive to run, and in the parametric form using the Gaussian assumption for class distributions is sometimes troublesome. Other classification methods are possible, and to define an alternative classifier, the main requirement is to define the decision rules. One class of decision rules are "minimum distance" rules.

Let m_i be the mean value of the pixels in training class i. In an n feature problem, m_i is a point (or vector) in n dimensional space. Minimum distance classifiers assign a pixel to the class i for which the distance from the pixel value v to m_i is minimum. Different distance measures may be used. These different measures have various theoretical or computational advantages. The formulae for two common ones are:

a) EUCLIDEAN DISTANCE

$$d(v,m_i) = \sqrt{((v_1 - m_{i1})^2 + \ldots + (v_n - m_{in})^2)} \qquad (5.2)$$

In the example of fruit classification:

$n = 2$ *bands*

$i = 1 .. 4$ *classes*

$v_1 = H$ *pixel value at band 1*

$v_2 = I$ *pixel value at band 2*

It can be shown that (5.2) is a special case of the parametric Bayesian classifier (5.1), in which the features are statistically independent, with equal variances, and the a priori probabilities are equal.

b) L1 or "City Block" DISTANCE

$$d(v,m_i) = |v_1 - m_{i1}| + \ldots + |v_n - m_{in}| \qquad (5.3)$$

Both distance measurements given in a) and b) are computationally faster than Bayesian classification but may sacrifice classification accuracy. Because the variances are used in the parametric Bayesian classifier but not the minimum distance, the Bayesian method will be more accurate. If the square root is not taken in the minimum distance equation of 5.2 then acquiring the value of $d^2(v,m_i)$ is now faster and easier to calculate and can still represent minimum distances. Consequently, this modified form of the Euclidean distance measurement was the formula used for classification in the experimental results that are shown in the next chapter.

5.3. Unsupervised Classification

Methods of unsupervised classification attempt to find clusters in the distribution of the pixels in pixel space (i.e. the features associated with a pixel form a pixel space, for example the H-I plane is a two dimensional pixel space for pixels represented with H and I features). Although clusters are often fairly easy for a human to identify in one and two dimensional plots, their centres and boundaries are difficult to identify mathematically. Cluster analysis is a field of study used to identify clusters mathematically. This section states the K-means algorithm (Niblack (1986b), Duda and Hart (1973), SAS/STAT User's Guide (1990)) and the ISODATA (Ball and Hall (1965)) unsupervised classification methods.

5.3.1. K - Means Algorithm

The K-means algorithm is an iterative clustering method. Expressed in terms for image data, it has the following steps. Initially the user supplies a set of means, or cluster centres, m_1, m_2, m_3, ... (and thus implicity the number of classes). Each m_i is a vector in n dimensional pixel space :

1. For each pixel in the image, assign the pixel to the class whose mean is closest. (one of the minimum distance formulae in section 5.2.2 could be used).
2. Recompute the mean of each class as the average of the pixels assigned to it.
3. If any of the class means has changed significantly, go to step 1. Otherwise stop.

5.3.2. ISODATA

ISODATA (Iterative Self Organising Data Analysis Techniques), is the name of a K-means type of algorithm that includes parameters to allow classes to be split and merged. The additional parameters are:

1. A threshold on the minimum distance between two cluster centres, so that two cluster centres close together are merged.
2. A threshold on the standard deviation within each band for each class, so that a cluster with too much variability is split in two. One way of doing the splitting is to define two new cluster centres at some given distance (another parameter) on either side of the old mean in the band of maximum standard deviation.
3. A minimum number of pixels in a cluster. Clusters with less than this minimum are dropped and their pixels assigned to other clusters (or a single cluster for all unclassified pixels). This avoids many small clusters.
4. A maximum number of clusters which can be merged at any iteration. This is needed to avoid over-merging.

Even with these parameters, best results are usually obtained when running this unsupervised classifier in a supervised mode. In this case, after each iteration, the user views the results, adjusts parameters to control the splitting and merging, and stops the iterations when satisfactory classes have been obtained.

Ball and Hall (1965) illustrate, in great detail, the capabilities of using ISODATA. It is generally found that at most six iterations are required for adequate cluster classification.

The advantage of an unsupervised classification such as ISODATA is that it tends to identify clusters in the pixel space that are numerically separable, whereas the advantage of supervised classification is that the classes that are used are meaningful to the user. The methods may be combined by using the unsupervised clustering to check the numerical separability of the user defined classes prior to the supervised run. The next section will give examples of how the theory cover in this section can be applied to fruit image classification.

5.4. Examples of Theoretical Classification Methods

The subsections that follow aim to give some feel as to how the theory discussed in the earlier sections of this chapter can be applied to fruit classification. Throughout this section, the classification methods proposed will be illustrated on 10 hypothetical Granny Smith apple samples. In some cases these apples will be used as training data. It must be noted that more apple samples should be used if the proposed industrial inspection system is to test the methods described below.

Each apple has an associated letter identifying it. The apples are divided into four classes but only three grades :

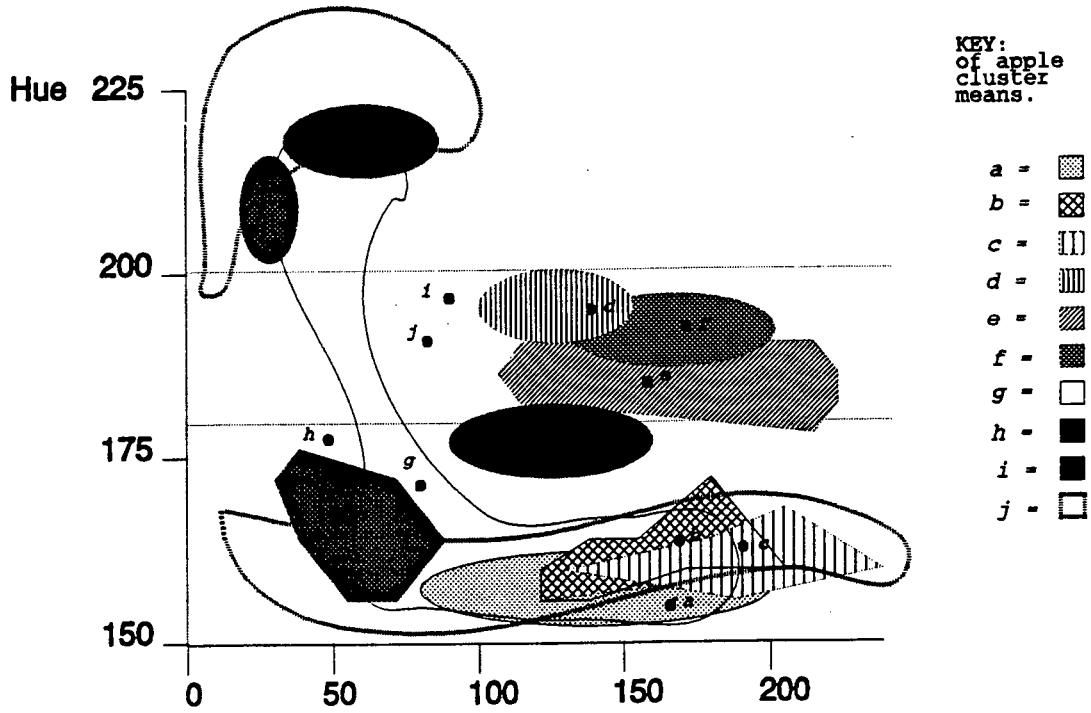
3 First Grade pure green apples	(a,b,c)
3 Second Grade slightly yellow apples	(d,e,f)
2 Third Grade green apples with small red/brown blemishes	(g,h)
2 Third Grade green apples with large red/brown blemishes	(i,j)

5.4.1. Two Level Clustering Based on Minimum Distance

It is evident that there are two approaches or steps to sort on fruit colour. The first step is to identify what the distribution is of pixels in the fruit image. The second step takes the result of the pixel distributions and decides the class (or grade) of the fruit using some decision rule.

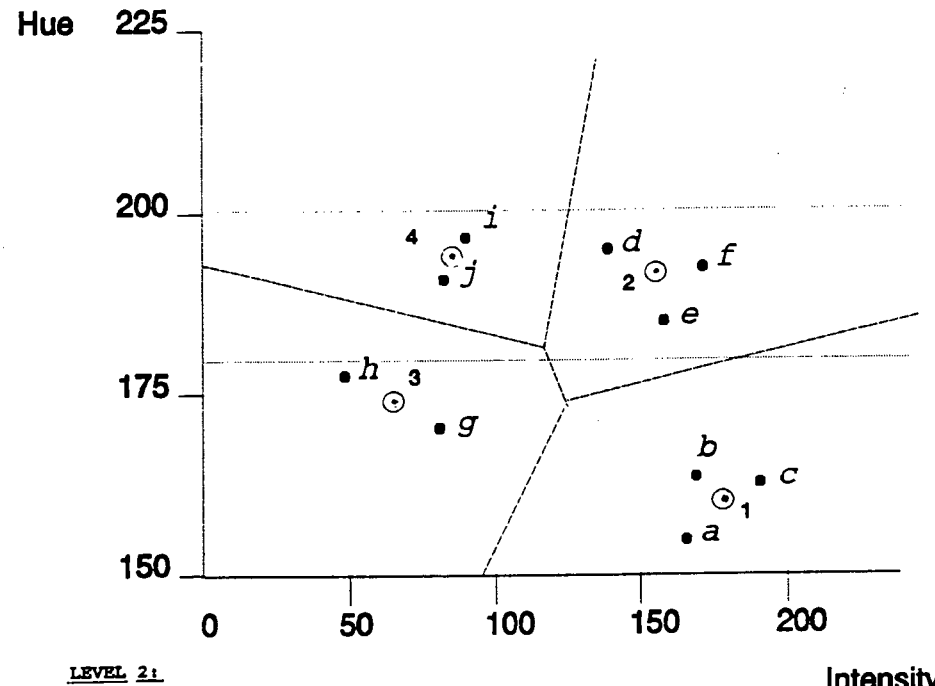
The two steps can be called two levels of clustering. The first level (Figure 5.2 (a)) is represented as a 2-D plot of pixel hue versus intensity for all apple samples. The mean hue (*Have*) and mean intensity (*Iave*) for each apple can be assumed to represent the result of each apple's pixel distributions.

The second level of clustering (Figure 5.2 (b)) is to split the results of the pixel distributions (ie the means) based on a minimum distance criterion. Once the training data has defined the classes, any apple hue and intensity mean can classify the apple to the class with the



LEVEL 1:

a) 2-D plot representative of all the pixels in the 10 apple samples.



LEVEL 2:

b) Ideal clustering of representative means a,b,c,d,e,f,g,h,i,j for each apple image The means of the clusters give rise to the four classes 1,2,3,4.

Figure 5.2: The two levels of clustering with 10 apple samples (a) level 1, (b) level 2.

nearest *Have* and *Iave*. Hence each point in the level two cluster represents a fruit F of the form:

$$F = \begin{bmatrix} Have \\ Iave \end{bmatrix}$$

This classification method is simple, quick and easy to implement. Foreseeable problems on accuracy do exist. There will be large amounts of overlap with points in the level 2 clustering and hence classes may not be easily distinguished. For example, a fruit with equally large areas of green and brown (Third Grade) will have a mean point very close to a fruit which is mainly yellow (Second Grade). The method of classifying with variances (Section 5.4.3.) attempts to resolve this problem. But first section 5.4.2 will illustrate examples that use level 1 clustering and classification.

5.4.2. Classifying Using K-means and ISODATA

Let the training data be the a priori means of specific browns, yellows, and greens. Hence three classes are initially set up. (see Figure 5.3).

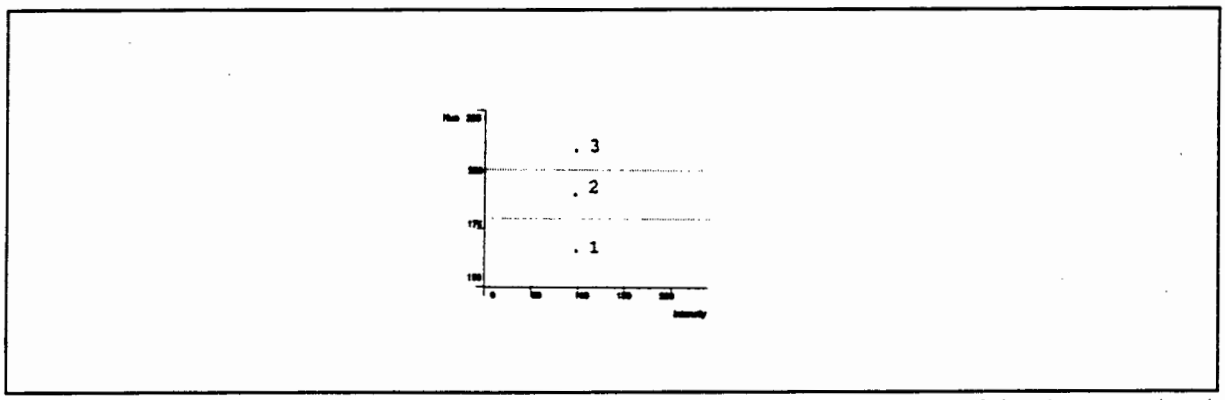
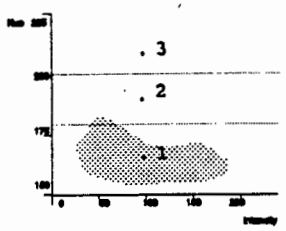
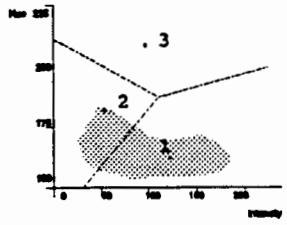


Figure 5.3: Three classes initially set up on the H-I plane. This is a priori data for use in the ISODATA iterations of figures 5.3.1 and 5.3.2.

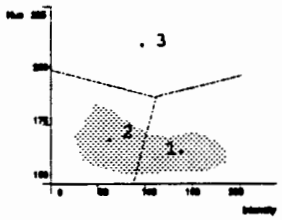
The example in Figure 5.3.1, illustrates the classification of an unblemished green apple using the K-means with ISODATA iterations. The first iteration adjusts the mean point of each class, according to the pixel distributions. The class boundaries are then redefined. It is shown that the apple can be classed accurately after three iterations.



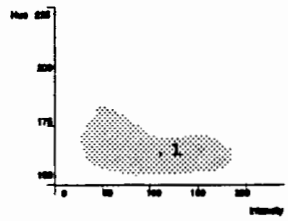
a) Place pixels of apple on H-I plane



b) adjusted means from a) and redefinition of class boundaries.

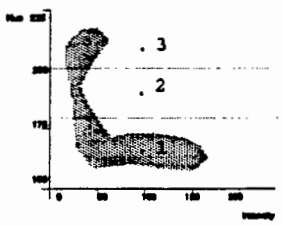


c) Adjusted means from b) and redefinition of class boundaries.
NOTE: class 3 would have been eliminated (too few elements),

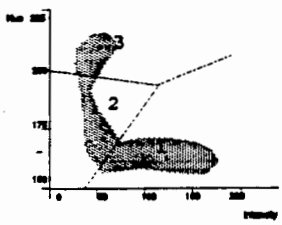


d) Merge class 1 & 2 if hue separation is small, and redefine new mean as class 1 (since new mean is nearest to a priori class 1 mean). Only one class remains. Hence the fruit is graded as class 1.

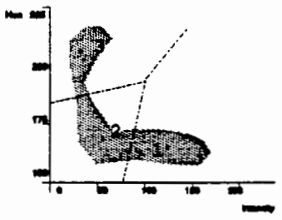
Figure 5.3.1: An example using ISODATA to classify an unblemished green apple.



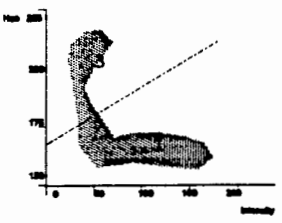
a) Place pixels of apple on H-I plane, using three a priori class means.



b) adjusted means from a) and redefinition of class boundaries.



c) Adjusted means from b) and redefinition of class boundaries.



d) Merge class 1 & 2 if hue separation is small, and redefine new mean as class 1 (since new mean is nearest to a priori class 1 mean). Two classes remain, no further iteration is needed. A class 3 present means the apple is of low grade.

Figure 5.3.2: An example using ISODATA to classify a green apple with a brown blemish.

The example in Figure 5.3.2, illustrates the classification of a green apple with a brown blemish. It is shown that iteration with K-means and ISODATA, yields an accurately classed apple after three iterations.

The examples show that the iterative process can be used to indicate the classes of pixels on the fruit images. From this information the fruit can thus be graded. This method has the problem that a whole iterative process must be carried out for each fruit image, in order to classify it. It appears that previous classifications do not help in reducing the number of iterations for any subsequent classifications. This method is therefore not efficient, although it may class fruit accurately.

5.4.3. Classifying Using Variance Data

From section 5.4.1. it is apparent that level 2 clustering is not sufficient if each point in the cluster is being represented by a mean hue and mean intensity for a fruit. The problem here is class overlapping.

By experimentation it can be shown that there are threshold standard deviation values between the different classes of apple images. The thresholds on variance for hue-intensity bands could be between:

1. a uniformly coloured fruit (small variance);
2. a uniformly coloured fruit with a small uniformly coloured blemish (large variance);
3. a non-uniformly coloured fruit (largest variance).

Hence to avoid the overlapping problem further features should be added to each point at the level 2 cluster. It seems that variance in hue and intensity are good features to consider. Each point now in the cluster space is changed from:

$$F = \begin{bmatrix} Have \\ Iave \end{bmatrix} \quad to \quad F = \begin{bmatrix} Have \\ Hvar \\ Iave \\ Ivar \end{bmatrix}$$

Clustering and classifying with four feature bands may be difficult to represent conceptually. However, the example below attempts to show that the overlap problem may be removed, by using the four band vector.

Example: Minimum Distance Classification With 4 Bands.

Assume the 10 apples samples described earlier are used, and that the means and variances for their hue and intensity pixel values are calculated. The result is the four class clusters as in Figure 5.2 (b) but now in 4 dimensions.

An apple can now be classed once its four parameters are calculated. The minimum Euclidean distance of that 4 dimensional point, to each of the four class means will result in the apple hopefully being correctly classed. ie from equation (5.2):

$$d(F,m_i) = \sqrt{((Have - m_{i1})^2 + (Hvar - m_{i2})^2 + (Iave - m_{i3})^2 + (Ivar - m_{i4})^2)}$$

where m_i is the mean point for all n points in class i. This new point in the cluster space can now be used to update the class means as in the K-means and ISODATA algorithms.

The 4 dimensional point representation of a fruit could be more accurately classed using the equation for parametric Bayesian classification (equation (5.1)). Now the 4x4 covariance matrix for each class must be used. One may predict that accurate classification is possible with this method. However, the calculation time for equation (5.1) may be too great, if one thinks of the time to calculate the 4x4 covariance matrix, followed by the Mahalanobis distance and finally the parametric classification into one of the four classes.

This method shows that high computation speed must be sacrificed for greater classification accuracy, in the automated fruit sorting process.

5.4.4. Speed and Accuracy Using Parametric Bayesian Classification, ISODATA and Hue Variance

Classification speed would increase if one could reduce the voluminous calculations of the 4 band representation of a fruit point in 4 dimensional space. From experimental observation, it seems that hue and intensity are independent bands for a fruit image. Hence, if a fruit were classified only on average hue and hue variance, the result would be a point in 2 dimensional space for each fruit sampled. It is hoped that this 2-D representation is a more accurate representation of a fruit than the 2-D representation with mean hue and intensity.

Using this new 2-D fruit representation on the level two clustering method, the 10 apple samples should be classed as in Figure 5.4.

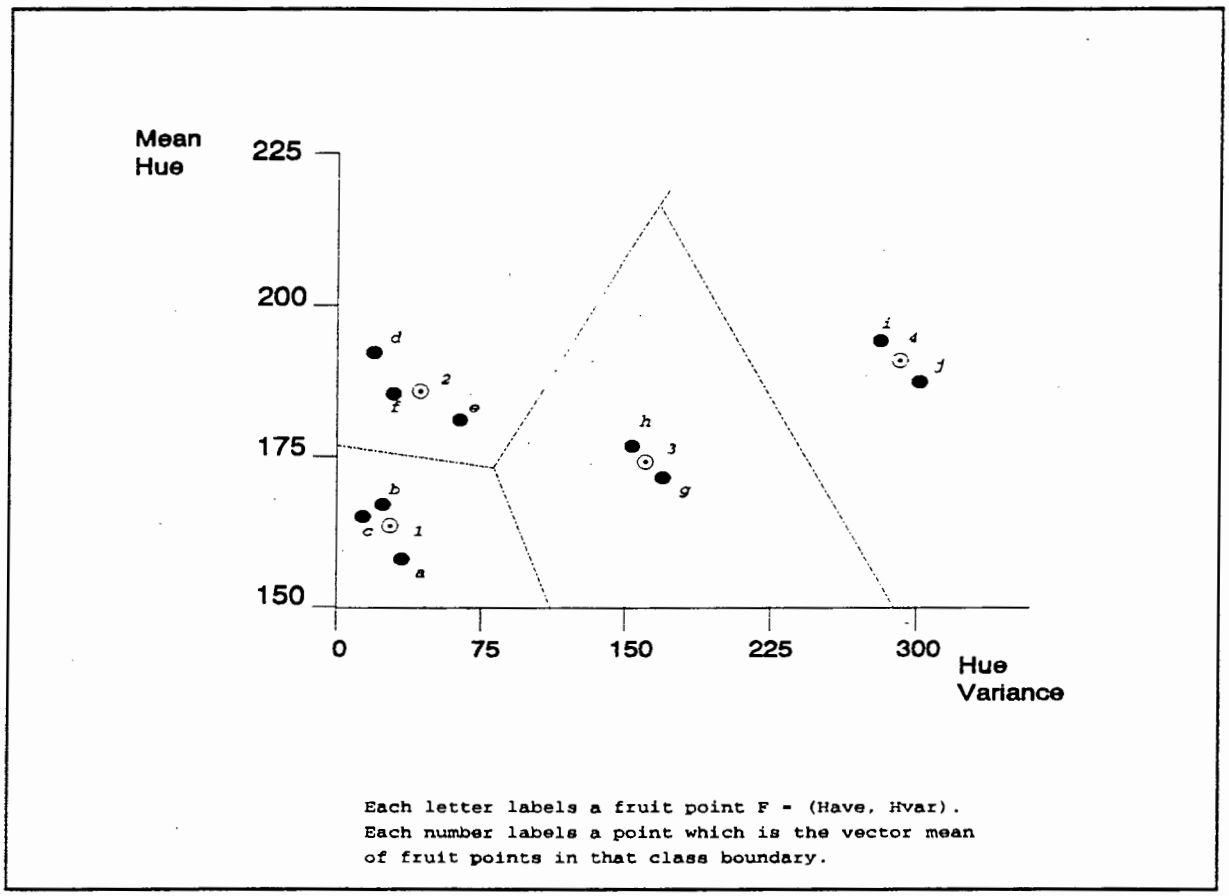
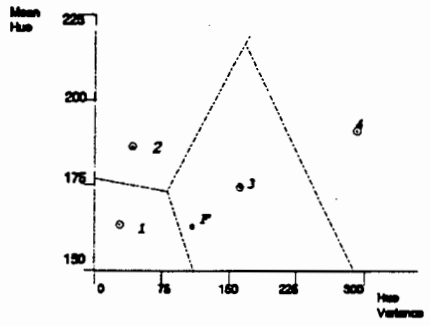
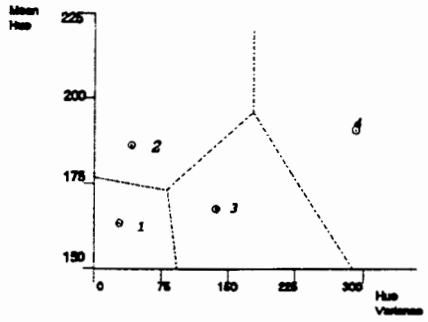


Figure 5.4: Level 2 clustering of 10 apples using the mean and variance of the hue pixel values for each fruit.

It is now possible to iteratively update the classes with more fruit samples. The K-means and ISODATA algorithms can be used for this iteration. Since only two bands are now used as compared to four previously, classification speed should increase.



a) Initially placing fruit point F in the previously formed classes, generated by K-means and ISODATA iterations.



b) New mean in class 3 is computed. From this the new class boundaries are generated. If necessary, classes affected by the new boundaries, must have there means recomputed. In this case classes remain fairly stable. The fruit is thus classed as 3 after only one iteration.

Figure 5.5: An example of classing fruit *F* (a green apple with small blemish) using Hue mean and variance, and the K-means algorithm with ISODATA.

In the above example of Figure 5.5, the attempt is to classify a green apple with a small blemish, given that training data has formed the four classes given in Figure 5.4. Although not illustrated in this example, one must remember that if the standard deviation for a certain class was too large then the vector mean for the class would be split (ISODATA step 2). The

result would be a new class. Hence by this method classes not yet considered by the user could be detected.

A few hundred apple samples can be used as training data to find as many classes as required (or as possible). The classes produced could be so well defined that only minimum Euclidean distance classification need be used on sorting subsequent fruit.

It is predicted that the following data would be adequate for use in the proposed industrial inspection system :

1. the mean hue of the fruit to be classed,
2. the hue variance of the fruit to be classed, and
3. the vector mean for each class $m_i[Have,Hvar]$.

In chapter 6 section 6.6 experimental results will reveal the actual out come of using the 3 data variables given above. If, however, further accuracy is required then the parametric Bayesian classification rule can be used (equation (5.1)). Thus the Mahalanobis distance must be used and the covariance matrix Σ_i for each class i of points must be found, where:

$$\Sigma_i = \frac{1}{n-1}\begin{pmatrix} \sum_{j=1}^{n}(Have_j - m_{i1})^2 & \sum_{j=1}^{n}(Have_j - m_{i1})(Hvar_j - m_{i2}) \\ \sum_{j=1}^{n}(Hvar_j - m_{i2})(Have_j - m_{i1}) & \sum_{j=1}^{n}(Hvar_j - m_{i2})^2 \end{pmatrix}$$

Only by experimentation, will the trade off between classification speed and classification accuracy be found. The next chapter will provide experimental results which use the minimum Euclidean distance supervised classification method and unsupervised K-means algorithm given in this chapter, to colour segment and hence classify fruit images.

5.5. Overview and Discussion

Various clustering and classification techniques have been covered in this chapter. Some methods may or may not be suitable for the grading of fruit. It has been shown that the variance and covariance matrix of pixel hues and intensities in a fruit image, play an important role in establishing accurate fruit classification.

Using the Mahalanobis distance, calculated from hue means and variances of fruit classes, seems like a method which would lead to the most accurate classification of fruit. However, this method may not be optimal for speed. The alternative is to use minimum Euclidean distances from fruit point representations (made of *Have* and *Hvar*), to class vector means.

The primary goal of clustering and classification is to classify fruit in as short a time possible, and as accurately as possible. The next chapter gives the experimental results of some of the classification methods discussed in this chapter. The experiments aim to verify the classification predictions given by some of the examples in this chapter.

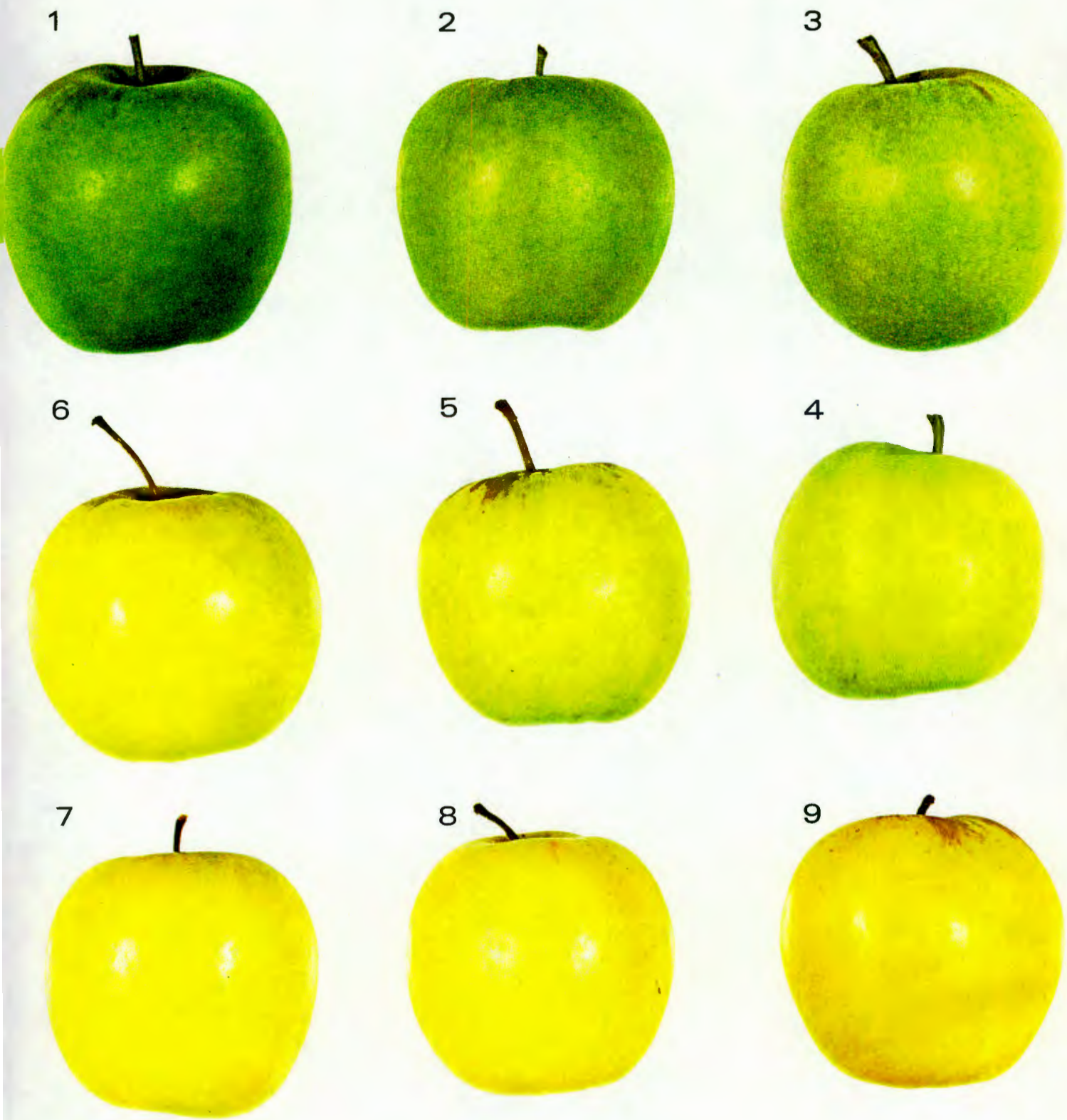
PLATE 1.

