COLOR SEGMENTATION AND CLASSIFICATION OF FOOD IMAGES

BY

LING CHEN

A Thesis

Submitted to the Faculty of Graduate Studies

in Partial Fulfillment of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

University of Manitoba

Winnipeg, Manitoba

August, 1994

ISBN 0-612-13023-1

Canada

COLOR SEGMENTATION AND CLASSIFICATION OF FOOD IMAGES

BY

LING CHEN

A Thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

To My Husband

Chenglu Wen

I hereby declare that I am the sole author of this thesis.

I authorize the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Manitoba to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Abstract

In nutrition studies, it is frequently important to obtain long term accurate estimates of food intake. A color photograph of the meal, taken just before consumption, is a convenient way of recording the information, provided that it can be correctly interpreted. The objective of this thesis was to study the possibility of identifying food items by color alone. Food plates which were photographed on 35mm color slides were digitized into a Macintosh IIcx computer. After pre–processing steps of compression, color model transformation, and image blurring, the algorithms of segmentation and classification were applied to the image. There were two steps of segmentation. The first was identification of the background and the plate by a thresholding method based on the analysis of histograms. Then an algorithm which combined region–growing with splitting–and–merging was applied to the remaining area. Once an image was segmented into several segments with above method, average hue and saturation of every segment were calculated and were compared to each element of a previously measured training set. A segment was assigned as a food item if it matches one element in the set. Alternatively, it was assigned as unknown if it did not match any element. About 30 slides were used as training data, and about 10 slides were used to test the algorithms and programs. The algorithms and programs were successful. The study shows that it is possible to identify food items on a color image.

Acknowledgements

I would like to thank Professor W. H. Lehn for his kindly guidance and support during the last two years. I would also like to thank Dr. G. P. Sevenhuysen, of the department of Foods and Nutrition, Human Ecology, who proposed and supported this research.

CONTENTS

Chapter 1 Introduction 1

1.1 Introduction 1
1.2 Color Food Image Processing 1
1.3 Color Models 2
1.3.1 RGB Model 2
1.3.2 HSI and HSB Model 4
1.3.3 Other Model 7
1.4 Data Format and Convention 7

Chapter 2 System Description 9

2.1 Image Digitizing 9
2.2 Image Compression 10
2.3 Color Model Transformation 10
2.4 Blurring 10
2.5 Identification of Background and Plate 11
2.6 Segmentation 11
2.7 Classification 12

Chapter 3 Pre–Processing 14

3.1 Transformation 14
3.2 Image Compression 22
3.3 Image Blurring 23

Chapter 4 Training Data and Color Variation 29

4.1 Training Data 29
4.2 Color Variation 31

Chapter 5 Segmentation 34

5.1 Segmentation 34

5.2 Color Segmentation–Survey of Related Research 38
5.3 Color Segmentation 40
5.3.1 Identification of Background and Plate 41
5.3.2 Segmentation of Food Items 42

Chapter 6 Classification 52

6.1 Classification 52
6.2 Color Classification–Survey of Related Research 55
6.3 Classification –Program and Result 56

Chapter 7 Conclusion and Discussion 61

References 63

Appendix A Description of Barneyscan 65

Appendix B Segmentation Results 67

Appendix C.1 Program COMP4.C 78

Appendix C.2 Program SEG–SAVE–CLASS.C 84

CHAPTER 1 INTRODUCTION

1.1 Introduction

Color has been playing a more and more important role in segmentation and classification of images. Part of its significance comes from the fact that color is the most obvious information and has a greater amount of detailed information which generally enables objects to be distinquished more easily. Thus, color segmentation and classification are reasonable procedures to be considered in food image processing. Segmentation and classification of color images by color differences are usually done by using a variety of color models (or spaces) and criteria for threshold selection. The RGB (red, green, blue) model and HSB (hue, saturation, brightness) and HSI (hue, saturation, intensity) models are the most common ones used for color segmentation and classification. Histograms of every component in each space could be helpful for determining the criteria for threshold selection.

1.2 Color Food Image Processing

Food image processing has been proposed by Dr. G. P. Sevenhuysen and some of his colleagues as a method for accurately estimating individual food intake over long periods [2],[3]. In determining the relationship of diet and chronic diseases, such as heart disease and cancer, it is necessary to measure individual food consumption. The methods most commonly used are the recall interview, the diet history, and the food record using volume estimates of food. But all these

methods could not avoid the disadvantage of the subjectivity involved in quantifying amounts eaten. Therefore, they proposed a photographic method to record food consumption to minimize errors due to subjective estimation of food quantity by either investigator or respondent. A photographic record is made by the subject, who takes a single photograph of each plate of food at the time of eating. By segmentation and classification of the images by color, shape, texture, mottling, and size, they would decide which kind of food the observed subject has taken, and by a certain method of calculation of the amount of each kind of food eaten, they achieve the goal of individual food consumption calculation.