SVR
DFB

GOLDEN DELICIOUS-APPELS/APPLES

KLEUR COLOUR

STEL
SET A.28

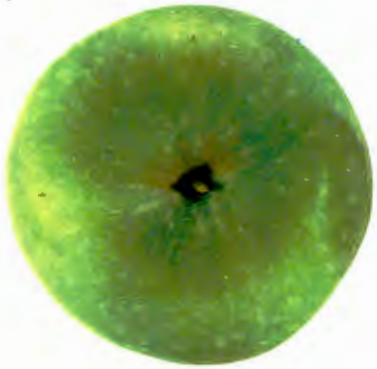


DFB

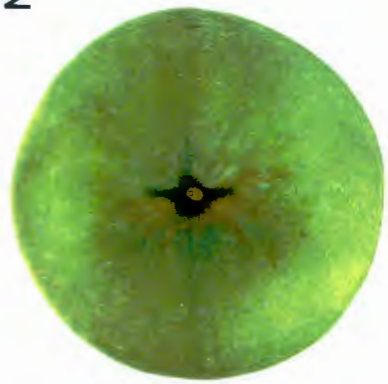
STINGELENTVERRUWING

STEM END RUSSETING

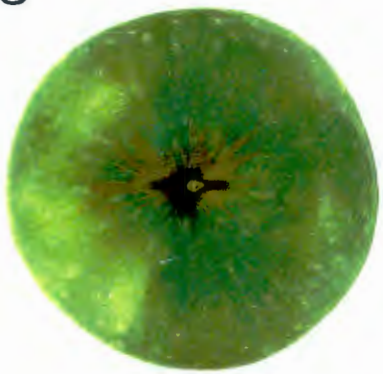
1



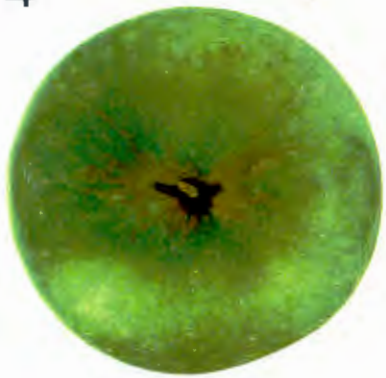
2



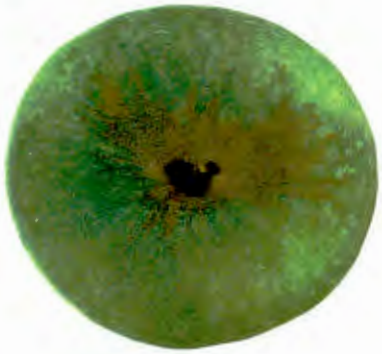
3



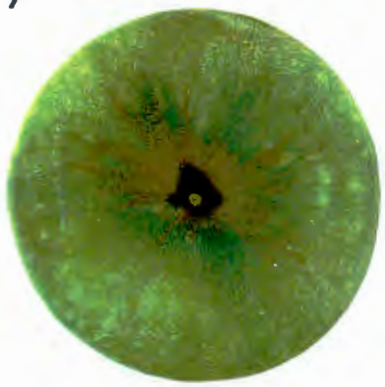
4



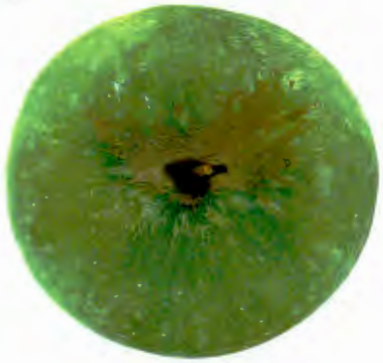
8



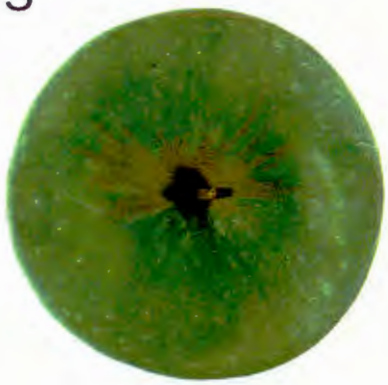
7



6

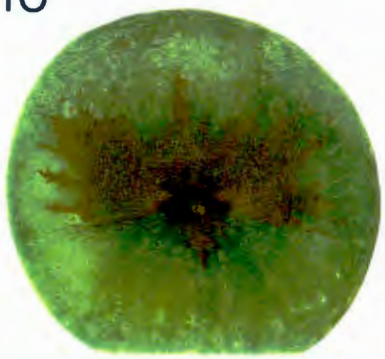


5

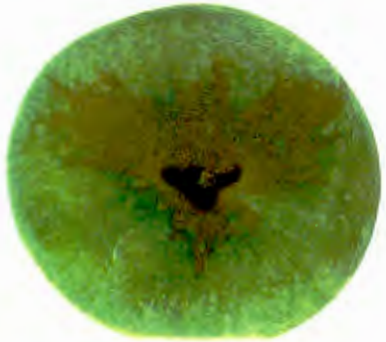


9

10



11



12

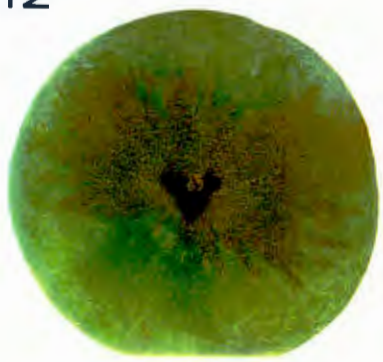


PLATE 2.

KLEUR COLOUR

PLATE 3.

CHAPTER 6

Experimental Results of Colour Classification Methods

6.1. Introduction

The experimental results given in this chapter use several apple samples to test some of the supervised and unsupervised classification theory covered in chapter 5.

In order to test unsupervised clustering classifications two software packages were investigated. Statgraphics (Statgraphics User's Guide (1987)) is a statistical graphics system and runs on IBM and compatible PC, XT and AT computers. The package is relatively easy to use but could not be used in this research as the cluster analysis programs available could only operate on small arrays of data (a feature could not have more than 140 observations). The Statistical Analysis Software (SAS) package, which is available on the UCT Vax and offers a large suite of cluster analysis functions, was selected for use (SAS/STAT User's Guide (1990)).

Two SAS functions that are of particular interest are 'Cluster' and 'Fastclus'. The 'Fastclus' function was the one used for experimentation and analysis. A listing of the SAS program developed (HICLUSMEAN.SAS) is given in Appendix B. The advantages of this function are that: it allows for each feature to contain over 10 000 observations; it performs a faster cluster classification on large data sets than any other SAS function; it performs seeded K-means clustering; initial cluster seeds can be user defined or automatically found; and the number of classes to be found must be specified.

The HICLUSMEAN.SAS program took as input, files containing the Hue (H), Saturation (S1 or S2), and Intensity (I1) features. These files were generated from the colour analysis program CAPP (see Appendix A) for fruit images of between 7000 and 30000 pixels in area. In most cases the area refers to the number of pixels within the fruit image, sampled every second line. The HICLUSMEAN.SAS program could optionally find class centroids for any

combination of the three features input. The useful output generated was a file containing the class centroids and a graphics screen scatter plot (in two or three dimensions) of selected features showing how the pixels were classed.

Unsupervised clustering gave a feel in most cases as to where class centroids might be in the pixel scatter plots. The number of classes to be found was anything from two to five.

In the supervised classification approach, several small average colour regions were defined on the fruit or on colour pixel plots, to represent corresponding class centroids. The pixels in the fruit image were then classed by minimum Euclidean distances to the user defined class centroids. Function 'classify()' in module APPCLUS.C of the CAPP software (Appendix A) implements the squared version of the minimum Euclidean distance calculation.

The colour of lighting on the fruit samples is a major factor that affects the positions of the class centroids. In this chapter the first section illustrates how lighting colour can have an affect on cluster centroids with particular reference to the change in average hue of the same fruit under different lighting colours. A simple calibration technique is given. In most of the sections that follow the feature combinations that are used to test the clustering classifications are: H only; H and I; H, S1 and I; H, S2 and I; H and S1; and H and S2.

Section 6.3 gives the experimental results of several feature combinations involved in unsupervised and supervised classification of the colours of a Granny Smith with russeting. The class centroids derived from the fruit are used on further russeting and pink blush samples.

Section 6.4 gives the results of supervised classification on nine colour types of Golden Delicious apple colours. In section 6.5 the results are given for unsupervised and supervised colour segmentation techniques on several red Starking apples. Section 6.6 gives the manual clustering results of the hue variance (Hvar) and hue average (Have) for several green apple samples with russeting, pink blush or sunburn. The final section gives a guide to the average run times for the classification results, and an indication of the errors involved in the results.

6.2. Calibrating the Lighting Conditions

The colour of light illuminating the fruit samples is of major importance if the colour distribution over a sample is to be found. In order for results to be consistent and unvarying with time, a specific standard calibration colour must be used at the beginning of a colour analysis session. A simple manual calibration technique will be given towards the end of this section.

One method to adjust the lighting conditions is by adjusting how the camera observes the conditions. On the camera is a red/blue tint adjustment knob. A blue tint tends to simulate diffused sunlight lighting conditions, and a red tint tends to simulate warm incandescent lamp lighting. A compromise between red and blue tint generally has a neutral effect. Figure 6.1 shows the effect of these tint colours on a white matte sheet of paper. One can see the colour of the white pixels distributed on the hue/saturation plane calculated from the vectors v1 and v2 (see chapter 4).

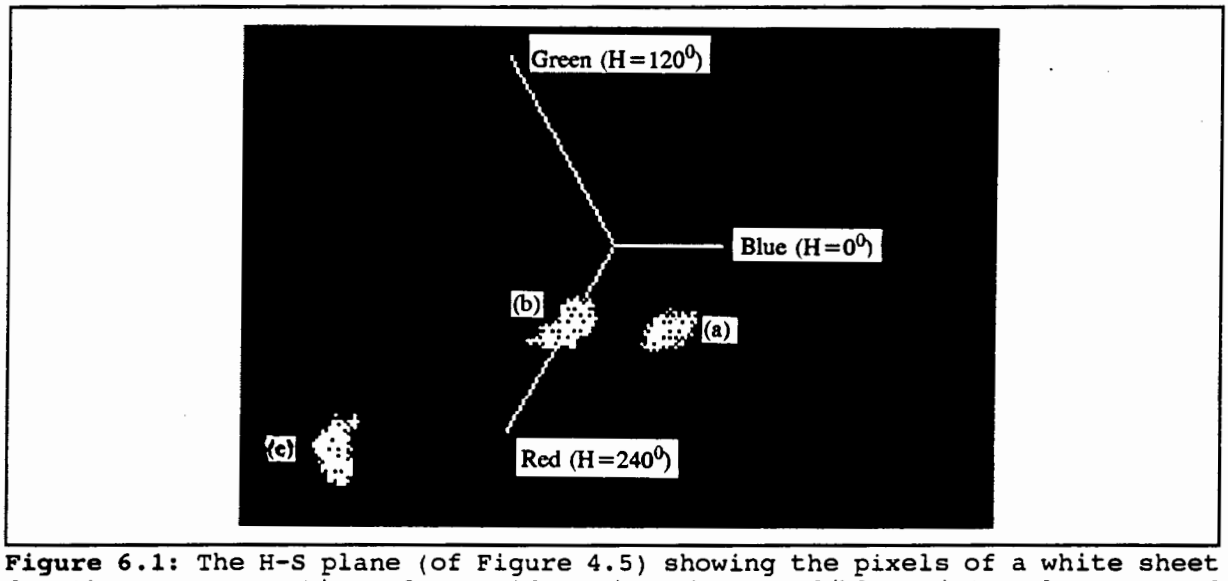


Figure 6.1: The H-S plane (of Figure 4.5) showing the pixels of a white sheet for the camera setting of a) a blue tint, b) a red/blue tint and, c) a red tint.

The effect of observing colour objects under a blue or red/blue tint are shown in Table 6.1. This table gives the average hue, average saturation (calculated as a vector distance) and average intensity (calculated as the average of the RGB components) for the 9 golden delicious colour standards given in Plate 1. One can see that a blue tint tends to decrease

saturation and intensity levels but increase the hue range (154^0 to 184^0) over the 9 fruit colours, whereas the red/blue tint generally gives larger saturation and intensity averages and a smaller hue range (178^0 to 201^0).

Table 6.1: Average HSI values for 9 class colours of Golden Delicious viewed under a blue and a red/blue tint.

Golden Delicious colour class	Average colour component of fruit viewed under a blue tint			Average colour component of fruit viewed under a red/blue tint		
	H	S	I	H	S	I
1	154	84	51	178	96	107
2	164	83	63	186	110	125
3	168	61	53	190	114	140
4	169	79	86	192	132	151
5	174	91	99	194	139	150
6	179	115	97	196	147	147
7	181	115	96	197	160	150
8	182	115	98	200	164	159
9	184	110	76	201	177	153

All the experiments that follow were conducted using the red/blue tint. This is because even though the hue range is small for red, yellow and green images under this tint condition, the experimental results showed that colour segmentation is possible under lighting conditions that are not completely ideal.

A simple method to manually calibrate for a set colour lighting condition for each experimental run, is as follows:

1) Capture an image of a white sheet with the tint knob at a red/blue setting.

2) Obtain the average H, S and I components for the region that will be used in subsequent experimental colour analysis.

3) Compare the hue average to an originally set hue value.

If the hue average is lower than the set value (i.e. more red) then adjust the tint knob slightly towards the blue end and repeat from step 1.

If the hue average is higher than the set value (i.e. more blue) then adjust the tint knob slightly towards the red end and repeat from step 1.

If the hue average is within a desired proximity of the set hue value then calibration is complete.

Obviously the above calibration method can be made to incorporate the saturation and brightness adjustments on the camera. Eventually the calibration method could be automated using a hardware control loop.

The experimental results in this chapter use a white calibration setting of 241^0 for hue, 28 for saturation and 196 for intensity.

To classify the grade of a fruit one method would be to observe the percentage distribution of a certain colour. Hence, if one observed the percentage distribution of the brown from russeting, a small value would indicate little russeting, and a large value would indicate a large russeting size. In addition a concentration factor giving the largest peripheral length of each class observed, would indicate if the class were concentrated in one region (large concentration factor) or scattered over the image (small concentration factor). In the results that follow, percentage distributions will be given for each class, and in most cases, figures containing a colour segmented image will show the concentration or distribution of the class.

It should be noted that all images used in the experiments were taken from a windowed region at the top centre of the scene viewed by the camera. The light at the base of this 188x229 pixel window was slightly brighter than at the top. In general this uneven lighting did not affect the experimental results except in certain cases which used the intensity (I) or saturation (S1) features. In these cases it will be stated how the lighting affected the results.

6.3. Colour Classification of Granny Smiths with Russeting

In this section, unsupervised and supervised classification results are given for a sample Granny Smith with russeting (fruit 7 on Plate 2). In all cases, four classes were chosen for colour segmentation of the fruit image. Four classes were used since the human eye could easily distinguish four colour types on the fruit, namely the green and yellow on the fruit skin, brown on the russeting, and dark brown at the stalk.

6.3.1. Unsupervised Colour Segmentation of a Granny Smith with Russeting

The following unsupervised clustering results were obtained using the SAS program whose listing is given in Appendix B. The unsupervised clustering technique is a K-means approach using automatically detected centroid seeds. Four features are used in six different combinations to show how segmentation is affected by the feature combinations. The features used are H, S1 (saturation calculated as a vector distance), S2 (saturation as ratio of the minimum RGB component to intensity I), I (intensity as the average of RGB).

Table 6.2 gives the centroid coordinates for various feature combinations (H, H-I, H-S1-I, H-S1, H-S2, H-S2-I) and percentage distributions for each of the four classes. Figure 6.2 shows the effect of colour segmentation on Granny Smith 7 (Plate 2) for each case. Figure 6.3 shows how the classes are distributed on the pixels for the H-I plane.

Table 6.2: Unsupervised class centroids and percentage distribution of colour classes for Granny Smith 7 with russeting (Plate 2)

Colour class number	a) Class centroids using one feature and percentage coverage of each class (Figure 6.2a)		b) Class centroids using two features and percentage coverage of each class (Figure 6.2b)			c) Class centroids using three features and percentage coverage of each class (Figure 6.2c)			
	H	% area	H	I	% area	H	S1	I	% area
1	175	32.5	174	100	25.6	175	88	104	32.8
2	180	54.2	180	123	59.5	180	115	124	51.5
3	198	11.6	198	91	13.5	198	84	91	14.0
4	223	1.8	223	39	1.5	222	43	41	1.6

Colour class number	d) Class centroids using two features and percentage coverage of each class (Figure 6.2d)			e) Class centroids using two features and percentage coverage of each class (Figure 6.2e)			f) Class centroids using three features and percentage coverage of each class (Figure 6.2f)			
	H	S1	% area	H	S2	% area	H	S2	I	% area
1	177	92	45.2	180	134	40.2	180	155	116	68.9
2	180	122	37.9	181	164	56.9	182	116	112	17.3
3	196	83	15.1	216	109	2.7	200	140	78	13.5
4	222	43	1.8	223	238	0.2	224	221	24	0.3

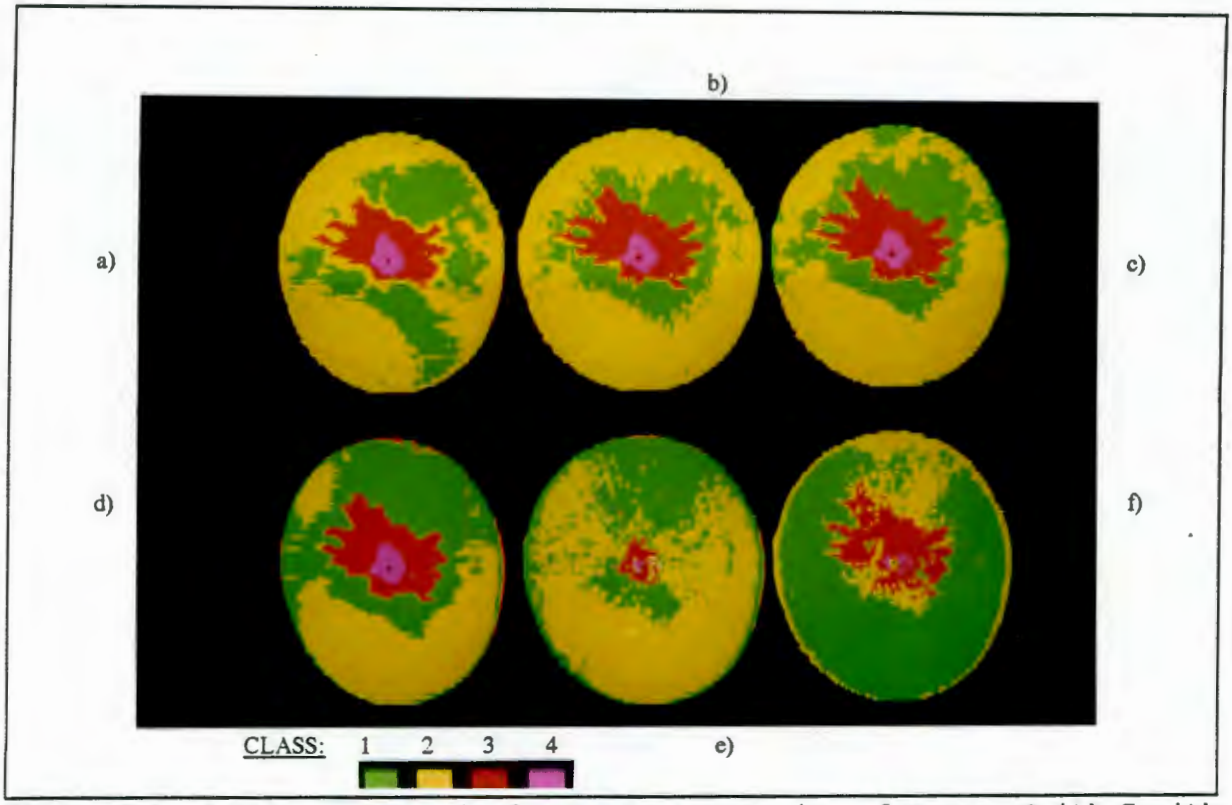


Figure 6.2: 4 class unsupervised colour segmentation of Granny Smith 7 with russeting (Plate 2) for each of the feature combinations a) H b) H-I c) H-S1-I d) H-S1 e) H-S2 f) H-S2-I

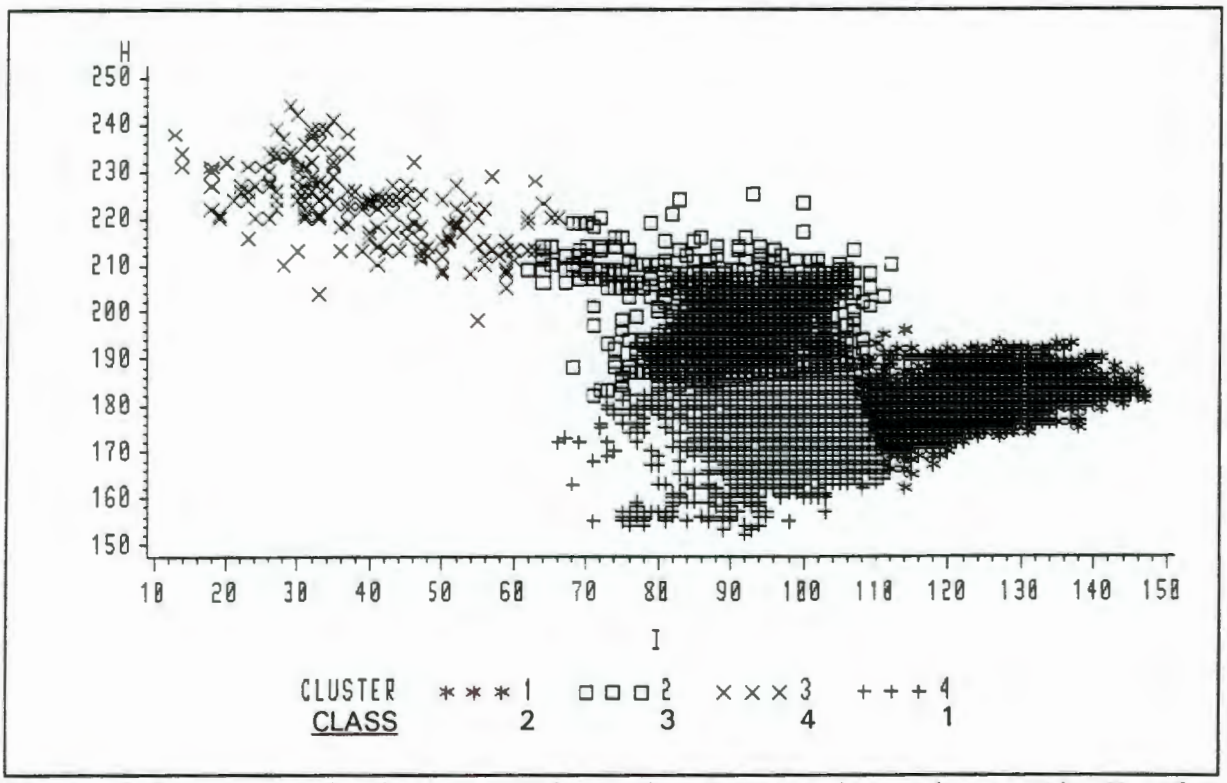


Figure 6.3: Unsupervised segmentation using K-means iteration on the H and I features of pixels in Granny Smith 7 (Plate 2). The class centroids are given in Table 6.2b.

Observations for unsupervised classification of Granny Smith 7

The cluster classes shown in the scatter plot of Figure 6.3 correspond to the classes found by the FASTCLUS function in the program listing given in Appendix B. The four classes clearly define the colour pixels for the H-I colour segmented image (Figure 6.2c). Note that the unsupervised CLUSTER numbers (Figure 6.3) have been changed to appropriate CLASS numbers (Figure 6.2), where CLUSTER 1 was changed to CLASS 2, CLUSTER 2 was changed to CLASS 3, CLUSTER 3 was changed to CLASS 4, and CLUSTER 4 was changed to CLASS 1.

In order to determine the best feature combination that most accurately segments a fruit image, one must compare the fruit image of Granny Smith 7 on Plate 2 to the colour segmented images in Figure 6.2. All feature combinations except those that use the S2 feature, accurately segment out the stalk region with class 4. From the figure the H-S2 feature combination gives the worst segmentation as the russeting region is not detected. The other 5 feature combinations do detect the russeting region with class 3. One can see that class 3 most accurately defines the russeting region and boundary using feature combinations H, H-I and H-S1-I. These three feature combinations also allow for class 2 to detect the yellow part of the fruit and class 1 to define the greener part of the fruit. The accuracy to which the hue (H) feature defines the green/yellow boundaries and the accuracy to which the H-I or H-S1-I features define these boundaries, is very subjective. Feature combinations using intensity I (namely Figure 6.2 b) and c)) allow for an accurate representation of the apple boundary, whereas the other feature combinations may include an incorrect detection of class 3 at the fruit boundary.

From the unsupervised classing of Granny Smith 7, one may conclude that H-I or H-S1-I feature combinations allow for the most accurate segmentation of the image into four classes. The H feature alone and H-S1 combination are also fairly good for segmentation but not as accurate as those feature combinations which include the I feature. Feature combinations using S2 are generally not very accurate for colour segmentation that best represents how the human eye would segment the image.

6.3.2. Supervised Colour Segmentation of a Granny Smith with Russeting

One can manually select the centroids by eye estimation of where class centroids are on the H-I or H-S plots or on the H histogram. These centroid estimates in most cases are just as good as those found by the SAS program given in the appendix. In the following experiment a pixel plot of the H versus I features of each pixel in the Granny Smith 7 image was displayed on the RGB Philips monitor. The H and I features were recorded for four regions that best represented class centres to the eye. Table 6.3 shows these class centroid values and the corresponding percentage distribution of the class over the fruit image (Granny Smith 7 on Plate 2). Figure 6.4 shows the four class distributions over the fruit image.

Table 6.3: Manually selected class centres from the H-I pixel plot of pixels in Granny Smith 7 (Plate 2)

Colour class numbers	Class centroids using two features and percentage coverage of each class (Figure 6.4)		
	H	I	% area
1	175	106	42.8
2	183	131	42.3
3	202	92	13.6
4	231	30	1.3

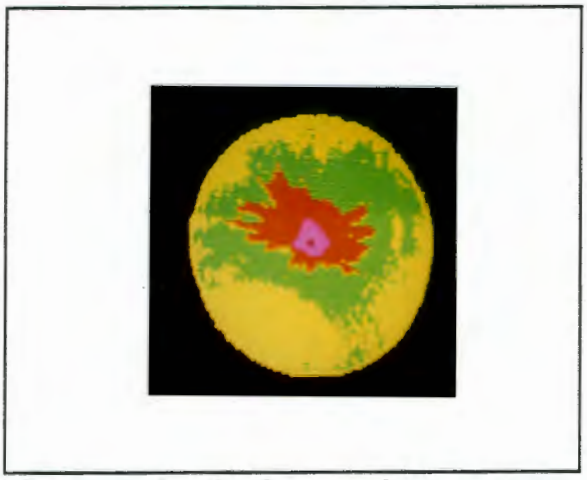


Figure 6.4: 4 class colour segmentation of Granny Smith 7 using supervised centroids from an H-I pixel plot given in Table 6.3.

If one compares the results of Table 6.3 to the SAS program generated H-I centroids of Table 6.2b, the class centroids are in fact quite similar. The segmented images for these two H-I results are also similar, with the russeting boundary being equally well defined. For this case the results of supervised and unsupervised classification have shown to be similar.

Another method to manually find cluster classes (supervised classification) is to select small regions on the actual image, that contain the class colour required. In this case four regions were selected, namely: a green region (class 1); a yellow region (class 2); a brown russeting region (class 3); and a dark stalk end region (class 4). These sample regions are 25 pixels in area and result in centroids that are calculated from the average of the feature components for

that area. Table 6.4 gives the results of percentage area distribution of these classes for H-I, H-S1-I, and H-S2-I feature combinations. Figure 6.5 shows the distributions of the classes on the Granny Smith 7 image. Note that the class regions selected for the centroids are superimposed on the colour segmented images shown in the figure.

Table 6.4: Supervised class centroids direct from the image of Granny Smith 7 with russeting (Plate 2) and percentage distribution of colour classes

Colour class number	a) Class centroids using two features and percentage coverage of each class (Figure 6.5a)			b) Class centroids using three features and percentage coverage of each class (Figure 6.5b)				c) Class centroids using three features and percentage coverage of each class (Figure 6.5c)			
	H	I	% area	H	S1	I	% area	H	S2	I	% area
1	162	97	26.2	162	75	97	27.5	162	125	97	24.0
2	180	135	55.7	180	134	135	45.8	180	167	135	52.4
3	199	91	16.7	199	84	91	25.3	199	147	91	22.7
4	235	33	1.4	235	28	33	1.5	235	86	33	0.9

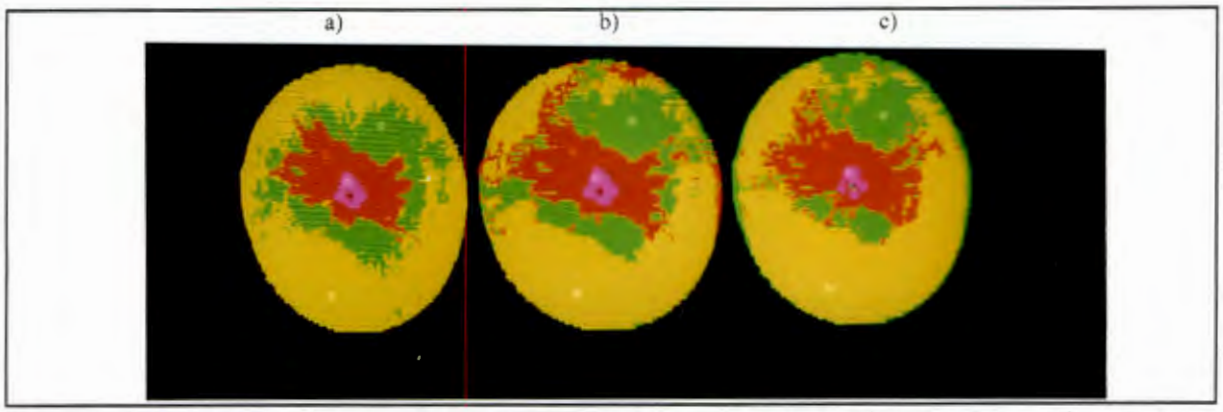


Figure 6.5: Segmentation into 4 supervised classes direct from the Granny Smith 7 image. Class centroids contain the feature combinations a) H-I b) H-S1-I and c) H-S2-I

Observations for supervised classification of Granny Smith 7

The three segmented images shown in Figure 6.5 reveal that manually selecting the 4 classes yields acceptably segmented colours for the Granny Smith 7 skin. Segmentation using the H-I feature combination gives very similar results to segmentation using H-I centroids manually chosen from the H-I scatter plot, and unsupervised H-I detected centroids. Segmentation using H-S1-I features (Figure 6.4b) shows too much incorrectly classed class 3 regions and hence appears to be less accurate than the unsupervised result in Figure 6.2c. However, segmentation using the H-S2-I features appears to produce an adequate colour classing in the

supervised case (Figure 6.4c) compared with the unsupervised case (Figure 6.2f). In general, however, the H-I feature combination still appears to produce the most accurately segmented image of Granny Smith 7.

6.3.3. Testing the Unsupervised Class Centroids of Granny Smith 7 with Russeting on Other Fruit Samples

This subsection aims to show that the colour classes set up for one fruit can work sufficiently well on other fruit, for colour segmentation purposes. Only the H and HI feature combinations are used to illustrate this. The class centroids generated by unsupervised classification of Granny Smith 7 (Plate 2) are used on two other Granny Smiths on that Plate namely, Granny Smiths 1 and 12. These fruit have smaller and larger russeting patches than Granny Smith 7, respectively. The class centroids are also tested on the Granny Smith with pink blush shown in Figure 4.11c. Table 6.5 gives the percentage distributions of each of the four colour classes on each of the three fruit. Figure 6.6 shows some of the colour segmented fruit images based on the results in Table 6.5.

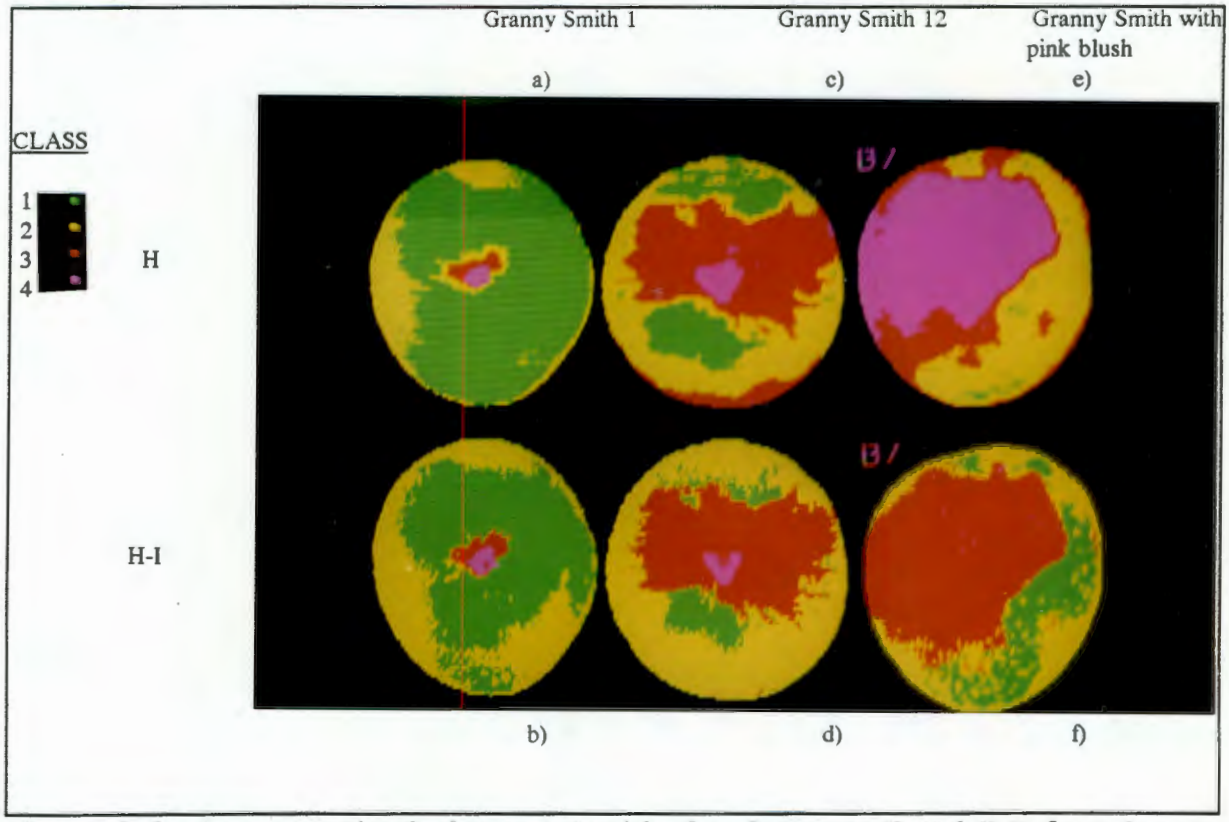


Figure 6.6: 4 unsupervised class centroids for features H and H-I from Granny Smith 7 used to segment three fruit images: Granny Smiths 1 and 12 and Granny Smith with pink blush (see Figure 4.11c).

Table 6.5: Unsupervised class centroids from Granny Smith 7 with russeting (Plate 2) used on Granny Smiths 1 and 12 and Granny Smith with pink blush (Fig. 4.11c)

a) Granny Smith 1 (Plate 2)	i) Class centroids using one feature and percentage coverage of each class (Figure 6.6a)		ii) Class centroids using two features and percentage coverage of each class (Figure 6.6b)		
Class Number	H	% area	H	I	% area
1	175	81.7	174	100	45.4
2	180	15.5	180	123	51.4
3	198	2.1	198	91	2.4
4	223	0.7	223	39	0.8

b) Granny Smith 12 (Plate 2)	i) Class centroids using one feature and percentage coverage of each class (Figure 6.6c)		ii) Class centroids using two features and percentage coverage of each class (Figure 6.6d)		
Class Number	H	% area	H	I	% area
1	175	19.5	174	100	11.0
2	180	34.9	180	123	48.7
3	198	42.8	198	91	39.1
4	223	2.8	223	39	1.2

c) Granny Smith with pink blush (Figure 4.11c)	i) Class centroids using one feature and percentage coverage of each class (Figure 6.6e)		ii) Class centroids using two features and percentage coverage of each class (Figure 6.6f)		
Class Number	H	% area	H	I	% area
1	175	0.6	174	100	3.2
2	180	29.7	180	123	47.9
3	198	20.5	198	91	48.8
4	223	49.1	223	39	0.2

Observations for unsupervised class centroids on different Granny Smith apple images

If one looks at Figure 6.6 and Plate 2 then one will notice that Granny Smiths 1 and 12 are fairly well colour segmented using only the hue (H) feature. The H-I feature combination tends to allow for a more accurate colour segmentation (i.e. an accurate class boundary definition) of the images. In Figure 6.6e the pink blush region is accurately defined by the magenta of class 4 using only the H feature. However, the H-I feature combination segments the pink blush region accurately with the red of class 3. One can see that the classes set up from using the colours from a fruit with russeting are sufficient for detecting pink blush regions if one considers pink blush to be a similar blemish to russeting. Clearly if more classes were defined, then pink blush could be described by segmentation classes as being different to the russeting colour.

6.4. Colour Classification of 9 Golden Delicious Classes

It has been shown in the previous results on fruit with russeting and pink blush, that the hue (H) feature alone produces sufficient colour segmentation. Adding more features in some cases simply increases classing accuracy. In addition unsupervised classification is not necessary if classes can be visually determined.

It was found that unsupervised K-means classification of the 9 Golden Delicious colour classes (Plate 1) generally produced unsatisfactory results using feature combinations such as H-I and H-S1-I. However, classification into 9 classes using H alone resulted in the 9 classes being adequately distinguished.

In this section only the results of supervised classification into the 9 colour classes will be described. This is because obtaining supervised classes is faster than the unsupervised approach, and the results for supervised classing should prove that this approach is sufficient to determine the 9 colour classes.

Each supervised class was taken by averaging the features (H, S1, and I) associated with all pixels sampled in each Golden Delicious of Plate 1. This produced nine classes made up of three features. The class centroids are given in Table 6.1b, which are the average feature components of each of the 9 Golden Delicious fruit viewed under a red/blue tint. A further class (class 10) was made in order to account for stalk colour. This class was taken from measuring the average stalk colour of Golden Delicious 9 (Plate 1). The 10 class centroids are given in Table 6.6. Table 6.7 shows the results of these class distributions over the 9 Golden Delicious images, for feature combinations of H, H-I and H-S1-I. Figure 6.7 shows the colour segmented images for three of these 9 images under the three different feature combinations. Feature combinations of H and S1 or H and S2 were not used because of the lighting dependence of pixel saturation values. Already the H-S1-I feature classes shown on the segmented images in Figure 6.7c reveal the effect of the glossy highlights on the colour apple chart in Plate 1.

Table 6.6: Supervised class centroids for 10 class colours to be used in classifying the 9 Golden Delicious apples in Plate 1.

Colour class numbers	Average feature components of class colour viewed under a red/blue tint		
	H	S	I
1	178	96	107
2	186	110	125
3	190	114	140
4	192	132	151
5	194	139	150
6	196	147	147
7	197	160	150
8	200	164	159
9	201	177	153
10	248	48	92

Observations for supervised classification of Golden Delicious images

Table 6.7 shows a general trend, that the highest class percentage distributions usually correspond to the Golden Delicious with that same class number. For example, Golden Delicious 2 has the highest percentage distribution of class two (segmentation class colour yellow in Figure 6.7a), i.e. 80.6% using the H feature, 59.9% using the H-I feature combination, and 54.9% using the H-S1-I feature combination for colour segmentation of the image. This trend is best shown when only the H feature is used for colour segmentation. One can see from Figure 6.7a that using H only results in an adequate classing of Golden Delicious 2, 5 and 8. The H only feature tends to be only slightly sensitive towards the fruit image edge (about 6 pixels in from the defined boundary) and slightly sensitive at the glossy highlights. However, considering that the hue range between the 9 fruit classes is small (178^0 to 201^0) and that some classes only differ by one degree of hue, Table 6.7 and Figure 6.7a show that the H feature alone is adequate for distinguishing between the classes.

Table 6.7: Colour segmentation results for the 9 fruit images of Plate 1, using percentage class distributions for three feature combinations (H, H-I, and H-S1-I)

Fruit classed from Plate 1	Feature combinations used from the classes in Table 6.6.	Percentage [%] area of each class covered on the fruit image. (NOTE: pixels from the images are sampled every 2 lines.)									
		1	2	3	4	5	6	7	8	9	10
Golden Delicious 1	H	91.4	3.4	0.9	0.8	0.7	0.2	0.3	0.3	2.0	0.1
	H-I	87.2	9.6	1.9	0.4	0	0.1	0	0	0	0.7
	H-S1-I	73.4	23.5	0.8	0	0	0	0	0	0	2.3
Golden Delicious 2	H	0.1	80.6	14.7	2.0	1.2	0.3	0.5	0.3	0.3	0
	H-I	16.9	59.9	19.3	2.4	0.3	0.7	0.2	0.1	0.1	0.1
	H-S1-I	13.8	54.9	21.9	3.2	3.2	3.0	0	0	0	0.3
Golden Delicious 3	H	0	1.8	51.6	31.7	7.6	1.4	1.7	0.9	3.3	0
	H-I	0.6	25.0	52.4	8.1	3.6	5.4	0.9	3.0	0.5	0.4
	H-S1-I	1.9	18.2	39.7	17.1	12.1	10.1	0	0	0	1.0
Golden Delicious 4	H	0	0	37.0	48.3	8.5	1.8	2.1	1.1	1.1	0
	H-I	0	0.8	30.8	42.3	8.4	9.5	1.3	6.3	0.7	0
	H-S1-I	0.2	1.5	21.8	19.4	21.7	23.0	12.0	0	0.1	0.4
Golden Delicious 5	H	0	0	0.9	19.5	51.7	12.6	7.9	3.2	4.0	0.2
	H-I	0.6	6.7	27.6	10.4	18.3	25.2	2.9	6.2	1.6	0.4
	H-S1-I	0.8	3.3	13.2	7.7	9.3	35.5	28.2	0.4	0.9	0.7
Golden Delicious 6	H	0	0	0	0	14.5	35.8	35.9	7.8	5.8	0.1
	H-I	0.3	10.9	23.4	0.1	2.0	47.5	8.2	5.7	1.7	0.4
	H-S1-I	1.2	5.8	8.8	4.2	3.6	22.4	24.8	0	28.2	0.9
Golden Delicious 7	H	0	0	0	0	0	0.9	47.4	34.5	17.0	0.2
	H-I	0	4.6	11.3	0	0	30.4	27.7	11.0	14.9	0.2
	H-S1-I	0.1	1.1	7.6	5.2	2.9	8.5	22.0	2.0	50.3	0.2
Golden Delicious 8	H	0	0	0	0	0	0	6.1	47.6	45.7	0.6
	H-I	0	0.5	5.2	0	0	21.3	7.6	36.3	28.4	0.7
	H-S1-I	0.1	0.25	5.2	5.8	3.4	3.3	4.9	10.2	66.2	0.7
Golden Delicious 9	H	0	0	0	0	0	0	7.5	37.7	54.4	0.4
	H-I	0	0.4	2.5	0	0	29.0	19.0	14.1	34.6	0.4
	H-S1-I	0	0.1	3.3	4.0	2.4	3.6	6.1	3.4	76.5	0.5

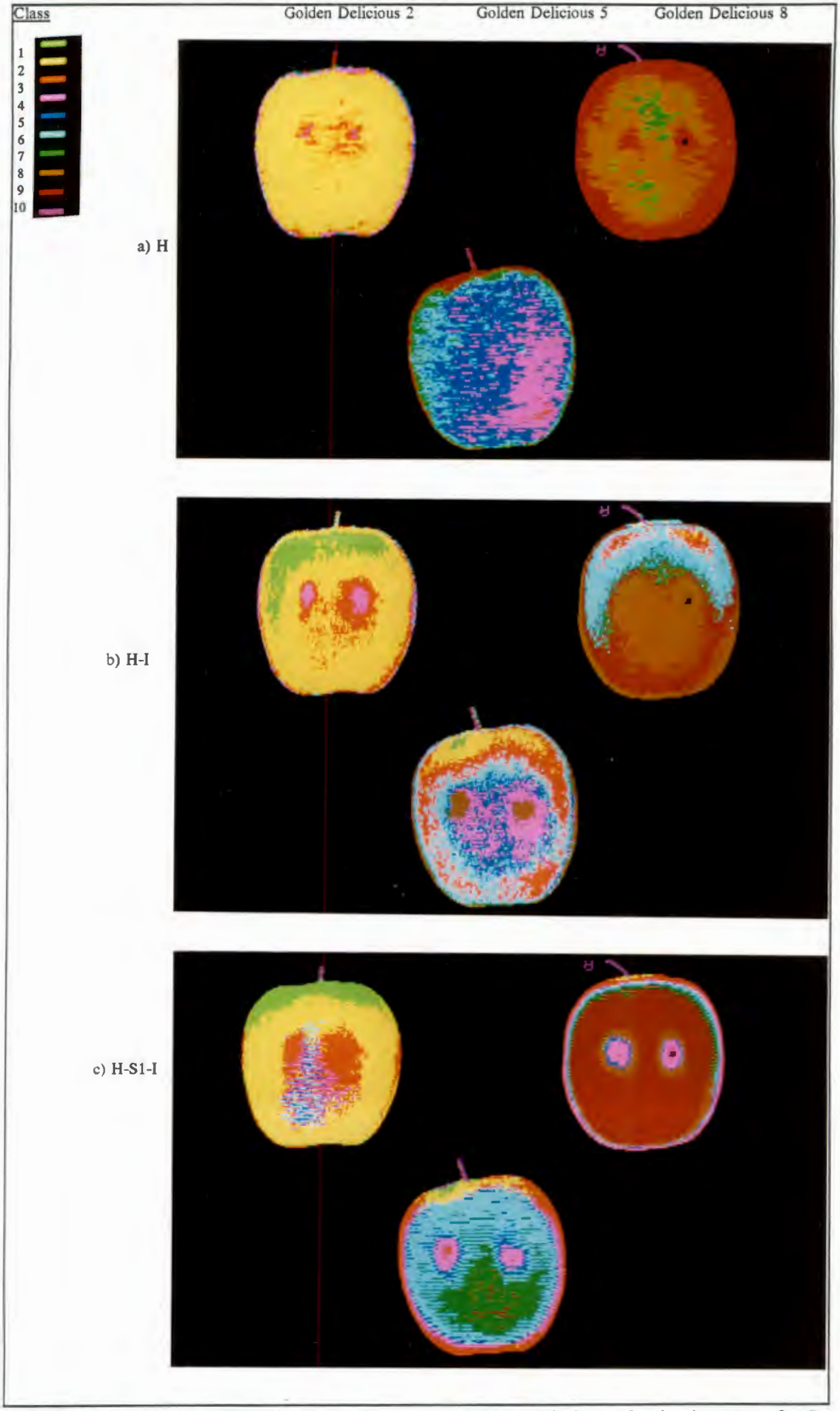


Figure 6.7: Colour segmentation of three Golden Delicious fruit images (2, 5, and 8) from Plate 1 into 10 classes. a) H classification b) H-I classification c) H-S1-I classification

On introducing the I feature, H-I segmentation becomes more sensitive to the glossy highlights and the non-uniformity of the light over the fruit (Figure 6.7b). With the saturation component, H-S1-I segmentation clearly defines the gloss and shows an undefined boundary around the fruit image at an average of 15 pixels from the edge. One can see, however, that the gloss patches and fruit boundary were ignored then the majority of pixels in each of the 3 apple images chosen would correspond to the class chosen for the fruit.

6.4.1. Testing the Supervised Class Centroids from the 10 Golden Delicious Colours on a Real Fruit Sample

In this subsection the 10 classes given in Table 6.6 were used to try and classify a real Golden Delicious apple with russeting. The fruit image used in this experiment is shown in chapter 4 Figure 4.11b. As before, three feature combinations (H, H-I, and H-S1-I) are used to show that colour segmentation varies according to what feature combinations are used. Table 6.8 shows the percentage distributions of each colour class over the real fruit image, and Figure 6.8 illustrates these class distributions on colour segmented images of the real Golden Delicious fruit image.

Table 6.8: Percentage class distribution results for the real Golden Delicious fruit image in Figure 4.11b, segmented with class centroids from Table 6.6

Feature combinations used from the classes in Table 6.6.	Percentage [%] area of each class covered on the fruit image. (NOTE: pixels from the images are sampled every 2 lines.)									
	1	2	3	4	5	6	7	8	9	10
H	0	9.0	29.1	25.5	13.9	3.4	4.4	3.7	10.7	0.3
H-I	19.2	25.5	28.9	8.8	6.3	6.3	0.4	0.7	0.1	3.2
H-S1-I	7.3	9.8	6.0	1.5	1.2	11.1	12.0	0	47.3	3.8

Observations for supervised classification of a real Golden Delicious image

Table 6.8 and Figure 6.8 show clearly that a fruit image can be classed quite differently depending on what features are used. The H feature suggests the fruit is mainly class 3 with a good coverage of class 4. The blue areas of class 5 (Figure 6.8a) generally indicate the

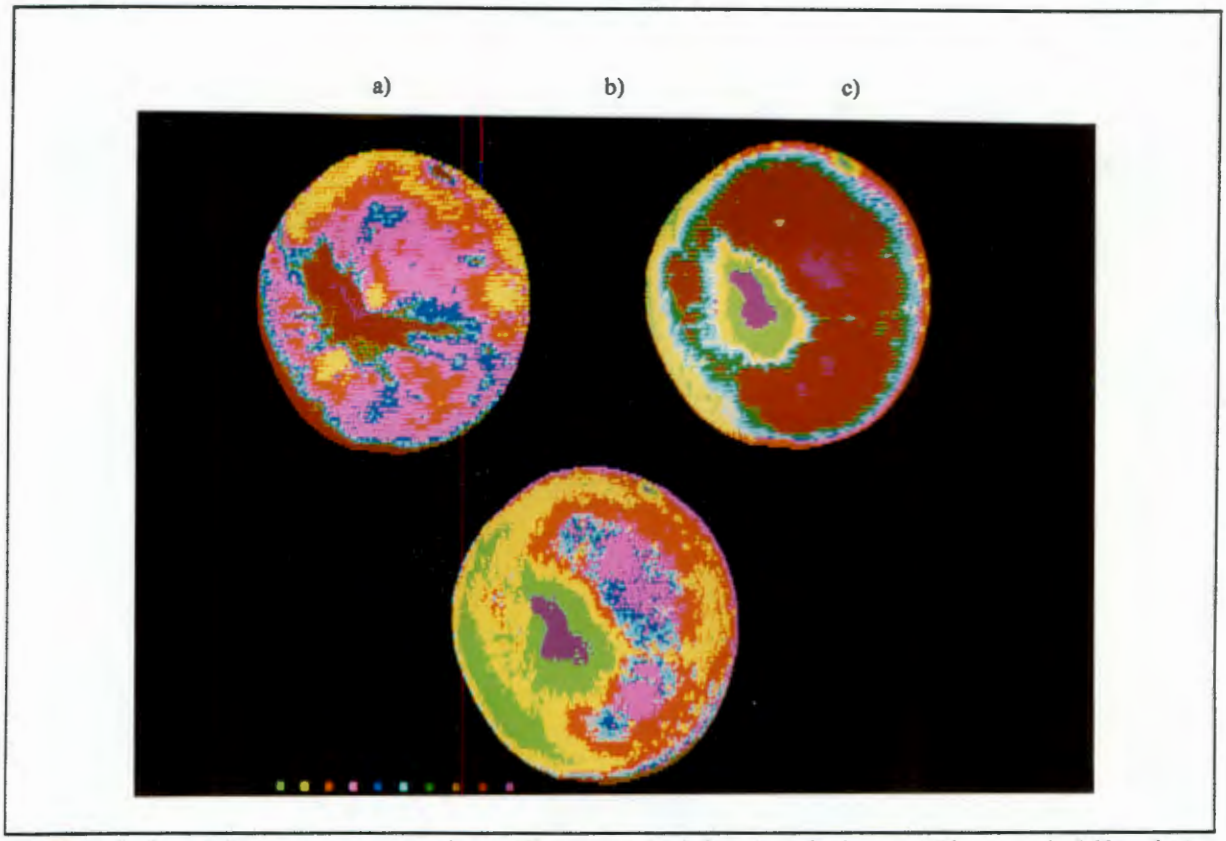


Figure 6.8: Colour segmentation of a real Golden Delicious (Figure 4.11b) into 10 classes using supervised class centroids consisting of features a) H, b) H-I and c) H-S1-I

brown markings on the fruit. The 10.7% of class 9 refers to the russeting area and small shadow region on the lower left of the fruit.

The H-I feature combination clearly defines the stalk as class 10, but now the russeting and shadow is indicated as class 1. Table 6.6, which contains the class centroids used for this segmentation, shows that class 1 (the greenest class) has the lowest intensity. Hence, since, the russeting and shadow are of lowest intensity they are classed by the H-I feature combination as class 1. Figure 6.8b also shows that the region on the fruit nearest the camera, has the greatest intensity (classes 4, 5 and 6). It should be noted that for this case the intensity feature is dominating over the hue feature for this minimum Euclidean distance segmentation using the H and I components.

Figure 6.8c shows that the lighting was quite bright on the fruit, which lead to most of the fruit being a saturated colour. Since Table 6.6 shows that the class centroid for highest saturation is class 9 (S1 = 177), 47.3% of the fruit is covered with this class.

This experiment shows that the present lighting distribution is not adequate for accurate colour segmentation, if the I and S1 components are to be used. In this case the colour segmentation given by the H feature alone should result in the most accurate colour classing of the fruit pixels.

Since the lighting conditions are different for the real fruit compared with the lighting on the fruit colour charts, the intensity and saturation components generally contribute to a colour classification of the real fruit that is lighting dependent and therefore inaccurate.

Out of five human observers, three classed the real fruit colour as being similar to Golden Delicious 4, one classed the real fruit as Golden Delicious 5, and one classed the fruit as Golden Delicious 6. The comparisons were done under fluorescent light (CWF). The hue feature classed the fruit as mainly class 3 and 4 under a mixture of incandescent and tungsten halogen light. Generally one should not compare the results of the human to the machine classification. This may be because lighting conditions are different so comparing human and machine classifications should be meaningless, also the machine classification would always be consistent in classing colours under a specific lighting condition.

6.5. Colour Classification of Red Starking Classes

In this section, unsupervised and supervised classification results are given for a sample Red Starking apple (fruit 6 on Plate 3). In the unsupervised cases, four classes were chosen for colour segmentation of the fruit image. This was to see if unsupervised classification could detect more than the two main colours the human eye could detect. In the supervised cases only two classes were chosen, namely one representing the greens and yellows on the fruit (class 1) and one representing the reds on the fruit image (class 2). These supervised classes are then used to colour segment other Starking images.

6.5.1. Unsupervised Colour Segmentation of a Red Starking

The following results were obtained using the SAS program whose listing is given in Appendix B. The unsupervised clustering technique is a K-means approach using automatically detected centroid seeds. Three features are used in four different combinations to show how segmentation is affected by the feature combinations. The features used are H, S1 (saturation calculated as a vector distance), and I (intensity calculated as the average of RGB).

Table 6.9 gives the centroid coordinates for the various feature combinations (H, H-I, H-S1, and H-S1-I) and percentage distributions for each of the four classes on the Starking 6 image (Plate 3). Figure 6.9 shows the effect of colour segmentation on Starking 6 for each case. Figure 6.10 shows how the classes are distributed for the fruit image pixels on the H-I plane.

Table 6.9: Unsupervised class centroids and percentage distribution of colour classes for Starking 6 (Plate 3)

Colour class number	a) Class centroids using one feature and percentage coverage of each class (Figure 6.9a)		b) Class centroids using two features and percentage coverage of each class (Figure 6.9b)			c) Class centroids using two features and percentage coverage of each class (Figure 6.9c)			d) Class centroids using three features and percentage coverage of each class (Figure 6.9d)			
	H	% area	H	I	% area	H	S1	% area	H	S1	I	%area
1	213	29.4	216	124	25.4	219	124	63.8	219	121	110	57.8
2	222	48.1	220	104	38.9	225	112	30.5	225	90	117	7.7
3	231	22.1	227	81	17.6	229	80	5.1	226	120	87	31.6
4	248	0.4	230	99	18.1	243	53	0.6	239	63	95	1.0

Observations for unsupervised classification of Starking 6

Note that the H-I plot (Figure 6.10) gives a CLUSTER number to each plot, as generated by the FASTCLUS function in Appendix B. The clusters have been renamed to more appropriate numbers where CLUSTER 1 represents CLASS 3, CLUSTER 2 represents CLASS 2, CLUSTER 3 represents CLASS 1, and CLUSTER 4 represents CLASS 4.

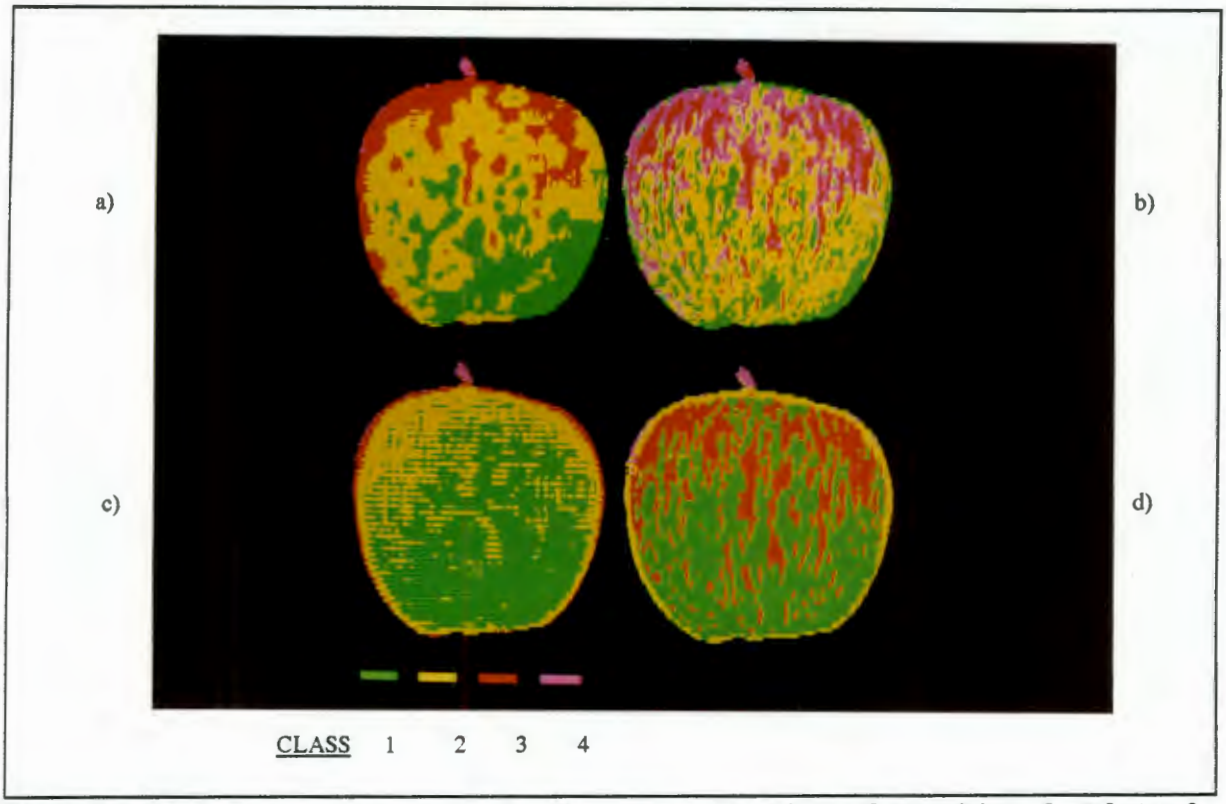


Figure 6.9: 4 class unsupervised colour segmentation of Starking 6 (Plate 3) for each of the feature combinations a) H b) H-I c) H-S1 d) H-S1-I

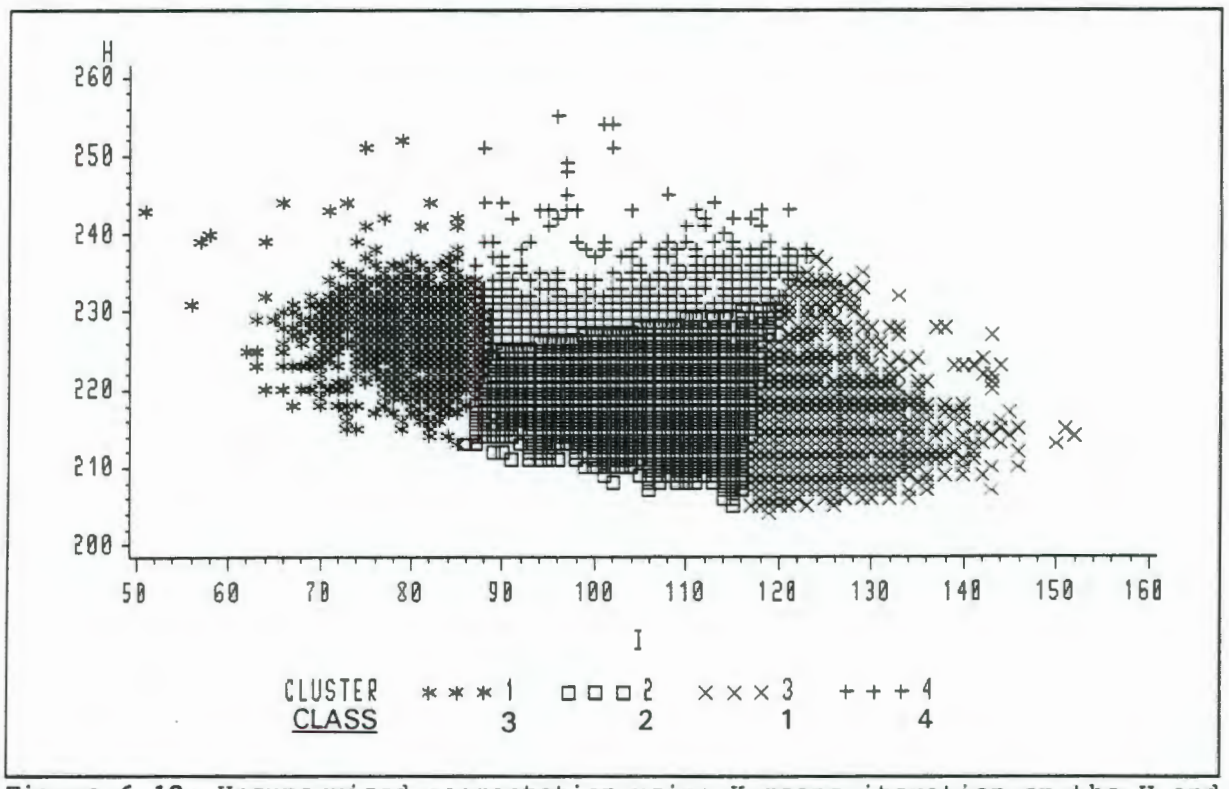


Figure 6.10: Unsupervised segmentation using K-means iteration on the H and I features of pixels in Starking 6 (Plate 3). The class centroids are given in Table 6.9b.