This thesis concerns only the first step of the whole project: segmentation and classification by color, which is a further step towards the totally automatic processing of photographic records. The remaining work will be left for further study.

1.3 Color Models

A color model (or a color space) is a specification of a 3–D coordinate system and a subspace within that system where each color is represented by a single point.

1.3.1 RGB Model

The most commonly used color model both in hardware and in image processing is the RGB (red, green, blue) model. Owing to the structure of the human eye, all colors are seen as variable combinations of the three primary colors red, green, and blue. In the RGB model, each color appears in its primary spectral

components of red, green, and blue. It is based on a cartesian coordinate system. The color subspace of interest is the cube shown in Figure 1.1 [1].

For theoretical convenience, it is assumed that all color values have been normalized so that the cube in Figure 1.1 is the unit cube. For digital image processing, color values could have different maximum values depending on different systems. For example, for a 24–bit color system (which could have more than a million colors), each component (R, G, and B) could have a value from 0 to 255 (one byte).

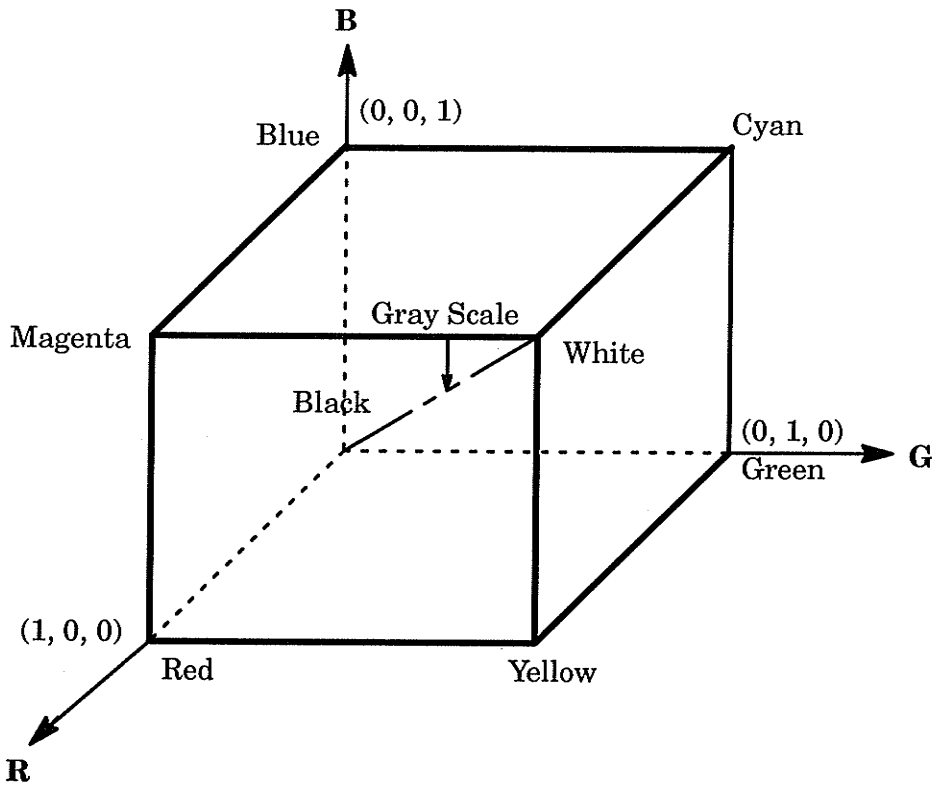


Figure 1.1 RGB Color Cube

Images in the RGB color model consist of three independent image planes, one for each primary color. When fed into an RGB monitor, these three images

combine on the phosphor screen to produce a composite color image. Thus the use of the RGB model for image processing makes sense when the images themselves are naturally expressed in terms of three color planes. Most color cameras used for acquiring digital images utilize the RGB format, which alone makes this an important model in image processing.

Sometimes the rgb model is used instead of the RGB model. Here,

$$r = \frac{R}{(R+G+B)} \tag{1–1}$$

$$g = \frac{G}{(R+G+B)} \tag{1–2}$$

$$b = \frac{B}{(R+G+B)} \tag{1–3}$$

1.3.2 HSI and HSB Models

Another color model frequently used in image processing is the HSI or HSB model. Here, hue is the color attribute of a color perception that describes a pure color (for example pure yellow, orange, or red). It is associated with the dominant wavelength in a mixture of light waves. Saturation gives a measure of the degree to which a pure color is diluted by white light, i.e. it refers to relative purity or the amount of white light mixed with a hue. I is the intensity. It is decoupled from the color information in the image. Brightness is the attribute of a visual sensation according to which a given visual stimulus appears to be more or less intense. The color components of the HSI model are defined with respect to the color triangle shown in Figure 1.2 [1].