In these experimental results one can see the true effect of the features on a fruit that is illuminated without gloss or major shadow effects. If one considers classes 1 and 2 as representing the green/yellow parts of the Starking and classes 3 and 4 as the red/brown parts of the fruit image, then the H feature gives a fairly rough segmentation of the colours (Figure 6.9a). Introducing the I feature allows for a clearer definition of the red and green striations (Figure 6.9b). The H-S1 feature combination detects most of the red on the fruit as class 2 and the green as class 1 (Figure 6.9c). The H-S1-I feature combination reveals a boundary of class 2 due to the effect of the S1 component but most of the red striations are detected due to the effect of the I component (Figure 6.9d).

In general it appears that hue gives an adequate representation of the colour distribution on the fruit. The intensity feature, improves on the boundary definition between the different classes on the fruit. The saturation feature shows the lighting effect on the fruit, which in this case saturation is fairly uniform and high over the central region of the fruit image, but low at about a 6 pixel thick boundary around the fruit.

6.5.2. Supervised Colour Segmentation of Starking 6 (Plate 3)

The H-I plane of Figure 6.10 contains a tight elliptically shaped scatter plot of all the sampled pixels in the Starking 6 image. Finding four class centroids in this plot is manually quite difficult, since there are no observable distinct clusters within the scatter plot. The hue range is in fact quite narrow (204^0 to 255^0) with a majority of the pixels clustering around a hue of 217^0. It was found that two classes were sufficient to segment the Starking image. The first class was determined by averaging the feature components in a 25 pixel area on a green/yellow region on the Starking 6 image. The second class was determined by averaging the feature components in a 25 pixel area on a red region of the fruit image. Table 6.10 shows the coordinates of the features representing the supervised classes. Table 6.11 shows the results of these class distributions over a set of six of the Starking apple images from Plate 3, for feature combinations of H, I and H-I. Figure 6.11 shows some of the colour segmented images resulting from the three different feature combinations used to segment Starking images 1, 2, 5, 6, 10 and 11 from Plate 3. Note that only one image is shown of Starkings 1 and 11 (Figures 6.11a and 6.11l), two images are shown for each of Starkings

2 and 10, and three images are shown for each of Starkings 5 and 6. The values in Table 6.11 should be sufficient to represent the concentrations of classes in those images not shown.

Table 6.10: Supervised class centroids for 2 class colours (derived from Starking 6) to be used in classifying the Starking apples in Plate 3.

Colour class numbers	Average feature components of class colour viewed under a red/blue tint		
	H	S1	I
1	204	125	138
2	234	125	67

Table 6.11: Colour segmentation results for Starking images 1, 2, 5, 6, 10, and 11 of Plate 3, using percentage class distributions for three feature combinations (H, I, and HI)

Fruit classed from Plate 3	Feature combinations used from the classes in Table 6.10.	Percentage [%] area of each class covered on the fruit image. (NOTE: pixels from the images are sampled every 2 lines.)	
		1	2
Starking 1	H	0	100
	I	5.0	95.0
	H-I	2.3	97.7
Starking 2	H	1.4	98.6
	I	23.7	76.3
	H-I	16.4	83.6
Starking 5	H	25.6	74.4
	I	72.3	27.7
	H-I	65.8	34.2
Starking 6	H	40.2	59.8
	I	55.5	44.5
	H-I	52.4	47.6
Starking 10	H	73.2	26.8
	I	72.7	27.3
	H-I	74.0	26.0
Starking 11	H	95.9	4.1
	I	95.2	4.8
	H-I	95.7	4.3

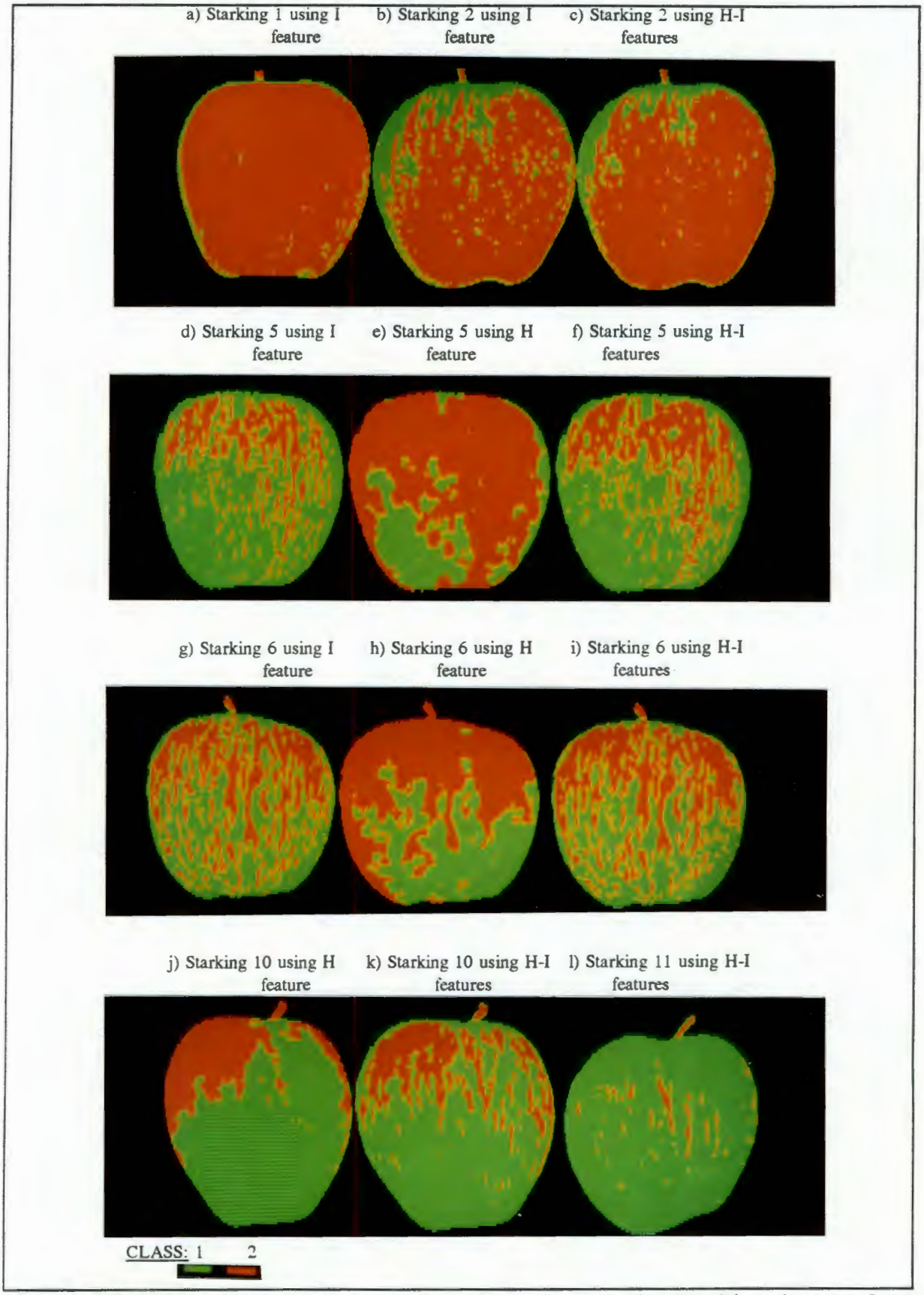


Figure 6.11: Two class supervised classification of six Starking images from Plate 3. The classification features used are H, I and H-I.

Observations for supervised classification of 6 Starking images

If one compares the images segmented by two classes (green and red) in Figure 6.11 to the actual images they represent in Plate 3, one can see that it is possible to distinguish between Starkings 1 and 2 and Starkings 10 and 11. However, the distinction between Starkings 5 and 6 is less obvious.

The general trend in the colour segmented images is that the I feature alone will always detect most of the red striations. The H feature alone generally detects more red than is actually visible on the fruit, and the striation boundaries are not clearly defined. The H-I feature combination gives the most accurate colour segmentation that best represents how the human eye would segment the images between red and green.

It should be noted that if the class centroids for the I feature were changed to a slightly lower value then more of the red striation would appear in the two class segmented images. For example, such a change could mean lowering class 1 of I from 138 to 131 and class 2 of I from 67 to 60 (see Table 6.10). Similarly, to decrease the amount of red detected using the H only feature then lowering just the centroid of class 2 from 234^0 to say 220^0 would yield a more accurate H only colour segmentation.

From the above observations one can deduce that it is possible to obtain an accurate colour segmentation of Starkings using just two classes and one feature (preferably intensity instead of hue). However, if it is known that there are more than two main colours on the fruit (other than red and green) then more classes can be used and the H-I feature combination would give the most accurate colour segmentation of the fruit.

6.5.3. Testing the 4 Unsupervised and 2 Supervised Class Centroids from Starking 6 on a Real Starking Fruit Sample

One side view of a real Starking apple (see Figure 6.12) was used to check if the 4 unsupervised and 2 supervised class centroids set up earlier, could classify this image of the real fruit. The results of the class percentage distributions over the image are given in Table 6.12. Figure 6.13 gives the unsupervised and supervised class segmented representations of the real fruit image shown in Figure 6.12.

Table 6.12: Percentage class distribution results for the real Starking fruit image in Figure 6.12, segmented with unsupervised class centroids from Table 6.9 and supervised class centroids from Table 6.10

Colour class number	a) Unsupervised class centroids using H and H-I features and percentage coverage of each class (Figure 6.13a and Figure 6.13b)					Colour class number	b) Supervised class centroids using H and H-I features and percentage coverage of each class (Figure 6.13c and Figure 6.13d)				
	H	% area	H	I	% area		H	% area	H	I	%area
1	213	17.4	216	124	9.2	1	204	23.9	204	138	22.0
2	222	33.2	220	104	21.3	2	234	76.2	234	67	78.0
3	231	47.9	227	81	50.1						
4	248	1.6	230	99	19.3						

Observations for the unsupervised and supervised classing of the real Starking image.

The results of colour segmentation of the real fruit, given in Table 6.12 and shown in Figure 6.13, confirm the final deductions mentioned in the observations of the supervised classification of six Starkings. The H feature alone gives a generally rough colour segmentation of the real Starking (Figure 6.13a and 6.13c). In both the supervised and unsupervised results (Figure 6.13b and 6.13d) it appears that the H-I feature combination yields a better colour segmentation than the H only results. One can see that the two class supervised segmentation produces an adequate result (Figure 6.13d), however, the four classes in the unsupervised segmentation (Figure 6.13b) yield a superior result, where more of the lighter red striations are detected.

One must remember that if a colour segmentation using say four classes and the H-I feature combination is used in the final grading of the fruit, one knows from observing the pixel distributions of the classes over the fruit that segmentation is sufficiently accurate. Hence, one only needs to grade according to the percentage distributions of each class.

Figure 6.12: A side view image of a real Starking apple to be colour segmented with supervised and unsupervised classes

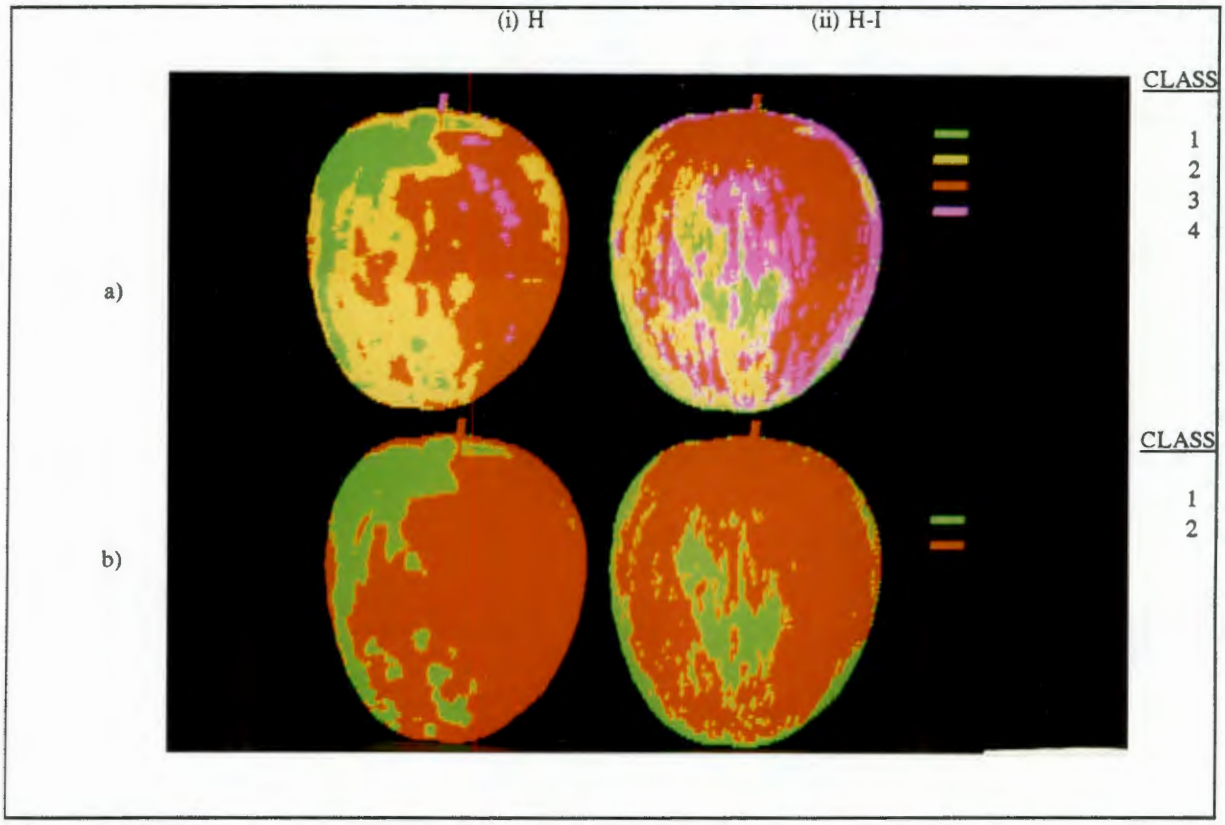


Figure 6.13: Colour segmentation of a real Starking: a) 4 class unsupervised colour segmentation for feature combinations i) H ii) H-I; b) 2 class supervised segmentation for features i) H and ii) H-I

6.6. The Grading of Fruit Images using Hue Averages and Variance

The previous sections in this chapter have all presented colour segmentation results of fruit skin colours. The fruit can be graded by determining the percentage coverage of a certain colour class over the area of the fruit observed. In this section, the results of an alternative method of grading are given. These results aim to show that fruit (Granny Smiths in this experiment) can be graded using the average hue (Have) and hue variance (Hvar) of all pixels sampled in the fruit image.

The results given in Table 6.13 and shown in Figure 6.14 are used to indicate that it is possible to numerically distinguish between Granny Smiths with russeting, pink blush and sunburn. Iterative clustering techniques, such as those described in section 5.4., are not used on these results. In this case a simple manual clustering method shows that the different types of Granny Smiths can be identified. The Golden Delicious hue averages and variances shown in Table 6.13 and Figure 6.14 are used to show that the Hvar and Have values for each relatively uniformly coloured Golden Delicious are quite distinguishable from the varying coloured Granny Smiths.

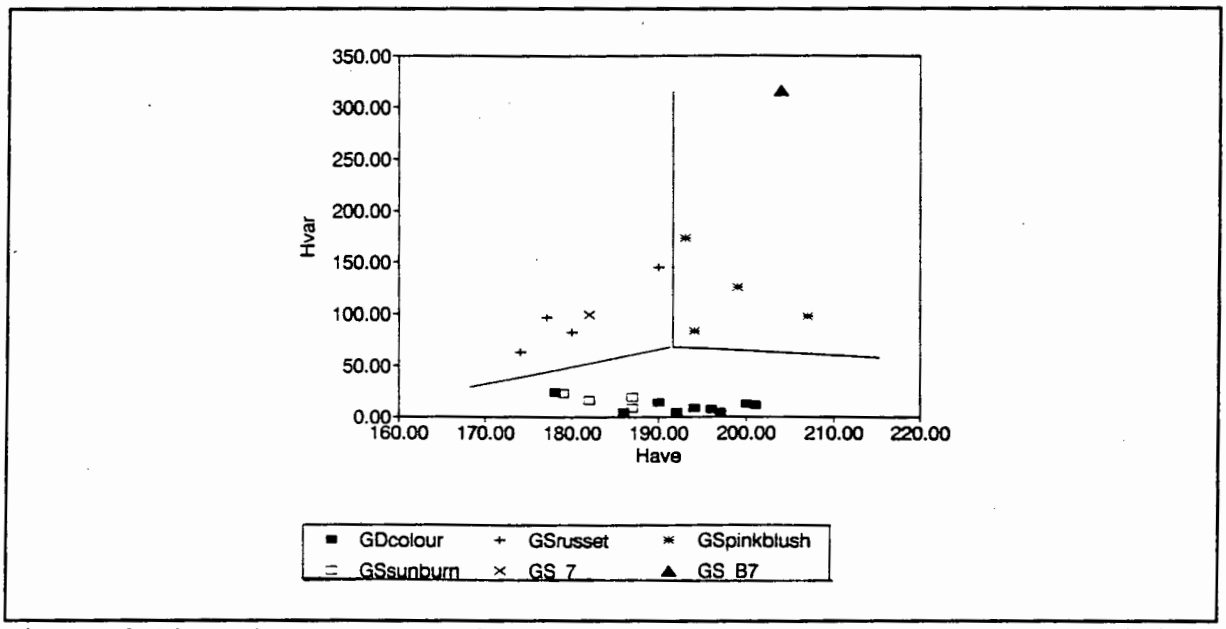


Figure 6.14: Points representing the Granny Smith (GS) and Golden Delicious (GD) fruit images by average hue and hue variance for each fruit (see Table 6.13)

Table 6.13: Hue averages (Have) and variances (Hvar) for various Granny Smith fruit image samples with russeting, pink blush, and sunburn, and the 9 Golden Delicious fruit images of Plate 1.

Fruit image	Have	Hvar
GSrusset:	177	96
GS_1	174	63
GS_12	190	145
	180	82
GS_7	182	99
GSpinkblush:	199	125
	194	83
	207	98
	193	173
GS_B7	204	316
GSsunburn:	179	23
	182	16
	187	19
	187	8

Fruit image		Have	Hvar
GDcolour:	1	178	24
	2	186	5
	3	190	14
	4	192	5
	5	194	8
	6	196	7
	7	197	4
	8	200	13
	9	201	11

Observations on Fruit Grading with Have and Hvar values

Figure 6.14 shows that it is possible to manually separate 12 different fruit based on their Have and Hvar values. The 12 fruit consist of Granny Smiths in which 4 have russeting, 4 have pink blush, and 4 have sunburn. One will notice in Table 6.13 that the hue variance is larger for the Granny Smith with large russeting (GS_12) compared to the Granny Smith with small russeting (GS_1). This verifies that fruit with larger blemishes have larger variance in hue. Figure 6.14 shows that below a hue of about 192^0 the Granny Smiths with russeting and sunburn are represented, where those apples with russeting all have hue variances larger than about 50. The apples with pink blush have generally redder hue averages all higher than 192^0 and hue variances also greater than about 50. One can see that, in comparison, the Golden Delicious without blemishes, have a hue average ranging from 178^0 to 201^0 and hue variances that are less than 25.

Two other apples have been added to Table 6.13 after the 12 Granny Smiths were manually segmented. The fruit images are GS_7 (Granny Smith 7 shown in Plate 2) and GS_B7 (Granny Smith with pink blush shown in Figure 4.11c). One can see from Figure 6.14 that these two fruit have been appropriately classed into the manually defined class boundaries for Granny Smiths with russeting and Granny Smiths with pink blush.

From the above observations one can deduce that the Have and Hvar features are adequate for general grading of the fruit. Obviously this approach may be faster than pixel by pixel colour segmentation of the fruit, but it would not be as accurate. An approach that may yield the most accurate grading of the fruit would be one that uses both the pixel by pixel colour segmentation classing and the Have and Hvar classing methods.

6.7. Guide to Run Times and Errors in the Results

All of the above experimentation was performed on a 16MHz 386 IBM compatible P.C. with no co-processor. The windowed area was on average 188x229 (43052) pixels in which image processing took place. On average the number of pixels sampled from the fruit image in the window was 15000 pixels. Each pixel would be transformed from its RGB components to HSI components using equations 4.1 and 4.8.

It takes on average 600ms to read from an array, the H, S and I components of each sampled pixel and obtain the average, variance, maximum and minimum values of each of these components.

It takes on average 1 to 2 minutes to colour segment into 10 classes using two features, an image of about 15000 sampled pixels. This segmentation process involves:

1) transforming each pixel from RGB to HSI,
2) finding the pixels class by determining its feature components minimum distance to the set class centroids with the same feature components.
3) calculating the percentage distribution of each class in the sampled image and
4) displaying the segmented image on an overlay to the actual image.

According to Celenk and Smith (1986), the L*a*b* system requires 2 to 9 minutes (using the DEC PDP-10 system) to colour segment an image. This time is slow, and should not really be compared to the times given by the 386 machine. However, an investigation into testing the colour segmentation process using DSP chips and hardware RGB to HSI converters, should reveal that run times will be greatly reduced.

The main error inherent in the experimental laboratory set up is the noise associated with each pixel before the image is grabbed. In a test to check the extent of the associated noise the following procedure and result shows the errors associated with reading each pixel feature:

> Ten images were grabbed of the same image. This image was of the Golden Delicious 1 shown in Plate 1. For each image grabbed the average H, S1 and I features were calculated for the fruit region. It was calculated that the ten images gave an average H value of 180 $\pm$ 2 , an I value of 111 $\pm$ 2 and an S1 value of 102 $\pm$ 2 overall.

This noise can be considered as white noise and should be considered when results are taken in the proposed industrial inspection system.

Other errors experienced are those due to uneven lighting on the fruit. This resulted in gloss spots and shadows around the edges. Under these lighting conditions errors in hue values can usually be detected within a 6 to 20 pixel boundary around the fruit. The saturation feature generally indicates these gloss and shadow regions as having low saturation values.

CHAPTER 7

Conclusions and Recommendations

7.1. Conclusions

From the findings, theory and experimental results described in this thesis, several conclusions can be made, that fulfil the thesis objectives set out in chapter 1.

All colour representation systems can be derived from the CIE 1931 XYZ tristimulus values. Two main colour representation systems are spectral distribution curves and colour represented as equivalent stimuli. All these colour systems have been used in various colour representation applications, and hence, are worth considering for an on-line colour inspection system.

Experimental results with a computer spectrophotometer show that colours can be represented by: spectral power curves; the CIE 1931 x,y chromaticity coordinates; the CIE 1960 UCS u,v values; L*u*v* values; L*a*b* values; RGB values; and HSI values. Each of these colour systems has shown to be viable for a colour inspection system in terms of adequately representing colours. However, the HSI system was chosen for further investigation since it related to the human perceptual understanding of colour, and transformation to this colour space was sufficiently fast.

There is no standard HSI colour representation system, each system is generally tailored from an RGB system to suite a specific application. Hence, an HSI system was derived from considering seven RGB to HSI transformations. Experimental results showed that various combinations of the H, S, and I values could be used to represent all colour pixels in a fruit image. Hence, a method was needed to find which features best classify the pixels and hence grade the fruit.

The H and I features were used to describe several clustering and classification theories. The fastest classification methods were required for experimentation, hence classifications based on minimum Euclidean distances were chosen.

A colour calibration of the lighting system is needed if results are to be consistent for comparisons. The more pure white the light is, the larger and better the colour distribution is over the fruit.

Finding class centroids by the unsupervised K-means method and supervised selection of colour sample areas, generally produced similar results. It was found that the H-I and H-S1-I feature combinations were best for colour segmenting Granny Smith apples with russeting into 4 classes. The H feature alone, using 10 classes allowed for the best classification of Golden Delicious apples. However, if the lighting conditions on the fruit images sampled did not produce glossy highlights, then the H-I and H-S1-I feature combinations would have given better classifications. The I feature alone gave adequate colour segmentation of Starking apples into two classes. This classing was generally better than using the H feature alone. However, if 4 classes were used then the H-I feature combination would allow for a better colour segmentation of the fruit image. The Have and Hvar features allow for an adequate and quick method of grading a fruit cultivar.

Several conclusions relating to the H, S, and I features can be made. The hue range for most fruit is narrow (less than 50^0). This means that the H feature alone is able to classify the fruit for general colour and not fine markings. In some cases the H feature identifies blemishes but not sufficiently well.

Saturation calculated as a vector distance (S1) is a good feature for detecting the lighting distribution on the fruit. Saturation with hue seems to be of little use in detecting actual colour and blemishes. However, the H-S1-I feature combination yields accurate colour segmentation results in some cases, but usually this feature combination is very sensitive towards the lighting distribution.

Intensity is of great importance if clearly defined boundaries are required of markings and blemishes. In general the H-I feature combination produces the best results if lighting is evenly distributed over the fruit.

The theory and algorithms given in the chapters, and the listing of the colour analysis prototype package in Appendix A, complete the objectives set out for this thesis.

7.2. Recommendations

Based on the findings given in the chapters and on the conclusions made, the following recommendations can be made.

- The HSI colour system should be used for the automated industrial colour inspection system. A hardware implementation of equations (4.8) and (4.1) should be made for RGB to HSI conversion, if it is found that the DT7910 RGB to HSI converter is too slow.

- An automated lighting calibration system must be developed based on the method described in section 6.2. Lighting conditions should be evenly distributed over the fruit, and the lighting colour should be as near to absolute white as possible.

- Two methods to grade the fruit should be implemented. The first is to have predefined colour palettes from which classes can be selected. These classes will be specific to each cultivar to be passed through the inspection system. Pixels sampled from a captured fruit image are classed according to a minimum Euclidean distance to each selected class centroid, using the H-I feature combination. The pixel distribution for each class will determine the grade of the fruit.

  The second method to implement is that from each fruit image the hue average (Have) and hue variance (Hvar), must be determined. By classing the fruit to the nearest predefined classes of Have and Hvar the grade of the fruit will be determined.

7.3. Future Outlook and Further Recommendations

This project has only dealt with the classification of one view of several fruit images. To increase the accuracy in determining the over all colour distribution on a whole fruit, several views of the fruit must be used. The proposed colour inspection system could use the colour grading methods described above for each view of the fruit. However, care must be taken not to use multiple views of the same region.

The vector components v1 and v2 that result in the polar coordinates of H and S, are other features that should be investigated in combination with the I feature, in the colour segmentation and classification methods. In addition the new CMC colour standard should be investigated, since it may become an industrial standard (Burke (1992)).

An alternative colour system that should be investigated are the chrominance signals given in the PAL system transmission of colour signals (PAL Colour Tv (1967)). In the PAL system three colour signals are needed to carry all the colour information; the luminance signal (Y) is one, the chrominance signals are the other two. The chrominance signals are obtained by subtracting the luminance signal from the colour signals for red (R) and blue (B). This thesis was concerned with the RGB output signal from a colour camera, hence, PAL signals were generally not considered. It is important to note that if the PAL signal is used there is a band effect that is caused from the phase alternating of the colour signal on alternate lines displayed. This band effect causes consecutive lines to appear alternately darker and lighter than what is normally interpreted by an observer. This band effect is averaged out when the lines are viewed by the human observer and a uniformly coloured area will appear uniform to the observer. This banding effect would have to be accounted for if the PAL chrominance signals were to be used in a colour inspection system.

If the proposed colour inspection system is to use neural network technology then according to Traven (1991) a one-layer network (without lateral interaction) can be used. He uses the network with 32 hidden units, using the class conditional densities of the chromaticities (i.e. vector components v1 and v2) as input to the network. His results showed that it was possible to identify green, red and orange fruit in an image. Further investigation into neural networks for colour classification are beyond the scope of this thesis.

References

Ball G., Hall D., (1965) **ISODATA, A Novel Method of Data Analysis and Pattern Classification**, Technical Report, April, passim., Stanford Research Inst. California.

Benson K.B., (1986) in Slaughter D.C., and Harrell R.C., (1987) 'Colour Vision in Robotic Fruit Harvesting', IEEE Trans. of the ASAE, Vol.30(4), July, p 1146.

Brainard D.H., Wandell B.A., (1990) 'Calibrated Processing of Image Colour', COLOUR research and application, Vol 15, No.5 October, pp266-271.

Burke P.,(1992) Technology Manager, Woolworths (PTY) Ltd, Cape Town. Personal Communication.

Burger P., Gillies D., (1989) **Interactive Computer Graphics: functional, procedural and device level methods**, passim., Addison-Wesley.

Carden K.J., (1987) 'An application of classification and clustering techniques using multispectral astronomical images', Tec. Doc. 87/39, December, Rhodes University Dept. of Comp. Sci.

Celenk M., Smith S.H., (1986) 'Color Image Segmentation by Clustering and Parametric-Histogramming Technique', Proc. of 8th Int. Conf. on Pattern Recognition, October, pp 883-886.

Chang Y., Liang P., Hackwood S., (1989) 'Unified study of colour sampling' Applied Optics Vol. 28, No.4.

Cheung K., Atlas L., Ritcey J., Green C., Marks R., (1987) 'Conventional and composite matched filters with error correction: a comparison', Applied Optics Vol.26, No.19.

Courjon D., Bulabois J., Mered C., (1984) 'Spatial frequency pseudocolour encoding using coarse gratings', Applied Optics Vol.23, No. 10.

Data Translation Manual, 'DT7910 and DT7911', Data Translation, Inc.

Delwiche M.J., Sites P.W., (1988) 'Computer Vision to Locate Fruit on a Tree', Transactions of the ASAE, Vol. 31(1), January, pp 257-263.

Duda R.O., Hart P.E., (1973) 'Unsupervised Learning and Clustering', **Pattern Classification and Scene Analysis**, Ch6 pp189-256, Wiley and Sons.

Einhorn E.H., (1990a) 'Light, Sight and Vision', EEE449F Illumination Notes. University of Cape Town.

Einhorn E.H., (1990b) 'Colour Science in Lighting Engineering', EEE449F Illumination Notes. University of Cape Town.

Eklundh J.O., Yamamoto H., Rosenfeld A., (1980) 'A Relaxation Method for Multispectral Pixel Classification', IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAM1-2, No. 1, January, pp 72-75.

Galloway K., (1989) 'Quality Control Decisions: Pass or Fail', ACS Customer Support Bulletin, Applied Colour Systems, inc.

Gleb A., (1982) 'Covariance Matrix', **Applied Optimal Estimation**, Sec. 3.6, pp 72-75, The M.I.T. Press.

Healey G., (1989) 'Colour Discrimination by Computer', IEEE Trans. on Systems, Man & Cybernetics, Vol19, No.6. November, pp1613-1617.

Ho J., Funt B., Drew M., (1990) 'Separating a Colour Signal into Illumination and Surface Reflectance Components', IEEE Trans. Pattern Analysis Vol. 12, No. 10. pp 966-977.

ICS Texicon Spectraflash Manual (1991), ICS Texicon Ltd.

JVC Instruction Book, 'Colour Video Camera Head TK-870E', JVC Ltd.

Keller M.J., Covavisaruch N., Unklesbay K., (1986) 'Colour Image Analysis of Food', Proc. 8th Int. Conf. Pattern Recognition, pp 619-621, October.

Klinker G., Shafer S., Kanade T.,(1987) 'Using a colour reflection model to separate highlights from object colour',Proc. First Int. Conf. Computer Vision, London, June pp145-150

Lehar A.F., Stevens R.J.,(1984) 'High-Speed Manipulation of the Colour Chromaticity of Digital Images',IEEE CG&A, February pp34-39.

Ludman J., Javidi B., Kuo C.J., Chen Y.F., (1988) 'Colour object identification by monochromatic binary correlation' Applied Optics, Vol.27, No.5.

MacAdam D.L., (1981) **Colour Measurement: Theme & Variations**, passim, Springer Series in Optical Sciences.

Matrox Image Series, 'High-performance high-resolution Image Processing and Graphics', Matrox pamphlet, Matrox Electronic Systems Ltd.

Matrox IMAGER-AT (MVP-AT) User's Manual (1989), Matrox Electronic Systems Ltd.

Meyer G.W., Greenberg D.P., (1988) 'Colour-Defective Vision and Computer Graphics Displays', IEEE CG & A, September, pp 28-40.

Miller B.K., Delwiche M.J., (1989) 'A Color Vision System for Peach Grading', Transactions of the ASAE, Vol. 32(4), July, pp 1484-1490.

Morrissey-Golas B., (1989) 'Design Colour Imaging into any Graphics System', Electronic Design, November, pp 89-98.

Niblack W., (1986) 'Image Display', **An Introduction to Digital Image Processing**, Ch2, pp23-67, Prentice/Hall.

Niblack W., (1986b) 'Classification', **An Introduction to Digital Image Processing**, Ch7, pp167-189, Prentice/Hall.

PAL Colour Tv, (1967) 'Principles of The PAL System', Part 1, pp 0.1-0.9, Mullard.

Parkkinen J., Jaaskelainen T., (1987) 'Colour representation using statistical pattern recognition' Applied Optics, Vol.26, No. 19, October, pp 4240-4245.

Ramirez F., (1986) in Niblack W., (1986) **An Introduction to Digital Image Processing**, Ch2, pp23-67, p206, Prentice/Hall.

Robertson P.K., (1988) 'Visualizing Colour Gamuts: A User Interface for the Effective Use of Perceptual Colour Spaces in Data Displays', IEEE CG & A, September, pp 50-63.

Roper G., (1992) Public Relations Officer, Protea Laboratories, Capetown. Personal communication.

Rossotti H., (1983) **Colour, why the world isn't grey**, passim., Princeton Science Library.

SAS/STAT User's Guide, (1990) 'Introduction to Clustering Procedures', Ch6, pp53-101, 'The FASTCLUS Procedure', 4th ed., Vol 1, Ch22, pp823-849.

Slaughter D.C., Harrell R.C., (1987) 'Colour Vision in Robotic Fruit Harvesting', IEEE Trans. of the ASAE, Vol. 30(4), July, pp1144-1148.

Statgraphics User's Guide (1987), passim., STSC Inc., U.S.A.

Tajima J., (1983) 'Uniform Colour Scale Applications to Computer Graphics',Comp. Vis.,Graph. & Image Proc. Vol 21, pp 305-325.

Tominaga S., (1986) 'Colour Image Segmentation Using Three Perceptual Attributes', Proc. of 8th Int. Conf. Pattern Recognition, October, pp 628-630.

Traven H.G.C., (1991) 'A Neural Network Approach to Statistical Pattern Classification by "Semiparametric" Estimation of Probability Density Functions', IEEE Trans. on Neural Networks, Vol.2, N0.3, May, p373

Uehira K., Komiya K., (1990) 'Pattern edge colour difference in a colour image sensor with a GRIN rod lens', Applied Optics, Vol. 29, No. 28, October,pp 4081-4086

van Ryzin J.,(1977) **Classification and Clustering**, passim., Academic Press Inc.

van Zyl B., (1991) Manager, Teklogic, Somerset West. Personal communication.

Wandell B.A.,(1987) 'The Synthesis and Analysis of Colour Images', IEEE Trans. Pattern Anal. & Machine Intel. Vol.PAMI-9 No. 1, January, pp2-13.

Ware C., (1988) 'Colour Sequences for Univariate Maps: Theory, Experiments, and Principles', IEEE CG & A, September, pp 41-49.

Wright W.D., (1964) **The Measurement of Colour**, passim., 3rd Ed. Hilger and Watts, London.

Wyszecki G., Stiles W., (1967) **Colour Science**, passim., Wiley and Sons.

Yamatake-Honeywell Manual, 'CS70 Series Intelligent Colour Sensor', Yamatake-Honeywell micro switch division.

APPENDIX A

The CAPP User's Guide and Program Listing

A.1. Introduction

The Colour Analysis Prototype Package (CAPP) is a software tool developed in the Electrical Engineering department at the University of Cape Town. The software was developed using an exploratory programming approach to gain an insight as to how fruit colours (or any colour image) can be analyzed and hence classified.

CAPP was written in Microsoft C Version 6.0. It uses the Microsoft C libraries and the IMAGER-AT libraries. The IMAGER-AT is the software used as an interface to the MVP-AT colour frame grabber board. The minimum hardware requirements for CAPP are an IBM compatible AT with a 286 microprocessor with a hercules graphics card, an RGB colour monitor, an RGB colour video camera, and the MVP-AT colour frame grabber board designed by Matrox Inc.

The following features make CAPP a useful tool in analyzing any colour image:

- the letter driven menu system provides a useful interface for using the IMAGER-AT software utilities for general image input and output and frame buffer handling.

- the RGB pixels of the image can be interpreted as having the associated features of hue (H), saturation (S), and intensity (I). These are the features used in the colour analysis routines.

- H, S, and I histograms, H versus S, and H versus I colour plots of the pixels in user defined areas of interest can reveal a greater insight into the distributions of the colours in the image.

- A cursor can reveal the RGB and HSI values at any position in the image, and a line profile can be drawn for any feature between any two points in the image.

- Colour segmentation is possible in three ways. Firsty, by defining threshold levels within the histogram of a certain feature. Secondly, by using user selected colour class centroids for clustering by minimum distances. Finally, it is possible to use centroids determined by unsupervised clustering to segment the image by minimum distances.

Section A.2. is a user's guide to the software, by way of an example to classify the colours on an image. Section A.3. gives a general program design and recommendations are made in order to develop this software further. A listing of the CAPP program code ends this Appendix.

A.2. CAPP User's Guide

A example will be used to illustrate the procedures needed to capture and colour segment an image. Most routines that are not used in this example are self explanatory since the main menu provides comments as to what the routine should do. In the example the set up will be assumed to be the same as in Figure 4.10. An image of a single fruit on a white background will be captured. Statistics about each feature within the fruit will be given and the image will be colour segmented by supervised minimum distance classification into 4 classes using the H and I features of the pixels representing the fruit.

Inherently this example will act as a guide to the stages needed for an automated colour sorting process. The key strokes available for each option will be given in [] and the key pressed will be in **bold**. Observations will be given in (). Typing **capp** at the DOS prompt will execute the program. If the program has initialised the MVP-AT board correctly then a series of red bars should appear across the RGB monitor screen. This indicates that all frame buffers are now full. The stages to follow in the example are now shown.

STAGE 0: Clear all frame buffers [**c 6**]

STAGE 1: Obtain a 3x512x512x8 bit colour image by
a) Image grabbing [**i <enter>**] or
b) Load a colour RGB image into a buffer [**l 6**]

STAGE 2: Get an area of interest [**g**] - arrow keys move the window, [s] toggles window steps of 1 or 20, [f] toggles fixing the top left corner of the window, **<enter>** returns to main menu with the area chosen. (the red window will now disappear).

STAGE 3: Average out the PAL band effect if a colour processor was used to convert a PAL signal from the camera to the RGB components. [a]

STAGE 4: Threshold the fruit region of interest so as to isolate the pixels within the fruit region from the background. [**t**]

STAGE 4.1: Enter an integer of the threshold level [**80 <enter>**]. Select the type of background black or white [b, **w**], and choose the number of alternate lines to sample pixels [**2 <enter>**] (the larger the number of alternate lines the smaller the sample set of pixels is). (horizontal red lies will be drawn as a mask over the area thresholded). If this threshold was not sufficient then repeat this stage [**y**, n].

STAGE 4.2: If the threshold was sufficient then there is the option to generate the H, S, and I planes and view statistical data about the features in the fruit. [**y**,n] ([n] brings up the main menu).

STAGE 4.3: Option to generate HSI planes to the screen or to the files HUE.DAT for H feature, SAT.DAT for S feature, and INTENSIT.DAT for the I feature.[1,**2**] (the red overlay will turn multi-coloured and frame buffer 2 will now contain the I plane. Primary statistical data such as the HSI means, maximums and minimums are given). The files generated data can be used in an unsupervised clustering program HICLUMEAN.SAS using the SAS software on the UCT VAX.

STAGE 4.4: Option to view more statistical data [**y**,n]. (More data showing the variances, skewness, and covariance matix of the H and I features are given. This data could eventually be used in Parametric Bayesian classification, or Have vs Hvar classification (see Chapter 5).)

STAGE 4.5: Option to write array to file HBYI.DAT of 2-D histogram for H vs I [y,**n**]. This array can be

viewed three dimensionally with the software package 3-D BOEING.(see Figure 5.1(ii))

STAGE 4.6: Option to segment the H feature by manually defining threshold levels in the feature. (for example finding the distribution between 3 defined levels in the H feature) [y,**n**].

STAGE 4.7: Option to show on the console screen a scatter plot of sampled pixels plotted with their H feature vs their I feature. (see Figure 5.1(i)) [y,**n**].

STAGE 5: Toggle the overlay mask of the threshold region on or off for viewing purposes [**m m**].

STAGE 6: With the overlay mask on press [**u**] to cluster and classify the fruit image.

STAGE 6.1: Select how centroids should be found. Centriods from a file can be previously written user supervised centroids or centroids generated by unsupervised classification (HICLUSMEAN.SAS from the VAX). Centroids can also be selected from the image [f,**i**].

STAGE 6.2: Enter number of classes to use [**4 <enter>**].

STAGE 6.3: Enter number of features to use [**2 <enter>**].

STAGE 6.4: Enter label of features 1 and 2 [**h i**]

STAGE 6.5: (a small window will appear at the centre of the monitor) Move the window to an area to average out the features and hence provide class 1 with its feature values [**<arrows> <enter>**]. This stage will be repeated for each of the 4 classes - remember that [f] toggles fixing the top left corner of the window, and [s] toggles the step size of 1 or 20.

(There are 15 different colours available for the classes, the first 4 should appear repectively in the centroid sample windows as: green, yellow, red and magenta. Once all four classes are selected the overlay containing the fruit will linearly read each pixel beneath the overlay and place its class colour in the overlay. The final overlay will show the colour segmentation of the fruit. The console screen will show the percentage distribution of each of the classes spread over the fruit image.) From these distributions the grade of the fruit can hence be deduced.

A.3. Program Design

Since CAPP is only a prototype there is room for program optimisation and further development. In its present state CAPP was developed using an exploratory programming methodology, that is, as funtions were needed they were added to appropriate modules. CAPP.MAK is a project make file which contains the module files which build the CAPP program, CAPP.EXE is the executable file for the program.

The main data structures used are 'typedef struct' structures for holding general statistics and for holding top left and bottom right window coordinates. General one dimensional and two dimensional arrays are used for holding H, S, and I features associated with each sampled pixel.

The hierarchical structure of the program is shown in Figure A.1. This figure shows the module files and the communication links between them. In the program listing that follows the purpose of each module is given at the beginning of the module, and before each function a description of that function is given.

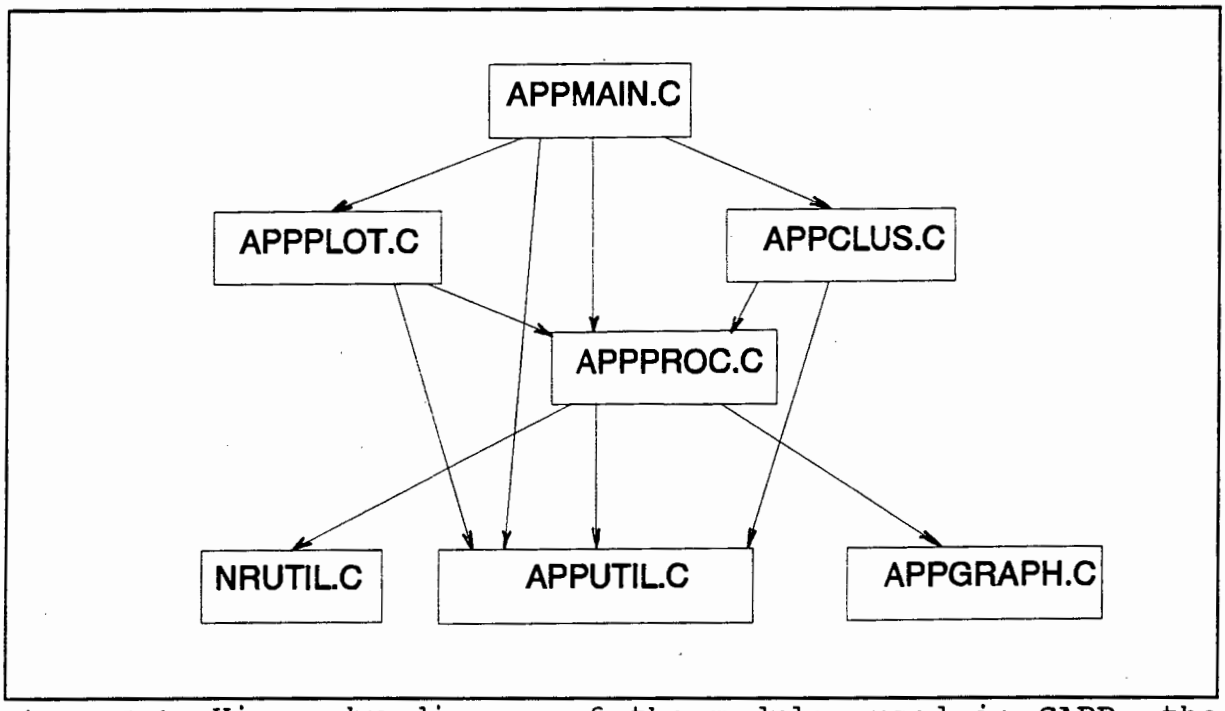


Figure A.1: Hierarchy diagram of the modules used in CAPP, the Colour Analysis Prototype Package.

A.4. Recommendations

It would not be advisable to add more functions to the CAPP program by linking in more routines. Instead, if a general Colour Analysis Package is to be made then the code should be rewritten more efficiently. The file modules should contain functions which are specifically related to that function name.