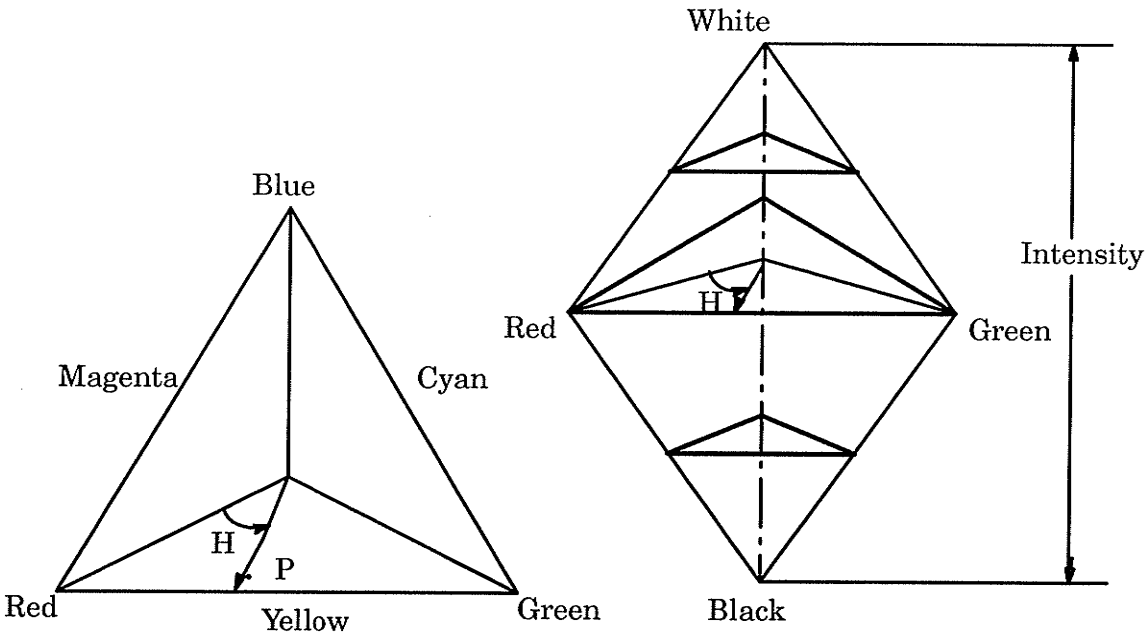


Figure 1.2 (a) HSI Color Triangle (b) HSI Color Solid

The two models could be converted from one to the other based on Figure 1.2. We have the expressions (1–4), (1–5), (1–6) [1], and (1–7) for the conversion from RGB to HSI and HSB models.

$$H = \cos^{-1} \frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-G)(G-B)]^{\frac{1}{2}}} \tag{1–4}$$

$$S = 1 - \frac{3}{(R+G+B)}[\min(R,G,B)] \tag{1–5}$$

$$I = \frac{1}{3}(R+G+B) \tag{1–6}$$

$$B = \max(R,G,B) \tag{1–7}$$

The conversion from HSB to RGB depends on the range of hue (0^0– 360^0). For the RG Sector($0^0 < H \leq 120^0$), we have the expressions (1–8), (1–9), and (1–10) [1].

$$b = \frac{1}{3}(1 - S) \tag{1–8}$$

$$r = \frac{1}{3}\left[1 + \frac{S\cos H}{\cos(60^0 - H)}\right] \tag{1–9}$$

$$g = 1 - (r + b) \tag{1–10}$$

For the GB Sector($120^0 < H \leq 240^0$), we have the expressions (1–11), (1–12), and (1–13) [1].

$$r = \frac{1}{3}(1 - S) \tag{1–11}$$

$$g = \frac{1}{3}\left[1 + \frac{S\cos H}{\cos(60^0 - H)}\right] \tag{1–12}$$

$$b = 1 - (r + g) \tag{1–13}$$

For the BR Sector($240^0 < H \leq 360^0$), we have the expressions (1–14), (1–15), and (1–16) [1].

$$g = \frac{1}{3}(1 - S) \tag{1–14}$$

$$b = \frac{1}{3}\left[1 + \frac{S\cos H}{\cos(60^0 - H)}\right] \tag{1–15}$$

$$r = 1 - (g + b) \tag{1–13}$$

1.3.3 Other Models

There are some other models in use today oriented either toward hardware (such as for color monitors and printers) or toward applications where color manipulation is a goal (such as in the creation of color graphics for animation). The CMY model (cyan, magenta, yellow) is the model for color printers; and the YIQ model (Y corresponds to luminance, and I and Q are two chromatic components called inphase, and quadrature, respectively) is the standard for color TV broadcast.