In some cases floating point operations are used with floating point values of H, S, and I. In order to reduce computation times these values should be kept strictly as short integers.

```
/*************************************************************************/
/* MODULE : APPMAIN.C                                                    */
/*                                                                       */
/* AUTHOR : G. Kay                                                       */
/*                                                                       */
/* DATE     : 5  June  1991                                              */
/* MODIFIED :    April 1992                                              */
/*                                                                       */
/* DESCRIPTION : This program provides several utilities to process      */
/*               colour images.                                          */
/* CONTAINS:   main      -  this holds the letter driven menu            */
/*             dispdata                                                  */
/*************************************************************************/
#include "appcom.h"             /* common  headers          */
#include "apptype.c"            /* common type definitions */
#include "appdefn.c"            /* common #defines          */


/****************** VARIABLE DECLARATIONS AND OPTION SETUP*****************/
int satoption = 0;      /* saturation option 0= vector dist calc, 1= min(RGB) rat
io to I */

unsigned int mem_base = 0xd000;             /* MVP-AT memory base address */
unsigned int io_base = 0x300;               /* MVP-AT I/O base address    */

/********************* FUNCTION PROTOTYPES USED IN APPMAIN *****************/

void clearbuff(void);
void dispdata(wincoords *window);
void erase(wincoords *procwindow);
void adjustolut(void);
void maskview(int *MASK);
void changeop(void);
void colograb(void);
void viewbuff(void);
void loadfile(void);
void savefile(void);
void choosexform(int framebuffer);
  void drawdisc(float mat[3][3]);
void plots(wincoords *procwindow);
  void getapple(int prflg,int choice,wincoords *procwindow);
   float huecalc(float,float);
   float satcalc(float,float,int,int,int);
   void segment(int HSI[],int size);
   void clean(int xt,int xb,int yt,int yb);
void HSItoRGB(void);
void pixoi(void);                                   /* pixel of interest*/
void RGBtoV1V2(unsigned long pix, int *R,int *G,int *B, float *V1, float *V2);
void thresh(wincoords *procwindow);
void avgPAL(wincoords *procwindow);
void zoom(int MODE);
void cut_paste(void);
  void init_aoi(wincoords *procwindow,int xt,int yt,int xb,int yb);
void clusdisplay(wincoords *procwindow);

int scancode(void);
int aoi(wincoords *window, int framebuffer);

/****************************** FUNCTION CODE ******************************/
void main(void)
```



```
{
  int ch=0;                 /* choice */
  wincoords *procwindow;    /* Pointer to struct containing procwin coords */
  int MASK=1;               /* mask toggle 1=show 0=don't show mask        */

  if (init(mem_base,io_base) != 1)     /* Check for MVP-AT */
    {
      printf("MVP-AT not found at (0x%x,0x%x)\n",mem_base,io_base);
      printf("Returning to operating system.......");
      exit(0);
    }
  /* Allocate storage for processing window coordinates */
  procwindow = (wincoords *) malloc(sizeof(wincoords));
  init_aoi(procwindow,154,17,342,246); /* Initialise proc. win. coordinates */
  adjustolut();    /* adjust the default overlay LUT for overlay display */

while(ch!='x')
{
    _clearscreen(_GCLEARSCREEN);
    sync(0,0);                          /* Use internal sync for the MVP */
    opmode(MODE2,FB0);                  /* Select processing mode        */
    disformat(1,1,0);                   /* 12.5Mhz,Interlaced,European   */
    iowin(0,0,512,512);              /* Set up European format for display */
    procwin(0,0,512,512);            /* processing window for mvpat commands*/
    (MASK == 1) ? (outpath(6,-1,2,1)) : (outpath(6,-1,0,0));
    opmode(1,0);                        /* Select graphics mode          */
    graphwin(0,0,512,512);              /* graphics window size          */
    opmode(0,6);
    video(1,1);
/*MENU*/
  printf("--COLOUR ANALYSIS PROTOTYPE PACKAGE--                                 G. Ka
y 1991\n\n");
  printf("a: Average out PAL band effect\n");
  printf("c: Clear a specified buffer\n");
  printf("d: Display statistical data of image (must plot first)\n");
  printf("e: Erase areas below, above, left or right of area of interest\n");
  printf("g: Get an area of interest\n");
  printf("h: HSI to RGB conversion and display\n");
  printf("r: R,G,B,H,S,I components of a specified pixel or line of pixels\n");
  printf("i: Image grab in colour\n");
  printf("k: (K) cut and paste to and from frame buffers\n");
  printf("o: dOs shell - type 'exit' to return to CAPP\n");
  printf("m: Mask viewing toggle\n");
  printf("n: chaNge options : saturation option = %d\n",satoption);
  printf("l: Load an image file\n");
  printf("s: Save buffer to a file\n");
  printf("t: Threshold the red or blue buffer to get fruit region in overlay\n");

  printf("u: clUster analysis segmentation display on overlay (framebuffer 2)\n")
;
  printf("v: View a specified buffer\n");
  printf("w: draW a hue-saturation disc or H-I plane (this process is O(n^3))\n")
;
  printf("z: Zoom onto an area of interest\n");
  printf("p: Plot and calculation of pixels in fruit region (H vs S; H,S or I vs
pixcount)\n");
  printf("x: eXit\n\n");
  ch = getche();
  switch(ch)
  {
```

```
    case 'a': avgPAL(procwindow); break;
    case 'c': clearbuff(); break;
    case 'd': dispdata(procwindow); break;
    case 'e': erase(procwindow); break;
    case 'g': while(aoi(procwindow,FB2) != 1); break;
    case 'h': HSItoRGB(); break;
    case 'r': pixoi(); break;
    case 'i': colograb(); break;
    case 'k': cut_paste(); break;
    case 'm': maskview(&MASK); break;
    case 'n': changeop(); break;        /* options are global vars */
    case 'o': system("command"); break;
    case 'l': loadfile(); break;
    case 's': savefile(); break;
    case 't': thresh(procwindow); break;
    case 'u': clusdisplay(procwindow); break;
    case 'v': viewbuff(); break;
    case 'w': choosexform(FB6); break;
    case 'z': zoom(MODE2); break;

    case 'p': plots(procwindow); break;
    case 'x':
    default : break;
  }
 }/* end while ch*/
 free(procwindow);
}/*end main */

void dispdata(wincoords *window)
/* AIM: to display statistical data about the apple image.
        Eventually the apple could be classified according to this data
   INPUTS: window - window of the area of interest
           STATS. - global data about the areas of window and the fruit region
                    and the peaks of the H, S, and I features.
   REQUIREMENTS: functions 'threshold' and 'plot' must be used before 'dispdata'
*/
{  long totaoi;                 /* total area of interest          */
   float appletowin;            /* ratio of inapple area to totaoi */
   float Htowin,Stowin,Itowin; /* HSI ratio to window area        */
   float Htoapp,Stoapp,Itoapp; /* HSI ratio to inapple area       */

   totaoi = (long)(window->x1 - window->x2)*(long)(window->y1 - window->y2);
   appletowin = (float)STATS.aoapple / totaoi ;
   Htowin = (float)STATS.maxHcount / totaoi;
   Htoapp = (float)STATS.maxHcount / STATS.aoapple;
   Stowin = (float)STATS.maxScount / totaoi;
   Stoapp = (float)STATS.maxScount / STATS.aoapple;
   Itowin = (float)STATS.maxIcount / totaoi;
   Itoapp = (float)STATS.maxIcount / STATS.aoapple;

   _clearscreen(_GCLEARSCREEN);
   printf("-- Image Statistics --\n\n");
   printf("Total area processed    =%ld\n",totaoi);
   printf("Total fruit region used =%ld\n",STATS.aoapple);
   printf("Fruit region            =%f%%\n\n",appletowin*100);
   printf("Hue [%d]        = %f%% of total area\n",STATS.maxH,Htowin*100);
   printf("                = %f%% of fruit area\n\n",Htoapp*100);
   printf("Saturation [%d] = %f%% of total area\n",STATS.maxS,Stowin*100);
   printf("                = %f%% of fruit area\n\n",Stoapp*100);
   printf("Intensity [%d]  = %f%% of total area\n",STATS.maxI,Itowin*100);
```



```
   printf("                = %f%% of fruit area\n\n",Itoapp*100);
   printf("\nPress a key...");
   getch();
}
```

```
/*****************************************************************************
 * MODULE: APPPLOT
 *
 * AUTHOR: G.Kay
 *
 * DESCRIPTION: colour plots of graphs and HSI discs to monitor
 *
 * CONTAINS:
 *       getapple
 *       plots
 *       drawdisc
 *       choosexform
 *****************************************************************************/
#include "appcom.h"
#include "apptype.c"
#define TRUE 1
#define FALSE 0
#define row 512
#define col 512

/**************** GLOBAL PARAMETERS FOR PLOT INITIALIZATIONS***************/
   int xorg=256, yorg=386;    /* x and y origin for H vs S plot  */
   int xorgH=24, yorgH=15;    /* x and y origin for H vs #pixels */
   int xorgS=24, yorgS=15;    /* x and y origin for S vs #pixels */
   int xorgI=24, yorgI=295;      /* x and y origin for I vs #pixels */

/********************** external functions required ***********************/

float huecalc(float m1, float m2);
float satcalc(float m1, float m2, int R, int G, int B);
void RGBtoV1V2(int usepix,unsigned long pix,int *R,int *G,int *B,float *V1,float
*V2);
void segment(int HSI[],int size);

/***************************FUNCTION CODE*********************************/

void getapple(int pflg,int choice,wincoords *procwindow)
/* AIM: To read in pixels from each of the 3 r, g, b buffers, and plot
 *       each pixel according to choice.
 *       The red, green and blue components are in FB0, FB1, FB3 resp
 *            the mask of in apple region is in FB2
 * CALLS: RGBtoV1V2,huecalc,satcalc,segment
 * INPUTS:  pflg - flag to do plots or just show data
 *          choice - users keyed input
 *          procwindow - processing window of interest
 */

{
    unsigned long pix;          /* the pixel colour */
    float V1,V2;                /* vectors to make H and S */
    int R,G,B;                  /* buffers of length col */
    float H,S;                  /* perceptual colours of the pixel */
    int k,j;                    /* loop counter */
    int *Hary,y;                        /* Hary[y]=hue array of pixel occurrences*/

    int *Sary,*Iary,x;
    int xstart,xstop,ystart,ystop;


    xstart = procwindow->x1;
```



```
    xstop  = procwindow->x2;
    ystart = procwindow->y1;            /* area of interesting region */
    ystop  = procwindow->y2;

    Hary = (int *)calloc(360,sizeof(int));  /* initialise array to 0 */
    Sary = (int *)calloc(256,sizeof(int));  /* initialise array to 0 */
    Iary = (int *)calloc(256,sizeof(int));  /* initialise array to 0 */
    _clearscreen(_GCLEARSCREEN);
    printf("Processing...\n");
    for (j=ystart ; j < ystop ; j++)
    {
      for(k=xstart; k < xstop; k++)        /* process the line */
      {
       opmode (0,2);
       if(pixr(k,j) != 0x00)              /* check if in apple */
         {
           opmode(0,6);              /* max pixr = 0xffffff */
           pix = pixr(k,j);
           RGBtoV1V2(TRUE,pix,&R,&G,&B,&V1,&V2);
/*printf("V1=%f,V2=%f,j=%d,k=%d,R=%d,G=%d,B=%d\n",V1,V2,j,k,R,G,B);*/

          switch(choice)
          {
           case '4': /* H vs S */
           {
              H = huecalc(V1,V2);
              S = satcalc(V1,V2,R,G,B);
              pixw(xorg+(int)ceil(S*cos(H*3.1415927/180)) , yorg-(int)ceil(S*sin(
H*3.1415927/180)) , pix);
              break;
           }
           case '5': /* H vs I */
           {
              y = (int)(huecalc(V1,V2));   /* horizontal hue y axis  */
              x = (R+G+B)/3;               /* vertical intensity x axis */
              pixw(xorgI + y,yorgI + x,pix);
              break;
           }
           case '1': /* H vs pixcount */
           {
              y = (int)(huecalc(V1,V2)) ;
              Hary[y] ++;
              if(pflg=='1') pixw(xorgH+Hary[y],yorgH-120+y,pix);
              break;
           }
           case '2': /* S vs pixcount */
           {
              y = (int)(satcalc(V1,V2,R,G,B)) ;
              Sary[y] ++;
              if(pflg=='1') pixw(xorgS+Sary[y],yorgS-30+y,pix);
              break;
           }
           case '3': /* I vs pixcount */
           {
              y = (R + G + B)/3;
              Iary[y] ++;
              if(pflg=='1') pixw(xorgI+Iary[y],yorgI-50+y,pix);
              break;
           }
           default : break;
```

```
      )/* end switch */
    )/*end if*/
  )/*end for k*/
)/*end for j*/
switch(choice)
{
  case '1':
  {     /* Calculate stats */
    STATS.maxH = 0;
    STATS.maxHcount = 0;
    for (y=0;y<360;y++)
    {
     if (Hary[y]!=0) printf("Hue[%d]=%d\t",y,Hary[y]);
     if (Hary[y] > STATS.maxHcount)
     {
      STATS.maxHcount=Hary[y];
      STATS.maxH=y;                 /* the hue value with the max count*/
     }
    }
    printf("\nHue[%d] has max count %d",STATS.maxH,STATS.maxHcount);
    segment(Hary,360);              /* define levels for % hue coverage */
    break;
  }
  case '2':
  {     /* Calculate stats */
    STATS.maxS = 0;
    STATS.maxScount = 0;
    for (y=0;y<256;y++)
    {
     if (Sary[y]!=0) printf("SAT[%d]=%d\t",y,Sary[y]);
     if (Sary[y] > STATS.maxScount)
     {
      STATS.maxScount=Sary[y];
      STATS.maxS=y;
     }
    }
    printf("\nSAT[%d] has max count %d",STATS.maxS,STATS.maxScount);
    segment(Sary,256);
    break;
  }
  case '3':
  {     /* Calculate stats */
    STATS.maxI = 0;
    STATS.maxIcount = 0;
    for (y=0;y<256;y++)
    {
     if (Iary[y]!=0) printf("INTEN[%d]=%d\t",y,Iary[y]);
     if (Iary[y] > STATS.maxIcount)
     {
      STATS.maxIcount=Iary[y];
      STATS.maxI=y;
     }
    }
    printf("\nINTEN[%d] has max count %d",STATS.maxI,STATS.maxIcount);
    segment(Iary,256);
    break;
  }
  default: break;
}
```



```
    free(Hary); free(Sary); free(Iary);
)/*end getapple*/

void plots(wincoords *procwindow)
/* AIM: interface to selecting a plot of H vs S or I; H,S or I vs pixcount
 *      draws the axis for the separate plots.
 * CALLS: clean,getapple
 * INPUTS: procwindow - processing window of interest
 */
{  int choice; /* plotting option */
   int pflg;   /* plot flag '1'=plot '0'=calculate*/
   int change; /* option to change position of graph on screen */

   _clearscreen(_GCLEARSCREEN);
   printf("--PLOTTING OPTIONS--\n\n");
   printf("0: Statistical calculations only\n");
   printf("1: Screen plots and statistical calcultions (slow)\n");
   pflg = getche();
   if((pflg!='0')&&(pflg!='1')) return;

   _clearscreen(_GCLEARSCREEN);
   printf("--PLOTTING OPTIONS--\n\n");
   printf("1: Hue        vs No. of pixels with that hue\n");
   printf("2: Saturation vs No. of pixels with that saturation\n");
   printf("3: Intensity  vs No. of pixels with that intensity\n");
   if(pflg=='1')
   {  printf("4: Hue vs Saturation scatter plot\n");
      printf("5: Hue vs Intensity scatter plot\n");
   }
   choice = getche();
  if(pflg=='1')
   switch (choice)
   {
      case '4': {
                  printf("\nChange origin of pixel plot (y/n):");
                  change = getche();
                  if (change == 'y')
                    { printf("\nx value of origin = ");
                      scanf("%d",&xorg);
                      printf("\ny value of origin = ");
                      scanf("%d",&yorg);
                    }
                /*  clean(xorg-256,col,yorg-155, yorg + 155);*/
                  /* plot H vs S axis */
                  opmode (1,3);             /*blue line */
                  move(xorg,yorg);
                  rline( 35,0 );
                  opmode (1,1);             /*green line */
                  move (xorg,yorg);
                  rline( -35,-60 );
                  opmode (1,0);             /*red line */
                  move (xorg,yorg);
                  rline( -35,60 );

                  move (xorg + 50,yorg);
                  gtext(1,24,"HUE=angle from blue axis");
                  move (xorg + 50,yorg + 14);
                  gtext(1,21,"SAT= dist from origin");

                  break;
```

```
            }
    case '5': { /* H vs I scatter plot */
                printf("\nChange origin of pixel plot (y/n):");
                change = getche();
                if (change == 'y')
                  { printf("\nx value of origin = ");
                    scanf("%d",&xorgI);
                    printf("\ny value of origin = ");
                    scanf("%d",&yorgI);
                  }
              /*        clean(xorgI-24,col,yorgI-15, yorgI + 155);*/
                opmode(1,1);
                move(xorgI-16,yorgI+5);
                gtext(1,2,"0");
                move(xorgI,yorgI-2);
                gtext(1,1,"0");
                move(xorgI,yorgI);
                rline(359,0);
                rline(0,-3);
                rmove(-16,0);
                gtext(1,10,"359    Hue");
                move (xorgI,yorgI);
                rmove(180,0);
                rline(0,-3);
                rmove(-16,0);
                gtext(1,3,"180");
                move (xorgI,yorgI);
                rline(0,180);
                rline(-3,0);
                rmove(-20,-1);
                gtext(1,3,"180");
                rmove(-16,-60);
                gtext(1,1,"I");
                move(xorgI,yorgI);
                break;
              }
    case '1': { /* H vs pixels axis */
                printf("\nChange origin of pixel plot (y/n):");
                change = getche();
                if (change == 'y')
                  { printf("\nx value of origin = ");
                    scanf("%d",&xorgH);
                    printf("\ny value of origin = ");
                    scanf("%d",&yorgH);
                  }
          /*    clean(xorgH-24,col,yorgH-15, yorgH + 150); */
                opmode(1,1);
                move(xorgH-22,yorgH+5);
                gtext(1,3,"120");
                move(xorgH,yorgH-2);
                gtext(1,1,"0");
                move(xorgH,yorgH);
                rline(450,0);
                rline(0,-3);
                rmove(-16,0);
                gtext(1,3,"450");
                rmove(-250,0);
                gtext(1,7,"#Pixels");
                move (xorgH,yorgH);
                rline(0,120);
```



```
                rline(-3,0);
                rmove(-20,-1);
                gtext(1,3,"240");
                rmove(-16,-50);
                gtext(1,1,"H");
                move(xorgH,yorgH);
                break;
              }

    case '2': { /* S vs pixels axis */
                printf("\nChange origin of pixel plot (y/n):");
                change = getche();
                if (change == 'y')
                  { printf("\nx value of origin = ");
                    scanf("%d",&xorgS);
                    printf("\ny value of origin = ");
                    scanf("%d",&yorgS);
                  }

               /* clean(0,col,0,163);*/
                opmode(1,1);
                move(xorgS-22,yorgS+5);
                gtext(1,2,"30");
                move(xorgS,yorgS-2);
                gtext(1,1,"0");
                move(xorgS,yorgS);
                rline(450,0);
                rline(0,-3);
                rmove(-16,0);
                gtext(1,3,"450");
                rmove(-250,0);
                gtext(1,7,"#Pixels");
                move (xorgS,yorgS);
                rline(0,150);
                rline(-3,0);
                rmove(-20,-1);
                gtext(1,3,"180");
                rmove(-16,-50);
                gtext(1,1,"S");
                move(xorgS,yorgS);
                break;
              }
    case '3': { /* I vs pixels axis */
                printf("\nChange origin of pixel plot (y/n):");
                change = getche();
                if (change == 'y')
                  { printf("\nx value of origin = ");
                    scanf("%d",&xorgI);
                    printf("\ny value of origin = ");
                    scanf("%d",&yorgI);
                  }

               /* clean(0,col,260,row); */
                opmode(1,1);
                move(xorgI-22,yorgI+5);
                gtext(1,2,"50");
                move(xorgI,yorgI-2);
                gtext(1,1,"0");
                move(xorgI,yorgI);
                rline(450,0);
```

```
                rline(0,-3);
                rmove(-16,0);
                gtext(1,3,"450");
                rmove(-250,0);
                gtext(1,7,"#Pixels");
                move (xorgI,yorgI);
                rline(0,180);
                rline(-3,0);
                rmove(-20,-1);
                gtext(1,3,"230");
                rmove(-16,-60);
                gtext(1,1,"I");
                move(xorgI,yorgI);
                break;
              }

    default : return;
  };
  getapple(pflg,choice,procwindow);
}

void drawdisc(float mat[3][3])
/* AIM : Draws to the monitor a disc of hue vs saturation at origin 256,256
 * CALLS:       huecalc()
 * INPUTS:      mat[3][3] - the RGB to HSI transformation matrix
*/
{
     unsigned long r,g,b;/* intensity level counters */
     int v1,v2;          /* x y coords on the screen */
     float V1,V2;        /* vectors making up H and S */
     float H,S,I;
     unsigned long pix;        /* the pixel colour */

 for (r=0;r<256;r++  )
 {
  for (g=0;g<256;g++  )
       for (b=0;b<256;b++  )
       {
            V1 = (mat[1][0]*b + mat[1][1]*g + mat[1][2]*r);
            V2 = (mat[2][0]*b + mat[2][1]*g + mat[2][2]*r);
            H = huecalc(V1,V2);
            I = (r+g+b)/3;
            if (I==0) S=0;
            else      S = 255 * (1 - min(min(r,g),min(r,b))/I);
            v1 = 256 + (int)(S*cos(H*3.1415927/180));
            v2 = 256 - (int)(S*sin(H*3.1415927/180));
            pix = b*256*256 + g*256 + r;
            pixw(v1,v2,pix);
       };
 }

/*
for (b=0;b<256;b+=32)
{
 for (g=0;g<256;g+=32)
       for (r=0;r<256;r+=32)
       {
            v1 = 256 + (int)ceil(mat[1][0]*b + mat[1][1]*g + mat[1][2]*r);
            v2 = 256 - (int)ceil(mat[2][0]*b + mat[2][1]*g + mat[2][2]*r);
```



```
            pix = b*256*256 + g*256 + r;
            pixw(v1,v2,pix);
       };
  }
*/

}/*end drawdisc*/

void drawHIplane(float XI[3][3])
/*  AIM: To draw to the RGB monitor the Hue vs Intensity plane
 *  CALLS: none
 *  INPUTS: the HSI to RGB matrix XI[3][3]
 */
{  float pi=3:14159265359;
   int H,S,I;
   int R,G,B;
   unsigned long pix;
   int xstart=127,ystart=60;    /* coords on monitor to draw the H-I plane */
   float V1,V2;


   printf("\nEnter Saturation level: ");
   scanf("%d",&S);
   for(H=359; H>=0; H--)
      for(I=0; I<256 ; I++)
      {
         V1= S * cos(H*((float)pi/180));
         V2= S * sin(H*((float)pi/180));

         B = (int)(XI[0][0]*I + XI[0][1]*V1 +XI[0][2]*V2);
         G = (int)(XI[1][0]*I + XI[1][1]*V1 +XI[1][2]*V2);
         R = (int)(XI[2][0]*I + XI[2][1]*V1 +XI[2][2]*V2);

        if ((R<=255)&&(G<=255)&&(B<=255)&&(R>=0)&&(G>=0)&&(B>=0))
        {
          pix =(unsigned long)R + ((unsigned long)G<<8) + ((unsigned long)B<<16);

          opmode(0,6);
          pixw(xstart+I,ystart+359-H,pix);
        }
      };
}

void choosexform(int framebuffer)
/*  AIM: To select from six, an RGB to HSI transformation matrix
 *  CALLS: drawdisc().
 *  INPUT: framebuffer - to draw disc to.
 */
{
      /* Niblack 1986*/
      float M[3][3] = {  .33333333,  .33333333, .33333333,
                        -.40824829, -.40824829, .81649658,
                         .40824829, -.81649658,    .0      };
      /*Lehar & Stevens 1984*/
      float X[3][3] = {  .33333333,  .33333333,  .33333333,
                         .81649658, -.40824829, -.40824829,
                         .0       ,  .70710678, -.70710678  };
      /*Ramirez in Niblack 1986 (tilted version of Niblack 1986*/
      float P[3][3] = {  .3       ,  .59       ,  .11       ,
                        -.105465  , -.207424   ,  .312889   ,
```

```
                        .445942  , -.445942  ,    .0       };
      /*Benson in Slaughter & Harrell 1987*/
      float H[3][3] = {  .299     ,  .587     ,  .114     ,
                         .596     , -.275     , -.321     ,
                         .212     , -.523     ,  .311     };

      /*DT7910 RGB to HSI conversion chip*/
/*      float F[3][3] = {  .333333  ,  .333333  ,  .333333 ,
                        1.547005  , -.57735   , -.57735  ,
                         .0       ,  1.0      , -1.0      };
*/      float F[3][3] = {  .333333  ,  .333333  ,  .333333 ,
                          1.0       , -.5       , -.5      ,
                           .0       ,  .866025  , -.866025 };

      /*inverse of DT7910 chip simulation */
      float FI[3][3] = {  1.0,  .66667, .0     ,
                          1.0, -.33333, .57735,
                          1.0, -.33333, -.57735 };

      char *litstr;
      int lit;              /* the intensity level */
      int ch;               /* choice */


  outpath(framebuffer,-1,0,0);
  opmode(0,framebuffer);                              /* Select I/O mode */

/*  printf("Enter intensity level for colour disc (0 to 255): ");
  gets(litstr);
  lit = atoi(litstr);
*/
   _clearscreen(_GCLEARSCREEN);
  printf("-- RGB to HSI matrix transformations for H-S plane --\n");
  printf("\n   [1] Niblack 1986");
  printf("\n   [2] Niblack 'tilted' 1986 ");
  printf("\n   [3] Lehar & Stevens 1984 ");
  printf("\n   [4] Slaughter & Harrell 1987 ");
  printf("\n   [5] DT7910 RGB to HSI converter chip");
  printf("\n\n-- H-I plane simulation with RGB --");
  printf("\n   [6] DT7910 RGB to HSI converter chip");
  printf("\n\n     Enter transform number:");
  ch = getche();
  switch (ch)
  {
     case '1': drawdisc(M); break;
     case '2': drawdisc(P); break;
     case '3': drawdisc(X); break;
     case '4': drawdisc(H); break;
     case '5': drawdisc(F); break;
     case '6': drawHIplane(FI); break;
     default : break;
  }
}/*end choosexform*/
```

```
/*****************************************************************************
 * MODULE: APPCLUS
 *
 * AUTHOR: G.Kay
 *
 * DESCRIPTION: do supervised clustering and colour segmentation, or take
 *              centroids generated from unsupervised clustering (from SAS
 *              on the VAX), and perform colour segmentation of fruit region
 * CONTAINS:    getclassfi
 *              selectcalsses
 *              getclusmean
 *              classify
 *              clusdisplay
 *****************************************************************************/

#include <stdio.h>
#include <limits.h>
#include <stdlib.h>
#include <string.h>
#include <graph.h>
#include "im2to3.h"                    /* Prefix MVP-AT functions with 'im' */
#include "debugat.h"                           /* MVP-AT debug information */
#include "apptype.c"
#include "nrutil.h"

#define SQ(a) ((a)*(a))
#define TRUE 1
#define MAXFEAT 3       /* maximum number of features i.e. H,S,I */

/********************** external functions required ************************/

float huecalc(float m1, float m2);
float satcalc(float m1, float m2, int R, int G, int B);
void RGBtoV1V2(int usepix,unsigned long pix,int *R,int *G,int *B,float *V1,float
*V2);

/***************************** FUNCTION CODE *******************************/

void getclassfi(int c, int *nofclasses, int **centroid, int *feat, int *nofeat)
/*  AIM: To get a region of interest and average the H,S,I features in the
 *       region.
 *  INPUTS: c        - the class to which the mean H,S,I must be found
 *          nofclasses- the number of classes to be selected by the user
 *          centroid  - array where selected features are stored for each class
 *          feat      - array of numbers indicating user selected features
 *          nofeat    - number of features to use
 *  OUTPUT: centroid   - updated array for class c
 *  CALLS:  init_aoi, aoi, (in module APPUTIL)
 *          RGBtoV1V2, huecalc, satcalc (in module APPROC)
 */

{  wincoords *classwin;     /* class window coordinates */
   int xt,yt,xb,yb;         /* top left,& bottom right coords of class window */
   unsigned long pix;
   int x,y;                 /* index to pixels in class window */
   long aoclasswin=0;       /* area of class window */
   int R,G,B,H,S,I;
   float V1,V2;
   long Hcave=0, Scave=0, Icave=0; /* H,S,I class averages for classwin*/
   int i;                   /* index to selected features in centroid */
```



```
   int cave[3];             /* H,S,I averages of centroid */

  /* Allocate storage for class window coordinates */
  classwin = (wincoords *) malloc(sizeof(wincoords));
  init_aoi(classwin,250,250,255,255);    /* Initialise class win. coordinates */
  while(aoi(classwin,2) != 1);   /* gets xt,yt,xb,yb  on 'enter'*/

  xt = classwin->x1;
  yt = classwin->y1;
  xb = classwin->x2;
  yb = classwin->y2;

  for (y=yt ; y < yb ; y++)
  {
    for(x=xt ; x < xb ; x++)      /* process the line */
    {
     aoclasswin++;
     opmode(0,6);                  /* max pixr = 0xffffff */
     pix = pixr(x,y);
     RGBtoV1V2(TRUE,pix,&R,&G,&B,&V1,&V2);
     H = (int)huecalc(V1,V2);
     S = (int)satcalc(V1,V2,R,G,B);
     I = (R+G+B)/3;

     Hcave += (long)H;
     Scave += (long)S;
     Icave += (long)I;

     opmode(0,2);
     pixw(x,y,c);

    }
  }
  cave[0] = (int)(Hcave/aoclasswin);
  cave[1] = (int)(Scave/aoclasswin);
  cave[2] = (int)(Icave/aoclasswin);

  for (i=1; i<=(*nofeat); i++)      /* adjusting so that centroids first array*/
                                    /* elements are those selected by user */
     centroid[c][i]= cave[feat[i]];

  printf("\n\n Area of class window = %ld",aoclasswin);
  printf("\nHue        class mean = %d",cave[0]);
  printf("\nSaturation class mean = %d",cave[1]);
  printf("\nIntensity  class mean = %d",cave[2]);

  free(classwin);
}

int **selectclasses(int *nofclasses, int *nofeat, int *feat)
/*  AIM: To allow the user to select a number of regions whose averages
 *       make up cluster centroids.
 *  INPUTS:  none
 *  OUTPUTS: nofclasses - number of user selected classes
 *           nofeat     - number of user selected features
 *           feat       - array of numbers indicating selected features to use
 *  RETURNS: centroid   - the centroid array of selected features
 *  CALLS:   getclassfi
 */
{   int c,i;             /* index to classes,features */
```

```
    int **centroid;     /* class centroid from image */

  printf("\nEnter number of classes to select : ");
  scanf("%d",nofclasses);
  printf("\nEnter number of features to use (eg 3, for H,S,I) : ");
  scanf("%d",nofeat);
/*  feat = (int *)malloc(sizeof(int)*(*nofeat+1));*/
  for (i=1; i<=(*nofeat); i++)
      {
        printf("\nEnter label for feature %d (h/s/i): ",i);
        switch(getche())
        {
          case 'h': feat[i]=0; break;
          case 's': feat[i]=1; break;
          case 'i': feat[i]=2; break;
          default : break;              /*!!! will have to account for errors*/
        }
      }

  centroid = imatrix(1,*nofclasses,1,*nofeat);

  for(c=1 ; c<=*nofclasses; c++)
  {
    printf("\nSelect a region to average for a class centroid %d :",c);
    getclassfi(c,nofclasses,centroid,feat,nofeat);          /* get class mean fro
m image */

  }
  return (centroid);
}

int **getclusmean(int *nofclasses, int *nofeat, int *feat)
/*  AIM: to get cluster means (or class centroids) from a data file
 *       generated by a cluster routine on SAS (VAX software)
 *  INPUTS:  data from file
 *  OUTPUTS: nofclasses - the number of classes read.
 *           nofeat     - the number of features read.
 *           feat       - array of numbers indicating the features to use
 *  RETURNS: centroid   - the array of classes by features for class centroids
 *  CALLS:   none
*/
{
  FILE *fpclus;         /* pointer to file containing centroids */
  char filename[50]="clusmean.dat";
  char line[80];        /* line holding HSI feature values */
  char string[20];      /* string of integer value */
  int **centroid;       /* array of mean features for each class */
  int feature;          /* index to numbers of features */
  int class;            /* line no. of input data */
  int i,j;              /* index to line( or feature) & string resp. */

  printf("\n\nEnter filename of cluster means : ");
  fflush(stdin);
  gets(filename);
  if( (fpclus = fopen(filename,"rt"))== NULL )
  {   printf("\n ERROR: unable to open file");
      return (NULL);
  }
  *nofclasses=0;
  *nofeat = 0;
```



```
  while( fgets(line,80,fpclus) != NULL )
       (*nofclasses)++;
  fseek(fpclus,0,SEEK_SET);   /* return to the start of the file */

  fgets(line,80,fpclus);      /* determine from the first line the nofeat */
  i=0;
  while ( line[i] != '\n' )
  { i++;
    if( ((line[i]==' ')||(line[i]=='\n'))&&(line[i-1]!=' ') )
      (*nofeat)++;
  }
  fseek(fpclus,0,SEEK_SET);   /* return to the start of the file */
  printf("\n %d classes read ",*nofclasses);
  printf("\n %d features read",*nofeat);
/*  feat=(int *)malloc(sizeof(int)*(*nofeat+1)); */
  for (i=1; i<=(*nofeat); i++)
      {
        printf("\nEnter label for feature %d (h/s/i): ",i);
        switch(getche())
        {
          case 'h': feat[i]=0; break;
          case 's': feat[i]=1; break;
          case 'i': feat[i]=2; break;
          default : break;              /*!!! will have to account for errors*/
        }
      }

  centroid = imatrix(1,*nofclasses,1,*nofeat);
  class=1;
  while(  fgets(line,80,fpclus) != NULL )  /* NULL indicates end of file */
  { feature=0;
    i=0;
    j=0;
    while( line[i]!='\n')
    { if( line[i] != ' ')
      {  string[j] = line[i];
         j++;
      }
      else
      {  if( (i!=0) && (line[i-1] != ' ') )
         {  string[j]=0;
            feature++;
            centroid[class][feature] = atoi(string);
         }
         j=0;
      }/*end else*/
      i++;
      if ( (line[i]=='\n')&&(line[i-1] != ' ') )
      { string[j]=0;
        feature++;
        centroid[class][feature] = atoi(string);
      }
    }
    class++;
  }
  fclose(fpclus);
  return( centroid );
}

void classify(wincoords *procwindow,int **centroid, int nofclasses, int nofeat, i
```

```
nt *feat)
/*  AIM: Takes as input the class centroids array. For each pixel in the
 *       overlay mask, the nearest centroid is the class of that pixel.
 *       Classes are written as a level of grey in the overlay buffer.
 *  INPUTS: centroid    - array of classes feature means
 *          nofclasses  - number of classes
 *          nofeat      - number of features detected
 *          feat        - array of numbers indicating user selected features
 *          procwindow  - coordinates of window of region to classify
 *  CALLS: getclusmean, selectclasses, classify.
 */
{
    int i;                  /* index to features */
    long *dist ;  /* array of distances from class centroids to current pix*/
    long mindist;           /* the nearest class by distance */
    long *aoclass;          /* array of area for each class */
    long aomask=0;          /* area of the overlay mask */
    int classified;         /* the class to which the pixel is classified */
    int c;                  /* index to classes */

    int x1,y1,x2,y2;                /* window top and bottom coords */
    int x,y;                        /* index to pixel in aoi */
    unsigned long pix;              /* pixel read */
    float V1,V2;                    /* H,S horizontal and vertical vectors*/
    int R,G,B;                      /* colour components of pix */
    int pixvalue[MAXFEAT]; /* H,S,I pixel features which will be given  */

    dist = (long *)malloc(sizeof(long)*(nofclasses+1));
    aoclass = (long *)calloc(nofclasses+1,sizeof(long));

    x1 = procwindow->x1;
    x2 = procwindow->x2;
    y1 = procwindow->y1;                /* area of interesting region */
    y2 = procwindow->y2;

    printf("\n Processing into %d classes",nofclasses);

    for (y=y1 ; y < y2 ; y++)
    {
      for(x=x1 ; x < x2 ; x++)    /* process the line */
      {
       opmode (0,2);
       if(pixr(x,y) != 0x00)             /* check if in apple */
         {
           aomask++;
           opmode(0,6);             /* max pixr = 0xffffff */
           pix = pixr(x,y);
           RGBtoV1V2(TRUE,pix,&R,&G,&B,&V1,&V2);
           pixvalue[0] = (int)huecalc(V1,V2);
           pixvalue[1] = (int)satcalc(V1,V2,R,G,B);
           pixvalue[2] = (R+G+B)/3;
           mindist = LONG_MAX;

    /* perform unsquare rooted mininum Euclidean distance on each centroid to pix
el read*/

           for (c = 1; c<=nofclasses; c++)  /* find mindist from pixel to each ce
ntroid*/
           {
              dist[c]=0;                    /* initialise distance to 0 */
```



```
              for (i = 1; i <= nofeat; i++)
                {
                  dist[c] += SQ((long)(centroid[c][i] - pixvalue[feat[i]]));/* ac
count for no. of features*/
                /*  printf("feat[%d]=%d pixval[feat[i]]=%d dist[%d]=%ld\n",i,feat[i
], pixvalue[feat[i]],c,dist[c]);
                */}
                if(dist[c] < mindist)
                {  mindist = dist[c];
                   classified = c;
                }
           }
           aoclass[classified]++;
           opmode(0,2);
           pixw(x,y,classified);
         }
      }
    }
    printf("\n\n Area of mask = %ld",aomask);
    printf("\n\n CLASS NO.\t AREA\t PERCENTAGE AREA");
    for (c = 1; c<=nofclasses; c++)
      printf("\n    %d   \t %ld\t %f%%",c,aoclass[c],(100*(float)aoclass[c]/aomas
k));
    printf("\n\n Press a key to continue...");
    getch();
    free(dist); free(aoclass);
}


void clusdisplay(wincoords *procwindow)
/* AIM: To use SAS generated cluster means or user defined cluster means
 *      to display an overlay showing the segmented (classed) data.
 * INPUTS: procwindow - the processing widow of interest
 * CALLS: getclusmean, selectclasses, classify
 */
{  int **centroid; /* array of class centroid means */
   int feat[MAXFEAT+1]   ; /* array containing numbers refering to features to us
e*/
   int c,f;        /* index to class and feature elements of centroid */
   int nofclasses=0; /* number of classes */
   int nofeat=0;   /* number of features */
   int choice;     /* get centroids from File or Image */

  _clearscreen(_GCLEARSCREEN);
  printf("--Cluster Analysis--");
  printf("\n\n Read class centroids from File or select from Image (f/i)? ");
  choice = getche();
  switch(choice)
  {
    case 'f':/* keep reading until correct*/
             while ( (centroid = getclusmean(&nofclasses,&nofeat,feat)) == NULL);

             break;
    case 'i':
             centroid = selectclasses(&nofclasses,&nofeat,feat);
             break;
    default : return;
  }
  for (c=1;c<=nofclasses;c++)
    for (f=1;f<=nofeat;f++)
```

```
      { printf("\ncentroid[%d][%d]=%d",c,f,centroid[c][f]);
   /*   printf(" feat[%d]=%d",f,feat[f]);    */
      }
  classify(procwindow,centroid,nofclasses,nofeat,feat);

  free_imatrix(centroid,1,nofclasses,1,nofeat);
/*  free(feat); */
  return;
}
```

```
/*****************************************************************************
 * MODULE: APPPROC.C
 *
 * AUTHOR: G.Kay
 *
 * DESCRIPTION: the processing module using general image processing routines
 *              and colour analysis transformations.
 *
 * CONTAINS:
 *      segment
 *      HSItoRGB
 *      huecalc
 *      satcalc
 *      RGBtoV1V2
 *      calcHSIplanes
 *      thresh
 *      avgPAL
 *      linehist
 *      pixoi
 *****************************************************************************/

#include "appcom.h"
#include "apptype.c"
#include "appdefn.c"
#include "nrutil.h"              /* numerical recipe for matrix allocation
                                    and stats */

/************************GLOBAL PARAMETER OPTIONS*****************************/
extern int satoption ;   /* 0 = sat as vector dist, 1= as rgb ratio to intensity*
/

/*****************************FUNCTION CODE***********************************/
void segment(int HSI[],int size)
/* AIM: to segment the passed array HSI (ie H,S or I) of size size, into
 *      various levels.
 *      The pixel count between levels is then given as the % area covered
 *      by this segment.
 * INPUTS: HSI - array containing the histogram of H, S or I of fruit region
 *         size- max range of feature i.e. 360 for H, 255 for S or I
 * CALLS:  uses the global STATS structure.
 */
{
   int y=0;          /* index to HSI array */
   int miny,maxy;    /* the lower and upper bounds for the HSI region */
   int yesno;        /* response */
   int levnum;       /* number of levels */
   int i;            /* index to level and segsum */
   int *level;       /* array of level values */
   long *segsum;     /* array of sum of pixels in each segment */

   opmode(1,1);
   drawmode(2);      /* used to put a dot on the y axis where levels are */
   /*getting maxy and miny */
   while (HSI[y] == 0)
     y++;
   miny = y;
   for( y=miny; y<size ;y++)
   {  if( (HSI[y] != 0)&&(HSI[y+1] == 0) )
        maxy = y;
   }
```



```
   printf("\n\nSegment between [%d] and [%d] (y/n) ? ",miny,maxy);
   yesno = getche();
   while(yesno == 'y')
   {
    /* getting the levels */
    _clearscreen(_GCLEARSCREEN);
    printf("-- Level Segmentation --\n\n");
    printf("Enter number of levels between [%d] and [%d] : ",miny,maxy);
    scanf("%d",&levnum);

    level = (int *)calloc((levnum+2),sizeof(int));
    segsum= (long *)calloc((levnum+1),sizeof(long));
    level[0] = miny;
    level[levnum + 1] = maxy;
    for (i=1; i < (levnum+1); i++)
    {
      printf("\nLevel[%d] = ",i);
      scanf("%d",&level[i]);          /* level[i+1] must always be > level[i]*/
    }

    /* mark on Hue axis */
/*   if(size==360)
    {  for(i=0; i< (levnum+2); i++)
       { move(xorgH-2,yorgH-120+level[i]); dot(); }
    }
*/
    for (i=0; i < (levnum+1); i++)
    {
      for(y= level[i]; y < level[i+1] ;y++)     /* noninclusive summing */
        segsum[i] += HSI[y];
      if(i==levnum) segsum[i] += HSI[maxy];     /* include upper bound */
      printf("\n[%d] to [%d] = %ld pixels",level[i],level[i+1],segsum[i]);
      printf(" (%f%% of fruit area)",((float)segsum[i]/STATS.aoapple)*100);
    }
    printf("\n\nSegment between [%d] and [%d] (y/n) ? ",miny,maxy);
    yesno = getche();
    /* mark dots off Hue axis */
/*   if(size==360)
    {  for(i=0; i< (levnum+2); i++)
       {  move(xorgH-2,yorgH+level[i]); dot(); }
    }
*/
    free(level); free(segsum);
   }/* end while*/
}/*end segment*/


void HSItoRGB(void)
/*  AIM: To convert from HSI to RGB using the inverse matrix transform
 *  INPUTS: none
 *  CALLS: none
 */
{
   /*inverse of Lehar & Stevens*/
/*      float XI[3][3] = {  1.0,        .81649658,   .0      ,
                            1.0, -.40824829,  .70710678,
                            1.0, -.40824829, -.70710678  };
*/
   /*inverse of DT7910 chip simulation */
```

```
      float XI[3][3] = {  1.0,  .66667, .0    ,
                          1.0, -.33333, .57735,
                          1.0, -.33333, -.57735 };
      float pi=3.14159265359;
      int ch=0;              /* continue choice*/
      int H,S,I;
      int R,G,B;
      float V1,V2;      /* normalised itermediate vectors making up H and S */
      unsigned long pix;/* the pixel colour */
      int x,y;          /* counters to plot the RGB representation of HSI*/

  while( ch != 27)            /* 27 is the <esc> character */
  {
   _clearscreen(_GCLEARSCREEN);
   printf(" Enter Hue integer (0-360): ");
   scanf("%d",&H);
   printf(" Enter Saturation integer(0-256): ");
   scanf("%d",&S);
   printf(" Enter Intensity integer(0-256): ");
   scanf("%d",&I);

   V1= S * cos(H*((float)pi/180));
   V2= S * sin(H*((float)pi/180));

   B = (int)(XI[0][0]*I + XI[0][1]*V1 +XI[0][2]*V2);
   G = (int)(XI[1][0]*I + XI[1][1]*V1 +XI[1][2]*V2);
   R = (int)(XI[2][0]*I + XI[2][1]*V1 +XI[2][2]*V2);

   printf("\nR = %d\tG = %d\tB = %d\n",R,G,B);
   printf("V1 = %f\tV2 = %f\n",V1,V2);
   if ((R>255)||(G>255)||(B>255))
     printf("\n Colour extends out of RGB cube, hence cannot be displayed");
   else
   {
/*      pix = (unsigned long)R + 256*(unsigned long)G + 256*256*(unsigned long)B;*
/
     pix =(unsigned long)R + ((unsigned long)G<<8) + ((unsigned long)B<<16);
     printf("\n Pixel value from RGB is = %lu",pix);
     opmode(0,6);
     for(x=0;x < 40;x++)
     { for(y=0;y < 40;y++)
         pixw(472+x,y,pix);
     }
   }
   printf("\n\n\nPRESS ANY KEY TO CONTINUE, (<esc> to quit)..");
   ch = getch();
  }/*end while*/
}

float huecalc(float m1, float m2)
/*  AIM: to calculate the hue value as an angle
 *  INPUTS:  m1 - the horizontal vector component
 *           m2 - the vvertical vector component
 *  RETURNS: hue - the derived hue value
 *  CALLS:   none
 */
       {
          float pi=3.14159265359;
          float hue;
          float angle;
```



```
       if(m1==0)
       { if(m2>0)
           hue =  90;              /* account for division by 0 */
         else if(m2<0)
           hue =  270;
         else if(m2==0)
           hue =  0;               /* default for undefined hue */
       }
       else
       {
         angle = atan(m2/m1)*180/pi;
         if ((m1>0)&&(m2<0))
          hue =  360 + angle;
         else if (m1<0)
          hue =  180 + angle;
         else
          hue =  angle;
        }
       return (hue);
       }

float satcalc(float m1, float m2, int R, int G, int B)
/*   AIM: to calculate the saturation of a colour
 *   INPUTS: satoption - the global option given in appmain.c,
 *           vectors m1 and m2 for satoption 0,
 *           RGB components for satoption 1
 *   RETURNS: the derived saturation value
 *   CALLS:  none
 */
  {

      switch(satoption)
      {
      case 0 :
              return(sqrt(m1*m1 + m2*m2));
      case 1 :
              if((R==G)&&(G==B)&&(B==0)) return(0);
              else return(255.0*(1.0 - (float)(3*min(min(R,G),min(G,B)))/(R+G+B)
));

      default: printf("\nERROR: in function satcalc"); return(0);
      }
   }


void RGBtoV1V2(int usepix,unsigned long pix,int *R,int *G,int *B,float *V1,float
*V2)
/* AIM: If usepix is TRUE convert the incomming pixel (pix) to RGB components
 *      and vectors V1 & V2, else just use the RGB components to give the
 *      H,S vectors V1 and V2 with a matrix transformation
 * INPUTS: see AIM
 * OUTPUTS: R, G, B, V1, V2, - the features derived from the input value pix
 * CALLS: none
 */
{
     /*DT7910 RGB to HSI conversion chip*/
/*     float X[3][3] = {  .333333  ,  .333333  ,  .333333 ,
                         1.547005  , -.57735   , -.57735  ,
                          .0       ,  1.0      , -1.0      };
```

```
*/      float X[3][3] = {  .333333  ,  .333333  ,  .333333 ,
                          1.0       , -.5       , -.5      ,
                           .0       ,  .866025  , -.866025 };

      /*Lehar & Stevens 1984*/
/*      float X[3][3] = {  .33333333,  .33333333,  .33333333,
                         .81649658, -.40824829, -.40824829,
                            .0    ,  .70710678, -.70710678  };
*/
        if(usepix==TRUE)
        {
          *R =  pix & 0x0000ff;         /* red value mask */
          *G = (pix & 0x00ff00) >> 8;
          *B = (pix & 0xff0000) >> 16;
        }
        *V1 =(X[1][0]**B + X[1][1]**G + X[1][2]**R);
        *V2 =(X[2][0]**B + X[2][1]**G + X[2][2]**R);
}

void calcHSIplanes(int x1,int y1,int x2,int y2,int jump)
/* AIM: To calculate the H,S,I values for each pixel in the threshold mask
 *      region. These planes will simulate the values given from hardware
 *      and stored in buffers. The planes can be written to file (for use in
 *      the SAS software (on the VAX), or to the frame buffers.
 *      General statistics can be given about the feature planes H, S, and I.
 *      Time to get these statistics is given.
 *      Options are given to segment the planes and draw H vs I scatter plots.
 *
 * INPUTS: -the top left and bottom rights coordinates of the area of interest
 *         -the jump value for altenate no. of scan lines
 *         -the area of the mask from 'long STATS.aoapple'
 * OUTPUTS: none.
 * CALLS: RGBtoV1V2,huecalc,satcalc,segment,appgraph,
 *        moment, stats2-d, imatrix, free_imatrix,
 *        uses global STATS structure
 */
{   int option;                         /* write to file or to monitor HSI planes*/
    int choice;                         /* draw scatter plot or not */
    int returnval=0;                    /* 0=okay, 1,2,3=memory problem */
    int x,y;                            /* index to pixel in aoi */
    int pixcnt=1;                       /* index to pixel count in mask */
    unsigned long pix;                  /* pixel read */
    float V1,V2;                        /* H,S horizontal and vertical vectors*/
    int R,G,B;                          /* colour components of pix */
    int H,S,I;                          /* values for HSI planes on monitor */
    int *Hplane,*Splane,*Iplane;        /* H,S,I planes which will be given */
                                        /* eventually from hardware */
    long Hcount=0;                      /* a count of all hue values to give Have*/
    long Icount=0;
    long Scount=0;
    float Have,Hadev,Hsdev,Hsvar,Hskew,Hcurt; /* the average hue & stats*/
    float Iave,Iadev,Isdev,Isvar,Iskew,Icurt; /* the average intnsty & stats*/
    float Save;
    char xlabel[80];
    char ylabel[80];
    char title[80];

    int *Hhist;                         /* array of hue histogram */

    int **HbyI;                         /* array of accumulated H by I values */
```



```
    int Hmax=0,Hmin=360,Imax=0,Imin=255,Smax=0,Smin=255;
    int h,i,j;                          /* indexes to the HbyI array */
    int HbyIpeak=0, Hpeak=0, Ipeak=0;
    FILE *fparea,*fpH,*fpI,*fpS,*fpHbyI;

    struct dostime_t tim0, tim1;        /* timing before & after process */
    float timediff;

    if( (Hplane=(int *)calloc((int)(STATS.aoapple+1),sizeof(int)))==NULL ) return
val=1;
    if( (Splane=(int *)calloc((int)(STATS.aoapple+1),sizeof(int)))==NULL ) return
val=2;
    if( (Iplane=(int *)calloc((int)(STATS.aoapple+1),sizeof(int)))==NULL ) return
val=3;
  if (returnval!=0)
  { printf("\nERROR: Out of memory on this size area_of_interest");
    switch(returnval)
      {
      case 1: break;
      case 2: free(Hplane); break;
      case 3: free(Hplane); free(Splane); break;
      }
    printf("\n      Statistical results will not be given and HSI values will rep
lace RGB values");
       /* out of memory option to replace RGB with HSI and no stats */
    printf("\n Writing H to buffer 0, S to buffer 1, I to buffer 3. ");

            for (y=y1 ; y < y2 ; y=y+jump)
            {
              for(x=x1 ; x < x2 ; x++)    /* process the line */
              {
               opmode (0,2);
               if(pixr(x,y) != 0)                /* check if in apple */
               {
                 opmode(0,6);               /* max pixr = 0xffffff */
                 pix = pixr(x,y);
                 RGBtoV1V2(TRUE,pix,&R,&G,&B,&V1,&V2);
                 H = (int)huecalc(V1,V2);
                 opmode(0,0); pixw(x,y,H);
                 S = (int)satcalc(V1,V2,R,G,B);
                 opmode(0,1); pixw(x,y,S);
                 I = (R+G+B)/3;
                 opmode(0,3); pixw(x,y,I);
                 opmode(0,2); pixw(x,y,(int)(.3*R+.59*G+.11*B));        /* change
 colour of mask to Luminance Y*/

               }
              }
            }
  }
  else
  {

    printf("\n\nChoose (1) Replace RGB planes with HSI planes or\n       (2) Leave
 RGB planes and write HSI planes to file : ");
    option = getche();
    printf("\n\nCalculating H,S and I planes in the mask...");

    switch(option)
    {
```

```
    case '1' :
          printf("\n Writing H to buffer 0, S to buffer 1, I to buffer 3. ");

          for (y=y1 ; y < y2 ; y=y+jump)
          {
            for(x=x1 ; x < x2 ; x++)    /* process the line */
            {
             opmode (0,2);
             if(pixr(x,y) != 0)                /* check if in apple */
             {
               opmode(0,6);               /* max pixr = 0xffffff */
               pix = pixr(x,y);
               RGBtoV1V2(TRUE,pix,&R,&G,&B,&V1,&V2);
               Hplane[pixcnt] = (int)huecalc(V1,V2);
               opmode(0,0); pixw(x,y,Hplane[pixcnt]);
               Splane[pixcnt] = (int)satcalc(V1,V2,R,G,B);
               opmode(0,1); pixw(x,y,Splane[pixcnt]);
               Iplane[pixcnt] = (R+G+B)/3;
               opmode(0,3); pixw(x,y,Iplane[pixcnt]);
               opmode(0,2); pixw(x,y,(int)(.3*R+.59*G+.11*B));        /* change
colour of mask to Luminance Y*/
               pixcnt++;
             }
            }
          }
          break;

    case '2' :
          if((fparea = fopen("area.dat","w")) == NULL) returnval=5;
          if((fpH = fopen("hue.dat","w")) == NULL) returnval=5;
          if((fpS = fopen("sat.dat","w")) == NULL) returnval=5;
          if((fpI = fopen("intensity.dat","w")) == NULL) returnval=5;
          if(returnval != 0)
          { printf("\nERROR: unable to write data .dat files ");
            getch();
            return;
          }
          fprintf(fparea,"%ld ",STATS.aoapple);

          for (y=y1 ; y < y2 ; y=y+jump)
          {
            for(x=x1 ; x < x2 ; x++)    /* process the line */
            {
             opmode (0,2);
             if(pixr(x,y) != 0)                /* check if in apple */
             {
               opmode(0,6);               /* max pixr = 0xffffff */
               pix = pixr(x,y);
               RGBtoV1V2(TRUE,pix,&R,&G,&B,&V1,&V2);
               Hplane[pixcnt] = (int)huecalc(V1,V2);
               fprintf(fpH,"%d\n",Hplane[pixcnt]);
               Splane[pixcnt] = (int)satcalc(V1,V2,R,G,B);
               fprintf(fpS,"%d\n",Splane[pixcnt]);
               Iplane[pixcnt] = (R+G+B)/3;
               fprintf(fpI,"%d\n",Iplane[pixcnt]);
               opmode(0,2);
               pixw(x,y,Iplane[pixcnt]);
               pixcnt++;
             }
            }
```



```
          }
          break;
      default : return;
    } /* end switch*/

    Hhist=(int *)calloc(360,sizeof(int)); /* used for segmentation */

    /* calculated average hue */
    printf("\nCalculating Hue, Saturation & Intensity statistics...");
     dos_gettime(&tim0);
    for(pixcnt=1; pixcnt<=STATS.aoapple ; pixcnt++)
    {
     if(Hplane[pixcnt] < Hmin) Hmin=Hplane[pixcnt];
     if(Hplane[pixcnt] > Hmax) Hmax=Hplane[pixcnt];
     Hcount += (long)Hplane[pixcnt];
     if((0 <=Iplane[pixcnt])&&(Iplane[pixcnt] < Imin)) Imin=Iplane[pixcnt];
     if((255 >=Iplane[pixcnt])&&(Iplane[pixcnt] > Imax)) Imax=Iplane[pixcnt];
     Icount += (long)Iplane[pixcnt];
     if((0 <=Splane[pixcnt])&&(Splane[pixcnt] < Smin)) Smin=Splane[pixcnt];
     if((255 >=Splane[pixcnt])&&(Splane[pixcnt] > Smax)) Smax=Splane[pixcnt];
     Scount += (long)Splane[pixcnt];
    }
    Have = ((float)Hcount)/STATS.aoapple;
    Iave = ((float)Icount)/STATS.aoapple;
    Save = ((float)Scount)/STATS.aoapple;

    _dos_gettime(&tim1);
    printf("\n\n\t\tAverage\tMaximum\tMinimum");
    printf(  "\nHue\t\t%5.2f\t%d\t%d",Have,Hmax,Hmin);
    printf(  "\nSaturation\t%5.2f\t%d\t%d",Save,Smax,Smin);
    printf(  "\nIntensity\t%5.2f\t%d\t%d",Iave,Imax,Imin);
    timediff =(float)(tim1.minute)*60.0 + (float)(tim1.second) + (float)(tim1.hse
cond)/100.0 - (float)(tim0.minute)*60.0 - (float)(tim0.second) - (float)(tim0.hse
cond)/100.0 ;
    printf(  "\nTime to get statistics = %f secs", timediff);

    printf("\n\nView more statistical results (y/n) ? ");
    choice = getche();
    if (choice=='y')
    {
     if((Hmax<Hmin)||(Imax<Imin)) {printf("\nError in sizing"); getch(); exit(1);
}
     moment(Hplane,(int)STATS.aoapple,&Have,&Hadev,&Hsdev,&Hsvar,&Hskew,&Hcurt);/
* get stats for Hue */
     printf("\n\tAverage \tStddev  \tVariance\tSkew    ");
     printf("\nHue\t%f\t%f\t%f\t%f",Have,Hsdev,Hsvar,Hskew);
     moment(Iplane,(int)STATS.aoapple,&Iave,&Iadev,&Isdev,&Isvar,&Iskew,&Icurt);/
* get stats for Intensity*/
     printf("\nIntnsty\t%f\t%f\t%f\t%f",Iave,Isdev,Isvar,Iskew);
     stats2_d(Hplane,Iplane,(int)STATS.aoapple);
    }
    /* constructing H by I array */
    printf("\n\nCalculating array of Hue vs Intensity ...");
    HbyI = imatrix(Hmin,Hmax,Imin,Imax);
    for(h=Hmin;h<=Hmax;h++)                    /* initialise HbyI */
        for(i=Imin;i<=Imax;i++)
           HbyI[h][i]=0;
    for (j=1; j<=(int)STATS.aoapple ; j++)
        if((Hplane[j]>=52)&&(Hplane[j]<=307))
        {
```

```
        HbyI[Hplane[j]][Iplane[j]]++;      /* accumulate at H,I position*/
        if(HbyI[Hplane[j]][Iplane[j]] >= HbyIpeak)
        { HbyIpeak = HbyI[Hplane[j]][Iplane[j]];
          Hpeak = Hplane[j];
          Ipeak = Iplane[j];
        }
        Hhist[Hplane[j]]++;                /* accumulate hue histogram */
      }
   printf("\nPeak of %d at Hue = %d , Intensity = %d",HbyIpeak,Hpeak,Ipeak);
   printf("\nWrite H by I file (y/n): ");
   choice = getche();
   if(choice=='y')
   {
       if((fpHbyI = fopen("hbyi.dat","w")) == NULL) returnval=5;
       if(returnval != 0)
       { printf("\nERROR: unable to write data .dat files ");
         getch();
         return;
       }

       for (h=Hmin; h<=Hmax ; h++)
       {       for(i=Imin; i<=Imax ;i++)
               fprintf(fpHbyI,"%d,",HbyI[h][i]);
               fprintf(fpHbyI,"\n");
       }
       fclose(fpHbyI);
   }
   /* segment hue range */
   segment(Hhist,360);


   /* draw scatterplot of H vs I */
   printf("\n\nDraw scatter plot of H vs I (y/n) ? ");
   choice = getche();
   if(choice == 'y')
   {
    strcpy(ylabel,"Hue values");
    strcpy(xlabel,"Intensity values");
    strcpy(title,"Scatter plot of H vs I values for region of interest");
    appgraph(Hplane,Iplane,(int)STATS.aoapple,ylabel,xlabel,title);
   }
   if(option=='1')
    { fclose(fparea); fclose(fpH); fclose(fpI); fclose(fpS); }
   free(Hplane); free(Splane); free(Iplane);
   free(Hhist);
   free_imatrix(HbyI,Hmin,Hmax,Imin,Imax);
  }/*end else*/
}

void thresh(wincoords *procwindow)
/* AIM: To threshold one of the colour buffers (red if background is black
 *      and blue if background is white) to produce a mask
 *      which contains the fruit regions (FFh) and 0 if not in the fruit.
 *      To calculate the value of the global value STATS.aoapple.
 *      Eventually the boundary of the fruit should be given instead of
 *      determining the boundary with a threshold.
 *
 * INPUTS: procwindow - the processing window of interest
 *         STATS.aoapple - the area of the thresholded region (area of apple)
 * CALLS: calcHSIplanes
```



```
*/
{
   int xstart,xstop,ystart,ystop;  /* area of interest */
   int j,k;
   int level;                      /* threshold level */
   int jump=1;                     /* jump to alternate no. of scan lines*/
   int backgrnd;                   /* the background colour of b or w */
   int inlimit;                    /* boolean for in the thresh limit */
   int choice = 'n';               /* 'y' if threshold was satisfactory*/

   xstart = procwindow->x1;
   xstop  = procwindow->x2;
   ystart = procwindow->y1;            /* area of interesting region */
   ystop  = procwindow->y2;

  while (choice == 'n')
  {
   clearscreen(_GCLEARSCREEN);
   STATS.aoapple = 0;                   /* area of the inapple region */

   printf("Enter threshold level:");
   scanf("%d",&level);
   printf("\nThreshold against white or black background (w/b) ? ");
   backgrnd = getche();
   printf("\nEnter number of alternate scan lines : ");
   scanf("%d",&jump);
   if(jump > (ystop-ystart) )
   { printf("ERROR: jump step is out of range of window height");
     getch();
     continue;
   }
   if((backgrnd=='w')||(backgrnd=='b'))
   {
    printf("\n Thresholding in process...");
    opmode (2,2);
    clear (2,0);                                /* clear overlay buffer first */
    for (j=ystart ; j < ystop ; j=j+jump)
    {
     for(k=xstart; k < xstop; k++)               /* process line */
     {
       if(backgrnd=='w')
       { opmode(0,3);                           /* choose blue buffer */
         inlimit = (pixr(k,j) < level);
       }
       else
       {  opmode(0,0);                          /* choose red buffer */
          inlimit = (pixr(k,j) > level);
       }
       if(inlimit)
       {  opmode(0,2);
          pixw(k,j, 0xff);
          STATS.aoapple++;
       }
       else
       {   opmode(0,2);
           pixw(k,j, 0x0);
       }
     }
    }
   }
```

```
    else break;
    printf("\n\nArea of mask = %ld pixels",STATS.aoapple);
    printf("\n\nWas threshold level sufficient (y/n) ? ");
    choice = getche();
 /*   if(STATS.aoapple>32767l) */        /* NB!!!! since scatter plots can only t
ake unsigned short no. of points (32767)  */
 /*   {
      printf("\n\nERROR: mask region must be <= 32767 for scatter plots");
      opmode(2,2);
      clear(2,0);
      printf("\nPress any key to redefine an area_of_interest...");
      getch();
      return;
    }         */
   }/*end while*/
   printf("\nSimulate HSI plane, calculate statistics, draw scatterplot (y/n) ? "
);
   if ( (choice=getche()) == 'y' )
     calcHSIplanes(xstart,ystart,xstop,ystop,jump);

 }/*end thresh */

void avgPAL(wincoords *procwindow)
/*  AIM: To average out alternate lines in the image of R,G & B files
       so as to eliminate the band effect given by a decoded PAL signal
   CALLS: none
 */
{
    int xstart,xstop,ystart,ystop;  /* area of interest */
    int fbuf[3] = {0,1,3} ;
    int pixinten;            /* intensity of a pixel */
    int i,j,k;

    xstart = procwindow->x1;
    xstop  = procwindow->x2;
    ystart = procwindow->y1;            /* area of interesting region */
    ystop  = procwindow->y2;

    printf("Averaging in process...");
    for(i=0; i<3; i++)
    {
      opmode (0,fbuf[i]);
      for(j = ystart; j < ystop ; j=j+2)
      {
         for (k=xstart; k < xstop; k++)
         {
            pixinten = (pixr(k,j) + pixr(k,j+1))/2;
            pixw(k,j,pixinten);
            pixw(k,j+1,pixinten);
         }
      }
    }
 }/*end avgPAL */

void linehist(int x1,int y1,int x2,int y2)
/*  AIM: To draw the R,G,B,H,S or I distribution along a given line starting
 *       at (x1,y1) and ending at (x2,y2). The function uses the MVP-AT
 *       function 'rvector' to read in the R,G and B components on the line
 *  INPUTS: see AIM
 *  CALLS: RGBtoV1V2,huecalc,satcalc,appgraph
```



```
 */
{
    int  *Rline,*Gline,*Bline;  /* R,G,B line arrays */
    int  *Hline, *Sline,*Iline; /* H,S,I line arrays */
    int  *pixno;                /* linear array of pixel numbers from the line or
igin*/
    float V1,V2;                /* H,S vectors */
    int  length;                /* the line length in # pixels (last pix is not i
ncluded)*/
    int i;                      /* index to position of pixel on line */
    int choice='d';             /* draw data or graphs  */
    int viewval;                /* view value being r,g,b,h,s,i */
    char valname[80];           /* text label for y axis */
    char xlabel[80];            /* text label for x axis */
    char title[80];             /* text label for title */

    length = max(abs(x2-x1),abs(y2-y1)); /* length is also given at
                                            Rline[0],Gline[0],etc.        */
    Rline = (int *)calloc((length+1),sizeof(int));
    Gline = (int *)calloc((length+1),sizeof(int));
    Bline = (int *)calloc((length+1),sizeof(int));
    Hline = (int *)calloc((length+1),sizeof(int));
    Sline = (int *)calloc((length+1),sizeof(int));
    Iline = (int *)calloc((length+1),sizeof(int));
    pixno = (int *)calloc((length+1),sizeof(int));
    opmode (2,0);                  /* processing mode on red buffer */
    rvector(FB0,Rline,x1,y1,x2,y2); /* read vector (frambuffer,work buffer */
                                    /*          initial,final coords)      */
    opmode (2,1);
    rvector(FB1,Gline,x1,y1,x2,y2);
    opmode (2,3);
    rvector(FB3,Bline,x1,y1,x2,y2);
    pixno[0]=0;
 /*   printf("R[%d]=%d\tG[%d]=%d\tB[%d]=%d\n",0,Rline[0],0,Gline[0],0,Bline[0]);
 */
    printf("\n Calculating R,G,B,H,S and I values...");
    for(i=1 ;i<(length+1) ; i++)
    {
      RGBtoV1V2(FALSE,0,&Rline[i],&Gline[i],&Bline[i],&V1,&V2);
      Hline[i] = (int)huecalc(V1,V2);          /* ignore H,S,Iline[0] */
      Sline[i] = (int)satcalc(V1,V2,Rline[i],Gline[i],Bline[i]);
      Iline[i] = (Rline[i]+Gline[i]+Bline[i])/3;
      pixno[i] = i;
     }


   while ((choice=='d')||(choice=='g'))
   {
    _clearscreen(_GCLEARSCREEN);
    printf("-- Histograms of line containing %d pixels --\n\n",length);
    printf("Show data or graphs on this screen (d/g) ?\n ");
    choice = getch();
    if ((choice!='d')&&(choice!='g')) break;
    if (choice=='d')
    {
      for(i=1; i<(length+1); i++)
      {
        printf("R[%d]=%d  G[%d]=%d  B[%d]=%d  H[%d]=%d  S[%d]=%d  I[%d]=%d\n"\
        ,i,Rline[i],i,Gline[i],i,Bline[i],i,Hline[i],i,Sline[i],i,Iline[i]);
        if(((i%20)==0)||(i==length))
```

```
        {
          printf("\n\n Continue ?...\n");
          getch();
        }
      }/* end for i */
    }/* end if choice d*/
    if (choice == 'g')
    {
      printf("\n\n\n View distribution of R,G,B,H,S or I ? ");
      viewval = getche();
      strcpy(xlabel,"Pixel number from line origin");
      switch (viewval)
      {
        case 'r': strcpy(valname,"Red");
                  strcpy(title,"Red Value Distribution Along The Line");
                  appgraph(Rline,pixno,length,valname,xlabel,title); break;
        case 'g': strcpy(valname,"Green");
                  strcpy(title,"Green Value Distribution Along The Line");
                  appgraph(Gline,pixno,length,valname,xlabel,title); break;
        case 'b': strcpy(valname,"Blue");
                  strcpy(title,"Blue Value Distribution Along The Line");
                  appgraph(Bline,pixno,length,valname,xlabel,title); break;
        case 'h': strcpy(valname,"Hue");
                  strcpy(title,"Hue Value Distribution Along The Line");
                  appgraph(Hline,pixno,length,valname,xlabel,title); break;
        case 's': strcpy(valname,"Saturation");
                  strcpy(title,"Saturation Value Distribution Along The Line");
                  appgraph(Sline,pixno,length,valname,xlabel,title); break;
        case 'i': strcpy(valname,"Intensity");
                  strcpy(title,"Intensity Value Distribution Along The Line");
                  appgraph(Iline,pixno,length,valname,xlabel,title); break;
        default : break;
       }
    }
   }/* end while */
   free(Rline); free(Gline); free(Bline);
   free(Hline); free(Sline); free(Iline);
}

void pixoi(void)
/* AIM: To give the R,G,B,H,S,I values of a pixel that is enclosed in a
 *      rectangle when the mask view is on. Arrow keys move the cursor
 *      over the pixel of interest, s steps the cursor 20 or 1 pixel.
 * INPUTS: pix - the value read from a pixel seen on the monitor
 * OUTPUTS: to the screen the R,G,B,H,S,I values of the pixel (pix).
 * CALLS: linehist,scancode,RGBtoV1V2, huecalc, satcalc
 */
{
  enum {SMALL, LARGE} increment;

  int exit;
  int returnvalue;
  int keycode;
  int linedrw;                    /* boolean,linedrw=TRUE means draw a line */
  int step = 1;                   /* Starting increment for box movement */
  static int x=256, y=256;        /* Old coordinate values */
  int nx, ny;                     /* New coordinate values */
  int xmark,ymark;                /* mark the origin of the drawn line */
  unsigned long pix;              /* pixel value containing 8 bit R,G & B*/
  int R,G,B;
```



```
  float V1,V2;
  int H,S,I;

  opmode(1,2);                    /* set graphics mode */
  drawmode(2);                    /* draw in XOR mode */
  increment = SMALL;
  exit = FALSE;
  linedrw = FALSE;
  nx=x;
  ny=y;

  if((x==256)&&(y==256))
  {
    move(x-1,y-1);
    rect(x+1,y+1);                /* Draw initial required rectangle */
  }

  do
  {
      keycode = scancode();
      switch (keycode)
        {
         case ESC    : if(linedrw==TRUE)
                       {
                         opmode(1,2);
                         move(xmark,ymark);
                         line(x,y);              /* erase previous line*/
                         x = xmark;
                         y = ymark;
                         linedrw = FALSE;        /* return to single cursor*/
                       }
                       else
                       {
                        exit = TRUE;
                        returnvalue = 0;
                       }
                       break;
           case RETURN : if(linedrw==FALSE)
                       {
                         linedrw = TRUE;
                         xmark= x;
                         ymark= y;
                       }
                       else
                       { linehist(xmark,ymark,x,y);
                         exit = TRUE;
                         returnvalue = 1;
                       }
                       break;
           case UARROW :
                       if ((y-step) < 0)
                          ny = 512 + (y - step);
                       else
                          ny = y - step;
                       break;
           case DARROW :
                       if ((y+step) > 511)
                          ny = (y + step) - 512;
                       else
                          ny = y + step;
```

```
                    break;
        case LARROW :
                    if ((x-step) < 0)
                       nx = 512 + x - step;
                    else
                       nx = x - step;
                    break;
        case RARROW :
                    if ((x+step) > 511)
                       nx = x + step - 512;
                    else
                       nx = x + step;
                    break;
        case STEP   : if (increment == LARGE)
                       {
                         step = 1;                          /* Fine step */
                         increment = SMALL;
                       }
                      else
                       {
                         step = 20;                        /* Coarse step */
                         increment = LARGE;
                       }
                      break;
        default     : putchar(BELL);
       }

      opmode(1,2);
      if(linedrw==FALSE)
      {
       move(x-1,y-1);
       rect(x+1,y+1);                /* erase the previous box cursor */
       x = nx;  y = ny;
       move(x-1,y-1);
       rect(x+1,y+1);                /* redraw cursor at new position */
      }
      else /* linedrw==TRUE */
      {
        move(xmark,ymark);           /* erase previous line */
        line(x,y);
        x = nx; y = ny;
        move(xmark,ymark);
        line(x,y);                   /* redraw line at new position */
      }
      opmode(0,6);
      pix = pixr(x,y);
      RGBtoV1V2(TRUE,pix,&R,&G,&B,&V1,&V2);
      H = (int)huecalc(V1,V2);
      S = (int)satcalc(V1,V2,R,G,B);
      I = (R+G+B)/3;

      _clearscreen(_GCLEARSCREEN);
      printf("(%d,%d)\n\n",x,y);
      printf("R = %d\nG = %d\nB = %d\n\n",R,G,B);
      printf("H = %d\nS = %d\nI = %d\n\n",H,S,I);

    } while (exit != TRUE);

/*  return(returnvalue);*/
}
```

```
/*****************************************************************************
 * MODULE: APPUTIL
 *
 * AUTHOR: G.Kay
 *
 * DESCRIPTION: This module contains general utility routines using the
 *              MVP-AT functions. These are mainly screen handling and general
 *              I/O utilities.
 * CONTAINS:
 *          clearbuff
 *          clean
 *          erase
 *          adjustolut
 *          maskview
 *          changeop
 *          viewbuff
 *          init_aoi
 *          zoom
 *          cut_paste
 *          aoi
 *          scancode
 *          colograb
 *          loadfile
 *          savefile
 *****************************************************************************/

#include "appcom.h"
#include "apptype.c"
#include "appdefn.c"

/************************GLOBAL OPTION PARAMETERS**************************/
extern int satoption ;   /*0 = sat as vector dist, 1= as rgb ratio to intensity*/


/*******************************FUNCTION CODE*********************************/
void clearbuff(void)
/* AIM: to clear a specified frame buffer
 * INPUTS: frame buffer to clear
 */
{
  int framebuffer;   /* frame 0=red,1=green,2=overlay,3=blue,
                              4=0 and 1, 5=2 and 3, 6=0,1,2 and 3 */

  opmode(2,0);
  _clearscreen(_GCLEARSCREEN);
  printf("Enter frame buffer to clear (0,1,2,3,4,5 or 6): ");
  scanf("%d",&framebuffer);
  if (framebuffer == 6)
  { clear (4,0); clear(5,0); }
  else
    clear (framebuffer,0);
}/* end clearbuff */

void clean(int xt,int xb,int yt,int yb)
/*AIM: To clean a specified window to black in frame buffer 6
 *INPUTS: top left and bottom right coordinates of window to clean to black.
 */

{
  int i;
```



```
  opmode (1,0);
  drawmode(0);     /* replaces affected pixels with black */
  setcolor(0);     /* black */
  for (i=0; i<4; i++)
  {
    opmode(1,i);
    move(xt,yt);
    frect(xb,yb);
  }
  setcolor(0xffff);   /* set back to white */
  opmode(0,6);
}

void erase(wincoords *procwindow)
/* AIM: to erase area above, below, left or right of processing window
 * INPUTS: procwindow - the processing window of interest
 * CALLS:  erase.
 */
{  int ch;   /* choice of area to erase */

    _clearscreen(_GCLEARSCREEN);
    printf("1: Erase area above window of interest\n\n");
    printf("2: Erase area below window of interest\n\n");
    printf("3: Erase area left of window of interest\n\n");
    printf("4: Erase area right of window of interest\n\n");

    ch=getche();
    switch(ch)
    {
      case '2': clean(0,col,procwindow->y2,row); break;
      case '1': clean(0,col,0,procwindow->y1);   break;
      case '3': clean(0,procwindow->x1,0,row);   break;
      case '4': clean(procwindow->x2,col,0,row); break;
      default :break;
     }
}

void adjustolut(void)
/* AIM: To use an MVP-AT macro to adjust the LUT of the overlay mask. There
 *      are 15 colours assigned to every 15 levels from 0 to 255, for pixels
 *      in the overlay of framebuffer 2.
 *      Colours are:1=green, 2=yellow, 3=red, 4=magenta, 5=blue, 6=cyan,
 *      7=darkgreen, 8=brown, 9=darkred, 10=darkmagenta, 11=darkblue,
 *      12=darkcyan, 13=darkgrey, 14=lightgrey, 15=white.
 * INPUTS: none.
 * OUTPUTS: adjusted overlay LUT values.
 */
{
     BYTE or[15]={0,0xff,0xff,0xff,0,0,0,0x7f,0x7f,0x7f,0,0,0x3f,0x7f,0xff};
     BYTE og[15]={0xff,0xff,0,0,0,0xff,0x7f,0x3f,0,0,0,0x7f,0x3f,0x7f,0   };
     BYTE ob[15]={0,0,0,0xff,0xff,0xff,0,0,0,0x7f,0x7f,0x7f,0x3f,0x7f,0   };
     /* overlay r,g,b LUT arrays */
     unsigned int ostrt[15]; /* overlay work buffer start address */
     BYTE bkl[256],bkh[256]; /* background r,g,b LUT (low byte) and hi byte locat
ions*/
     unsigned int bkhl[256]; /* background lo and hi bytes*/
     int i;

/*     opmode(0,2);
```

```
     for(i=0;i<256;i++)
       pixw(100+i,256,(unsigned long)i);
*/
/*   &ostrt=0;
     &or = 0x20;
     &og = 0x30;
     &ob = 0x40;
     &bkl = 0x2000;
     &bkh = 0x3000;
     &bkhl= 0x100;
*/
     opmode(2,0);

     scaling(0,0,255,255,bkhl); /* at location 100, 0-255 ramp for 256 levels*/
     cwb(bkhl,bkl,bkh,256);/* put this normal ramp at loction 2000 */

     olutlay(1,or,og,ob,bkl,bkl,bkl);
     /* place in the  output overlay LUT (1) the overlay RGB LUT locations
        (20,30,40) and the background image  RGB LUT locations (all at 2000)*/
     slut(1,16);  /* select output palette for LUT */
     outpath(6,-1,2,1);  /* display image with overlay */

}

void maskview(int *MASK)
/* AIM: To toggle viewing of the overlay mask. This feature enables
 *      or disables viewing of the fruit under the non destructive mask
 * INPUTS: MASK - indicating the overlay is showing (0) or not (1)
 * OUTPUTS: MASK
 */
{
     if (*MASK==1)
     { outpath(6,-1,2,2);       /* show the mask */
       *MASK = 0;
     }
     else
     { outpath(6,-1,0,0);       /* dont show the mask */
       *MASK = 1;
     }
}

void changeop(void)
/*  AIM: to change initially setup options e.g. choose option of method to
 *       calculate saturation. Could be extended to change window set ups
 *       and alternative I calculations.
 *  INPUTS: satoption - a global variable defined in appmain.c
 *  OUTPUTS: satoption - altered or not
 */
{
     int choice; /* users choice value */

     _clearscreen(_GCLEARSCREEN);
     printf("-- Global Options --");
     printf("\n\n Saturation option = %d", satoption);
     printf("\n\n\n New saturation option (0:vector dist, 1:RGB ratio to I) = ");
     choice = getche();
     switch (choice)
     {
       case '0' : satoption = 0; break;
       case '1' : satoption = 1; break;
```



```
       default  : break;
     }

}


void viewbuff(void)
/* AIM: to view a specified framebuffer
 * INPUTS: from keyboard, user specified framebuffer
 * OUTPUTS: the specified buffer to the RGB monitor screen.
 */
{
  int framebuffer;

  opmode(2,0);
  _clearscreen(_GCLEARSCREEN);
  printf("The present default viewing is FB6 with overlay FB2\n\n");
  printf("Enter frame buffer to view (0,1,2,3 or 6): ");
  scanf("%d",&framebuffer);

  outpath(framebuffer,-1, 0, 0);
  printf("\nPress any key to return to default viewing...");
  getch();
}/*end viewbuff*/


void init_aoi(wincoords *procwindow,int xt,int yt,int xb,int yb)
/* AIM : Initialises the data proc. win. coords to a set an area
 *       of interest
 * INPUTS : Pointer to structure containing proc. win. coordinates
 *          topleft coordinates and bottom right coordinates
 * OUTPUTS: None
 */
{
  procwindow->x1 = xt;    procwindow->x2 = xb;
  procwindow->y1 = yt;    procwindow->y2 = yb;
}

void zoom(MODE)
/* AIM: To zoom onto an area specified in continuous grab mode (3)
 *      or processing mode (2). Arrow keys move the window to zoom, <Enter>
 *      zooms on 8x magnification, <esc> quits.
 * INPUTS: MODE - see AIM
 * CALLS:  init_aoi, aoi.
 */
{  wincoords *zoomwin;     /* zoom window coordinates */
   int xt,yt;              /* top left coords of zoom window */
   int show;               /* show =1 means continue, show=0 means exit */

/*  printf("\nStep mode on the window must be used");
*/  opmode (MODE,6);
  /* Allocate storage for zoom window coordinates */
  zoomwin = (wincoords *) malloc(sizeof(wincoords));
  init_aoi(zoomwin,240,240,280,280);    /* Initialise zoom win. coordinates */
  while((show = aoi(zoomwin,FB2)) != 0)        /* gets xt yt */
  {  xt = zoomwin->x1;
     yt = zoomwin->y1;
     pan(xt);
     scroll(yt);
     outzoom(8,8);
```

```
    getch();
    outzoom(1,1);
    pan(0);
    scroll(0);
  }
  free(zoomwin);
}

void cut_paste(void)
/* AIM: To cut a rectangle from a specified frame buffer and paste it to
 *      any other frame buffer
 * INPUTS: users specified window (arrow keys), and frame buffers.
 * OUTPUTS: the pasted region onto the specified region
 * CALLS:  init_aoi, aoi.
 */
{
    wincoords *cutwin;   /* cutting window coordinates */
    wincoords *pastewin; /* pasting window coordinates */
    int sbuf, dbuf;      /* source and destination frame buffers */
    int xt=154, yt=17;   /* top left coords of cutwin */
    int xb=342, yb=246;  /* bottom right coords of cutwin*/
    int xd, yd;          /* top left coords of pastewin*/

    cutwin = (wincoords *)malloc(sizeof(wincoords));
    pastewin = (wincoords *)malloc(sizeof(wincoords));
    opmode(2,6);
    init_aoi(cutwin,xt,yt,xb,yb);
    _clearscreen(_GCLEARSCREEN);
    printf("-- Cutting and Pasting --");
    printf("\n\nEnter frame buffer to cut rectangle from (0,1,2,3) : ");
    scanf("%d",&sbuf);
    opmode(2,sbuf);
    while(aoi(cutwin,sbuf)!=1);
    xt = cutwin->x1;
    yt = cutwin->y1;
    xb = cutwin->x2;
    yb = cutwin->y2;
    printf("\nEnter frame buffer to paste rectangle to (0,1,2,3) : ");
    scanf("%d",&dbuf);
    init_aoi(pastewin,xt,yt,xb,yb);
    opmode(2,dbuf);
    while(aoi(pastewin,dbuf)!=1);
    xd = pastewin->x1;
    yd = pastewin->y1;
    opmode(2,6);
    pixblt(sbuf,xt,yt,xb,yb,dbuf,xd,yd,0,1); /*MVPAT function call */

    free(cutwin); free(pastewin);

}


int aoi(wincoords *window, int framebuffer)
/* AIM    : Enables the user to set the required size of the MVP-AT
 *          processing window. Arrow keys move the whole window at steps of
 *          20, s=toggle of steps between 1 or 20, f=toggle to fix or
 *          release top left corner, <enter>to quit with desired area obtained
 * INPUTS : Pointer to structure containing proc. win. coordinates
 *          Frame buffer to be set. Global #definitions of key values.
 * RETURNS: returnval - 0, or 1 indicating <esc> or <enter>.
```



```
 * CALLS  : scancode.
 */
{
  enum {ON, OFF} fixcoords;
  enum {SMALL, LARGE} increment;
/*  enum {TRUE, FALSE} boolean; */

  int exit;
  int returnvalue;
  int keycode;
  int step = 20;                          /* Starting increment for box movement *
/
  int ox1, ox2, oy1, oy2;                         /* Old coordinate values */
  int nx1, nx2, ny1, ny2;                         /* New coordinate values */
  int incx1, incy1, incx2, incy2;        /* Incremental direction values */
  BYTE rowbuff1[512];              /* Buffers for storing the sides of the */
  BYTE rowbuff2[512];              /* rectangle, indicating the area of    */
  BYTE colbuff1[512];              /* interest. Max aoi is thus 512x512    */
  BYTE colbuff2[512];

  ox1 = window->x1;                     /* Initialise aoi coordinates */
  oy1 = window->y1;
  ox2 = window->x2;
  oy2 = window->y2;

  opmode(MODE1,framebuffer);
  drawmode(2);                             /* draw in XOR mode */
  fixcoords = OFF;
  increment = LARGE;
  exit = FALSE;
  do
    {
      opmode(MODE0,framebuffer);                    /* Set I/O mode */
      rowr(ox1,oy1,ox2-ox1+1,rowbuff1);
      rowr(ox1,oy2,ox2-ox1+1,rowbuff2);
      colr(ox1,oy1,oy2-oy1+1,colbuff1);
      colr(ox2,oy1,oy2-oy1+1,colbuff2);
      opmode(MODE1,framebuffer);                    /* Set graphics mode */
      move(ox1,oy1);
      rect(ox2,oy2);                          /* Draw required rectangle */

      incx1 = 0; incy1 = 0; incx2 = 0; incy2 = 0;      /* Set increments */
      keycode = scancode();
      switch (keycode)
        {
          case ESC    : exit = TRUE;
                        returnvalue = 0;
                        break;
          case RETURN : exit = TRUE;
                        returnvalue = 1;
                        break;
          case UARROW : if (fixcoords == OFF)
                          if ((oy1-step)<0)
                            putchar(BELL);     /* Error - invalid step */
                        else
                          {
                            incx1 = 0; incy1 = -step;
                            incx2 = 0; incy2 = -step;
                          }
                        else
```

```
                    if ((oy2-step)<oy1)
                       putchar(BELL);
                    else
                       {
                         incx1 = 0; incy1 = 0;
                         incx2 = 0; incy2 = -step;
                       }
                  break;
    case DARROW : if (fixcoords == OFF)
                    if ((oy2+step)>511)
                       putchar(BELL);
                    else
                       {
                         incx1 = 0; incy1 = step;
                         incx2 = 0; incy2 = step;
                       }
                  else
                    if ((oy2+step)>511)
                       putchar(BELL);
                    else
                       {
                         incx1 = 0; incy1 = 0;
                         incx2 = 0; incy2 = step;
                       }
                  break;
    case LARROW : if (fixcoords == OFF)
                    if ((ox1-step)<0)
                       putchar(BELL);
                    else
                       {
                         incx1 = -step; incy1 = 0;
                         incx2 = -step; incy2 = 0;
                       }
                  else
                    if ((ox2-step)<ox1)
                       putchar(BELL);
                    else
                       {
                         incx1 = 0;     incy1 = 0;
                         incx2 = -step; incy2 = 0;
                       }
                  break;
    case RARROW : if (fixcoords == OFF)
                    if ((ox2+step)>511)
                       putchar(BELL);
                    else
                      {
                        incx1 = step; incy1 = 0;
                        incx2 = step; incy2 = 0;
                      }
                  else
                    if ((ox2+step)>511)
                       putch(BELL);
                    else
                       {
                         incx1 = 0;    incy1 = 0;
                         incx2 = step; incy2 = 0;
                       }
                  break;
    case PLACE  : if (fixcoords==OFF) fixcoords = ON;
```



```
                  else fixcoords = OFF;
                  break;
    case STEP   : if (increment == LARGE)
                    {
                      step = 1;                          /* Fine step */
                      increment = SMALL;
                    }
                  else
                    {
                      step = 20;                        /* Coarse step */
                      increment = LARGE;
                    }
                  break;
    default     : putchar(BELL);
   }

  opmode(MODE0,framebuffer);                          /* Set I/O mode */
  roww(ox1,oy1,ox2-ox1+1,rowbuff1);                 /* Erase rectangle */
  roww(ox1,oy2,ox2-ox1+1,rowbuff2);
  colw(ox1,oy1,oy2-oy1+1,colbuff1);
  colw(ox2,oy1,oy2-oy1+1,colbuff2);

  nx1 = ox1 + incx1; ny1 = oy1 + incy1;
  nx2 = ox2 + incx2; ny2 = oy2 + incy2;
  ox1 = nx1; oy1 = ny1;
  ox2 = nx2; oy2 = ny2;
  _clearscreen(_GCLEARSCREEN);
  printf("(%d,%d);(%d,%d)",nx1,ny1,nx2,ny2);

 } while (exit != TRUE);

 window->x1 = nx1;                          /* Save aoi coordinates */
 window->y1 = ny1;
 window->x2 = nx2;
 window->y2 = ny2;
 return(returnvalue);
}

int scancode(void)
/* AIM    : Function to return the scan code of a pressed key
 * INPUTS : None
 * OUTPUTS: Scan code returned in AH register
 */
{
   union REGS ireg;

   ireg.x.ax = 0x00;                                      /* Funtion 00h */
   int86(0x16,&ireg,&ireg);         /* BIOS keyboard service routine 16h.*/
   return(ireg.h.ah);              /* Scan code contained in AH register */

}

/*****************************************************************************/
/*                          ---- I/O ROUTINES ----                            */
/*****************************************************************************/

void colograb(void)
/* AIM: to grab a colour image which has the following cables attached
 *       to the MVPAT:
 *       (0)RED
```

```
 *       (1)GREEN
 *       (2)BLUE
 *       (3)SYNC: composite signal
 *      pressing any key will grab the image
 * INPUTS: any key to grab the image
 */
{
  opmode(2,6);
/*  disformat(1,1,0);   */   /* sets monitor to 12.5Mhz,interlaced,European*/
  sync(0,0); sync (1,1);  /* present MVPAT is difficult to sync */
  sync(0,0); sync (1,1);
  inmode (2);
  opmode (3,6);
  chan (2);
  outpath (6, -1, 0, 0);
  video (1, 1);
  cstart (0);
  cgrab (-1);
  printf("\nPress any key to grab image...\n");
  getch();
  cgrab(0);
/*  opmode(2,6); */
/*  disformat(0,1,1);*/ /* reset monitor back to 10Mhz,interlaced,American*/
}


void loadfile(void)
/* AIM : Loads a file into the given frame buffer of the MVP-AT
 * INPUTS: user specified framebuffer, and file name.
 */
{
  int framebuffer;
  char filename[50];
  char *fnptr;                                    /* Pointer to filename */

  _clearscreen(_GCLEARSCREEN);
  printf("Enter frame buffer for file (0,1,2,3 or 6): ");
  scanf("%d",&framebuffer);
  fnptr = &filename[0];
  printf("\nEnter file to be displayed : ");
  fflush(stdin);
  gets(fnptr);

  outpath(framebuffer,-1,0,0);
  opmode(0,framebuffer);                           /* Select I/O mode */
  if (fromdisk(fnptr)!=1)
    {
      printf("Error reading %s\n",fnptr);
      printf("Press a key to return to menu...");
      getch();
    }
  video(1,1);                 /* Enable display and view buffer contents */
}

void savefile(void)
/* AIM : Saves the contents of the required frame buffer to a disk file.
 * INPUTS: user specified framebuffer and file name.
 */
{
  int framebuffer;
```



```
  char filename[50];
  char *fnptr;                                    /* Ponter to filename */

  _clearscreen(_GCLEARSCREEN);
  printf("Enter frame buffer for file (0,1,2,3 or 6): ");
  scanf("%d",&framebuffer);

  fnptr = &filename[0];
  printf("Save contents of frame buffer %d to : ",framebuffer);
  fflush(stdin);
  gets(fnptr);

  opmode(0,framebuffer);                           /* Select I/O mode */
  todisk(fnptr);
}
```

```
/*****************************************************************************
 * MODULE: APPGRAPH
 *
 * AUTHOR: G.Kay
 *
 * DESCRIPTION: Draws graphs of R,G,B,H,S,I to the IBM compatible screen
 *
 * CONTAINS:
 *          appgraph
 *****************************************************************************/

#include <malloc.h>
#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <conio.h>
#include <string.h>
#include <graph.h>
#include <pgchart.h>
#include <stdlib.h>
#include <time.h>

/*****************************FUNCTION CODE*********************************/

void appgraph(int ydata[],int xdata[],int length,char *ylabel,char *xlabel,char *
title)
/* AIM: Draw gpaphs (or scatter plots) using MS chart routines of array of
 *      ydata (value of the line in R,G,B,H,S, or I) for 'length' number of
 *      points.
 * INPUTS: ydata - array of points for the y axis
 *         xdata - array of points for the x axis
 *         length- the number of points used in the above arrays.
 *         ylabel- yaxis label (eg.Red,Green,Blue,Hue,Saturation,Intensity labels

 *         xlabel- x axis label.
 *         title - label of graph
 * OUTPUTS: graph to the sreen.
 */
{
    chartenv env;
    float far *newy;    /* to convert int arrays to float arrays for*/
    float far *newx;    /* plotting with MS routines                */
    int i;              /* index to arrays      */
    int returnval=0;    /* 0=okay 1,2=out of memory on float conversion */

    if( !_setvideomode( _MAXRESMODE ) )
    {   printf("\nERROR: Unable to initialize graphics mode");
        return;
    }

    if((newy = (float far*)_fmalloc(length*sizeof(float far)))==NULL) returnval=1
;
    if((newx = (float far*)_fmalloc(length*sizeof(float far)))==NULL) returnval=2
;
    printf("\n\nConverting H,S and I planes in the mask to floats...");
   if (returnval!=0)
   { printf("\nERROR: Out of memory on this size area_of_interest");
     switch(returnval)
     {
      case 1: break;
```



```
      case 2: free(newy); break;
     }
     getch();
   }
   else
   {

    for (i=0; i<length ; i++)
    {
        newy[i] = (float )ydata[i+1]; /*add 1 to i to eliminate ydata[0]*/
        newx[i] = (float )xdata[i+1]; /*xdata start at 1 and end at 'length - 1'*
/
    }

    printf("\n\n About to draw scatterplot...");
    _pg_initchart();                    /* Initialize chart system    */


    /* Single-series scatter chart */
    _pg_defaultchart( &env, _PG_SCATTERCHART, _PG_POINTONLY );

    strcpy(env.xaxis.axistitle.title,xlabel);
    strcpy(env.yaxis.axistitle.title,ylabel);
    strcpy( env.maintitle.title, title );

    if(_pg_chartscatter( &env, newx, newy, length ))
    {
      _setvideomode( _DEFAULTMODE );
      printf("\n\nERROR: Unable to draw scatter plot with so many values");
      getch();
    }
    else
    {
     getch();
     _clearscreen( _GCLEARSCREEN );
     _setvideomode( _DEFAULTMODE );
    }
    free(newy);
    free(newx);
   }
}
```

```
/*************************************************************************
 * MODULE: NRUTIL
 *
 * AUTHORS: Vetterling W.T., Teukolsky S.A., Press W.H., Flannery B.P.,
 *          from 'Numerical Recipes in C: The Art of Scientific Computing'
 *          (1988), Cambridge University Press.
 *          G.Kay.
 *
 * DESCRIPTION: Utilities for memory allocation and statistical calculations.
 *              This module contains selected numerical recipes for C.
 *              A function for 2 dimensional statistics is also given.
 *
 * CONTAINS:
 *          nrerror - error message display function
 *          imatrix - function to allocate memory for an integer matrix
 *          free_imatrix - frees the memory of the matrix allocated.
 *          moments - general statistics for a single array of data
 *          stats2_d- Two dimensional stats for H and I features
 *************************************************************************/
#include <malloc.h>
#include <stdio.h>
#include <math.h>

/*****************************FUNCTION CODE*******************************/
void nrerror(error_text)
char error_text[];
{
        void exit();

        fprintf(stderr,"Numerical Recipes run-time error...\n");
        fprintf(stderr,"%s\n",error_text);
        fprintf(stderr,"...now exiting to system...\n");
        exit(1);
}


int **imatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
        int i,**m;

        m=(int **)malloc((unsigned) (nrh-nrl+1)*sizeof(int*));
        if (!m) nrerror("allocation failure 1 in imatrix()");
        m -= nrl;

        for(i=nrl;i<=nrh;i++) {
                m[i]=(int *)malloc((unsigned) (nch-ncl+1)*sizeof(int));
                if (!m[i]) nrerror("allocation failure 2 in imatrix()");
                m[i] -= ncl;
        }
        return m;
}



void free_imatrix(m,nrl,nrh,ncl,nch)
int **m;
int nrl,nrh,ncl,nch;
{
        int i;
```



```
        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}


void moment(data,n,ave,adev,sdev,svar,skew,curt)
int n;
int data[];
float *ave,*adev,*sdev,*svar,*skew,*curt;
{
        int j;
        float s,p;

        if (n <= 1) nrerror("n must be at least 2 in MOMENT");
        s=0.0;
        for (j=1;j<=n;j++) s += (float)(data[j]);
        *ave=s/n;
        *adev=(*svar)=(*skew)=(*curt)=0.0;
        for (j=1;j<=n;j++) {
                *adev += fabs(s=(float)(data[j])-(*ave));
                *svar += (p=s*s);
                *skew += (p *= s);
                *curt += (p *= s);
        }
        *adev /= n;
        *svar /= (n-1);
        *sdev=sqrt(*svar);
        if (*svar) {
                *skew /= (n*(*svar)*(*sdev));
                *curt=(*curt)/(n*(*svar)*(*svar))-3.0;
        } else nrerror("No skew/kurtosis when variance = 0 (in MOMENT)");
}

void stats2_d(int Hdata[],int Idata[],int n)
/* AIM: To provide means, variances, covariance matrix (determinant & inverse)
 * INPUT: - two data arrays Hdata, Idata, & there size n
 */
{
        int j;
        float Hsum,Isum,HIsum;
        float Have,Hsumsq;
        float Iave,Isumsq;
        float Hvar,Ivar,HIvar;
        float detHIcov;
        float HIcov[2][2],invHIcov[2][2];

        if (n <= 1) nrerror("n must be at least 2 in stats2_d");
        Hsum=(Isum)=(HIsum)=(Hsumsq)=(Isumsq)=0.0;
        for (j=1;j<=n;j++)
        {
          Hsum += (float)(Hdata[j]);
          Isum += (float)(Idata[j]);
          HIsum += (float)Hdata[j]*Idata[j];
          Hsumsq += (float)Hdata[j]*Hdata[j];
          Isumsq += (float)Idata[j]*Idata[j];
        }
        Have=Hsum/n;
        Iave=Isum/n;
        Hvar = (Hsumsq - n*Have*Have)/(n-1);
```

```
        Ivar = (Isumsq - n*Iave*Iave)/(n-1);
        HIvar= (HIsum  - n*Have*Iave)/(n-1);
        HIcov[0][0]=Hvar;
        HIcov[0][1]=(HIcov[1][0])=HIvar;
        HIcov[1][1]=Ivar;
        detHIcov = Hvar*Ivar - HIvar*HIvar;
        invHIcov[0][0] = Ivar/detHIcov;
        invHIcov[0][1] = (invHIcov[1][0]) = -HIvar/detHIcov;
        invHIcov[1][1] = Hvar/detHIcov;
        printf("\nHue mean      = %f",Have);
        printf("\nInt mean      = %f",Iave);
        printf("\nCovariance matrix = \n%f\t%f\n%f\t%f",Hvar,HIvar,HIvar,Ivar);
        printf("\nDeterminant   = %f",detHIcov);
        printf("\nInverse covariance matrix = \n%f\t%f\n%f\t%f",\
               invHIcov[0][0],invHIcov[0][1],invHIcov[1][0],invHIcov[1][1]);
}
```

```
/*****************************************************************************
 * HEADER: APPCOM.H
 *
 * DESCRIPTION: Header for modules using common #includes
 *****************************************************************************/
#include <dos.h>
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
#include <io.h>
#include <malloc.h>
#include <math.h>
#include <graph.h>
#include "im2to3.h"                 /* Prefix MVP-AT functions with 'im' */
#include "debugat.h"                        /* MVP-AT debug information */
```



```
/*****************************************************************************
 * HEADER: APPDEFN.C
 *
 * DESCRIPTION: Header for modules containing common #defines
 *****************************************************************************/

/* DEFINITIONS FOR KEY SCAN CODES */
#define ESC    0x01
#define RETURN 0x1C
#define UARROW 0x48
#define DARROW 0x50
#define LARROW 0x4B
#define RARROW 0x4D
#define PLACE  0x21
#define STEP   0x1F
#define BELL 7

#define TRUE 1
#define FALSE 0

#define row 512
#define col 512      /* length of row and column */

/* MVP-AT FRAME BUFFER DEFINITIONS */
#define FB0 0                                /* Frame buffer 0 - 512x512x8 */
#define FB1 1                                /* Frame buffer 1 - 512x512x8 */
#define FB2 2                                /* Frame buffer 2 - 512x512x8 */
#define FB3 3                                /* Frame buffer 3 - 512x512x8 */
#define FB4 4  /* Frame buffer 4 - 512x512x16 : combination of FB0 & FB1 */
#define FB5 5  /* Frame buffer 5 - 512x512x16 : combination of FB2 & FB3 */
#define FB6 6  /* Frame buffer 6 - 3x512x512x8: RGB=013, 2 is overlay    */
#define FB7 7  /* Frame buffer 7 - 1x1024x1024x8: working buffer         */

/* MVP-AT PROCESSING MODES */
#define MODE0 0                               /* I/O mode                */
#define MODE1 1                               /* Graphics mode           */
#define MODE2 2                               /* Processing mode         */
#define MODE3 3                               /* Continuous grabbing mode */
```



```
/*****************************************************************************
 * HEADER: APPTYPE.C
 *
 * DESCRIPTION: Header for modules containing common typedef statements
 *****************************************************************************/
/* TYPE DECLARATIONS */
typedef struct {              /* Processing window coordinates for MVP-AT */
         int x1;              /* (x1,y1) - top left corner                */
         int x2;              /* (x2,y2) - bottom right corner            */
         int y1;              /* NOTE : Parameter range - 0..511          */
         int y2;              /*        x1 & x2 must be multiples of 4    */
             } wincoords;

typedef struct{               /* statistics of the region of interest     */
         int maxHcount;           /* maximum pixel count for a hue value  */
         int maxH;                   /* hue value with max pixel count    */
         int maxScount;
         int maxS;
         int maxIcount;
         int maxI;
         long aoapple;        /* area of the inapple region               */
         long totaoi;         /* total area of the window being processed */
             } stats;

typedef unsigned char BYTE;

stats STATS;
```



```
/*****************************************************************************
 * HEADER: NRUTIL.H
 *
 * DESCRIPTION: The function prototype header for modules using the functions
 *              in module NRUTIL.H
 *****************************************************************************/

int **imatrix();
void free_imatrix();
void nrerror();
void moment();
void stats2_d();
```

APPENDIX B

The HICLUSMEAN.SAS Program Listing

The program HICLUSMEAN.SAS is used to take the feature files of HUE.DAT, SAT.DAT, and INTENSIT.DAT which are generated by the CAPP program. HICLUSMEAN when used in the SAS environment will generate a file containing the unsupervised class centroids which are determined by the processing function FASTCLUS. The program is capable of generating other items of interest, such as 2- and 3- dimensional graphic scatter plots of the pixels with there accociated CLUSTER value.

One should note that CLUSTER is the classification given to each pixel by the FASTCLUS routine. These CLUSTER centroids are in no particular order. When the CLUSTER centroids are written to a file CLUSMEAN.DAT then the centroids are entered with the centroid hue values placed in ascending order. This ordering of the centroids is referred to as CLASS numbers when this file is used in the CAPP cluster and classification function.

This program was written in the SAS statistical programming language on the U.C.T VAX. The following program listing shows an example which uses the H, S, and I features of the pixels associated with Granny Smith 7 (see Plate 2), generated by the CAPP program. In this example unsupervised clustering with the H and I features provides a file 'clusmean.dat' containing the CLASS centroid values used in Figure 6.2b. The example also shows how the H-I CLUSTER plot of Figure 6.3 was generated. This figure was a graphics screen dump of the H-I plot. The code listing is now shown.

```
/*************************************************************************************************************
   PROGRAM     : HICLUSMEAN.SAS

   AUTHOR      : G.Kay

   DATE        : 1991

   DESCRIPTION:  The purpose of this program is to provide the class centroids for the unsupervised K-means
                 classification of features associated with sampled colour pixels.

                 This program runs in the SAS environment on the UCT VAX. To start, type 'sas' at the $
                 prompt. The 386 PC terminal can be used as terminal type 'VT320' the graphics screen should
                 then be called TEK4010, alternatively the TEK4107 or TEK4207 graphics terminals should be
                 used.

                 At the SAS program Command prompt retrieve the file by typing 'include hiclusmean.sas'.
                 One can then edit the file names of the feature data to correspond to those features to
                 be classified. Code which is within comment marks (i.e. between */ and /*) can be activated
                 by deleting the comment marks. The number of clusters to be found can be specified by
                 editing the 'maxc' value in PROC FASTCLUS. The number of variables to be used in the
                 unsupervised classification can be be specified by entering the variable name in the 'var'
                 list of PROC FASTCLUS. These variables selected should be placed in the PROC SORT routine,
                 in the PUT cluster routine and in the PROC GPLOT routine. Typing 'submit' (or '<F3>' on
                 a VT320) at the command prompt will execute the program.
 *************************************************************************************************************/

/*--Select Global graphics options otherwise SAS default options will be used--*/

/*GOPTIONS reset=global gunit=pct border ftext=swissb htitle=6 htext=3;*/


/*-----------Option to print the graph produced to file plot1.GSF--------------*/

/* GOPTIONS device=tek4107 hpos=0 vpos=0 vsize=3.50 hsize=3.50
   gsfmode=replace
   rotate=portrait
   gsfname=plot1
   handshake=none;
 */


/*-----Obtain feature data from files and store them in the 'hbyi' data set----*/

DATA hbyi;
 FILENAME hue 'EEE:[KYXGAR01]hue.dat;9';
 INFILE hue;
  INPUT H;
 FILENAME intensit 'EEE:[KYXGAR01]intensit.dat;9';
 INFILE intensit;
  INPUT I;
 FILENAME sat 'EEE:[KYXGAR01]sat2.dat;9';
 INFILE sat;
  INPUT S;


/*-----------------Various Routines for the data read in----------------------*/

/*PROC FSVIEW DATA=hbyi;*/       /*--view, change or add variables--*/
/*PROC PRINT DATA=hbyi; */      /*--print data read to OUTPUT screen--*/
/*PROC MEANS DATA=hbyi; */       /*-- means, and stddev of data variables --*/
/*PROC PLOT DATA=hbyi;  *       /*-- ascii plot of the data --*/
/*  PLOT H * I ;        */
/*RUN;                  */


/*--FASTCLUS routine to do unsupervised clustering of 'var' data and put the --*/
/*--cluster centroids into DATA type clusmean---------------------------------*/

   PROC FASTCLUS DATA=hbyi out=hbyi maxc=4 mean=clusmean  noprint  ;
    var H I;
    title 'FASTCLUS Analysis';                  /*--labels for OUTPUT data--*/
    title2 'of colour pixels on Granny Smith 7';
   RUN;
```

```
/*------------- Reduce number of data points by removing duplicates -----------*/

  PROC SORT nodup;
   by H I ;
  RUN;


/*----Print the centroid values to OUTPUT screen and to file 'clusmean.dat'----*/
  PROC PRINT data=clusmean;                  /*--Print cluster means to OUTPUT--*/
  DATA _NULL_;                               /*-- don't create a data set ------*/
     SET clusmean;
     FILE clusmean;
     PUT cluster 1-3 H 5-11 I 13-19; /*--Positions to put 'cluster' centroid in file*/
  RUN;


/*--2-D Graphic scatter plot of variables with shaped symbols for the cluster--*
 *--points-------------------------------------------------------------------*/

   PROC GPLOT DATA=hbyi;
    symbol1 v=star c=white;
    symbol2 v=square c=white;
    symbol3 v=x c=white;
    symbol4 v=plus c=white;
    PLOT H*I=cluster;
   RUN;


/*--Give the 'cluster' value assigned to each point a shape and colour for the-*
 *--3-D graphs, and put these new values into DATA set 'clusshap'--------------*/

  DATA clusshap;
   LENGTH SHAPEVAL $ 8 COLORVAL $ 8;        /*--cluster symbol representation--*/
   SET hbyi;                    /*--use existing data set and add shape variable--*/
    IF cluster=1 THEN DO SHAPEVAL='spade';   COLORVAL='blue';  END;
    IF cluster=2 THEN DO SHAPEVAL='club';    COLORVAL='green'; END;
    IF cluster=3 THEN DO SHAPEVAL='diamond'; COLORVAL='red';   END;
    IF cluster=4 THEN DO SHAPEVAL='heart';   COLORVAL='yellow'; END;
  RUN;


/*--3-D Graphic scatter plot of variables shaped in the clussap Data set-------*/

/* PROC G3D DATA=clussnap;
      scatter H*S=I/ shape=SHAPEVAL color=COLORVAL;
   RUN; */
```