L*a*b* is also a color model often used for color image processing, which was developed to provide a computationally simple measure of color in agreement with the formerly popular Munsell color system. Here L* is correlated with brightness, a* with redness–greenness, and b* with yellowness–blueness. This system could also be converted to the RGB system, and vice versa.

1.4 Data Format and Convention

An RGB color image is comprised of three bitmaps, separated into red, green, and blue planes. They are produced by "Barneyscan" (a high resolution color digitizer combined with software–see Appendix A) in three separate scan passes

across the source image. In a color plane, a pixel value of 255 is the maximum brightness of that color primary, and 0 is the minimum.

An HSB image is an image made up of three channels: hue, saturation, and brightness. When an image is converted from an RGB image to an HSB image, brightness value of a pixel is the value of the largest of the R, G, or B color components.

The RGB or HSB image file format used in this system is non–interleaved raw format. In a non–interleaved raw file, data of the three component images are saved successively. For an RGB image, for example, there are three parts in a file. The first part is for red component. Values of red are saved pixel by pixel with one byte for one pixel. Once the red values of all pixels have been saved, the green values of all pixels start in the same way as the red values. Then the blue values follow. A HSB image could be saved in the same way. The only difference is that hue, saturation, and brightness values are located in three parts respectively, instead of red, green, and blue values.

CHAPTER 2 SYSTEM DESCRIPTION

2.1 Image Digitizing

The purpose of this project is to study the possibility of segmentation and classification of color images of food.

The photographic record of each food plate is made on a 35mm color transparency. In each record, there are usually background (grey), plate (white), and 2–5 food items on the plate. The transparencies are made under natural light, avoiding the influence of any artificial light on color. "Barneyscan" is a combination of a high resolution digitizer for 35 mm transparencies with software that drives the scanner and perform some basic image handling functions. It digitizes each transparency to a 24–bit RGB color image with the resolution of 1520 × 1024 pixels and stores it in Macintosh computer, requiring a storage of 4.5 megabytes for each image file. To save storage space (hard disk and memory) and processing time, unnecessary background is cut off, resulting in an image of 800 × 800 with 24 bits per pixel, a file size about 1.9 Megabytes.

The Barneyscan can adjust exposure time automatically when scanning so that the maximum values of R, G, or B in the image are 255. Since we use a white plate, which could be a reference for color adjustment, we set the exposure time for every slide so that the maximum R, G, and B values of the plate could be around 250. With this method, we avoid over–exposure, which might result in the loss of some color information. Theoretically, the R, G, B values of the white plate should be roughly equal. But due to the exposure system, the three values are slightly different sometimes. In this case, we can adjust the balance of color

for the whole image by "Barneyscan", so that the three values become roughly equal.

2.2 Image Compression

Because we are dealing with color information only and doing color segmentation and classification while neglecting details like texture or mottling, we compress the image by calculating the average R, G, B of four (2×2) connected pixels to be the new R, G, B values for one pixel in the new image (see also 3.2). Therefore the size of the new image is one quarter of the original size. The image ready for processing is a 400×400 24–bit color image with the size of 480 Kbytes.

2.3 Color Model Transformation

The images we get from the scanner are in RGB form. Values of R, G, and B are very sensitive to the brightness of images, i.e. they are very sensitive to the exposure time both for making slides and for scanning. Experimental results indicate that the HSB (hue, saturation, and brightness) model is relatively reliable. So we transform all RGB images into HSB images in this project (see also 3.1).

2.4 Blurring

When we only consider colors, too many details can make segmentation tedious. There are often sharp points in the image due to the high scanning resolu-

tion, noise, and the food itself. These pixels have very different colors than their neighborhoods. It is not expected to segment them as distinct foods. They are of no significance for color classification either. So we blur the image with a Gaussian blurring filter before it is segmented (see also 3.3).

2.5 Identification of Background and Plate

The next step is picking up the background and plate based on histogram analysis, since every image has almost the same background and plate (if we ignored the influence of different exposure on color). A set of thresholds in HSB space is built up to detect the background cluster and the plate cluster according to training results. Once the background and plate are determined, these two areas are not to be segmented.

2.6 Segmentation

The segmentation algorithm comes from the idea of combining a region growing algorithm and a region splitting–and–merging algorithm and is applied in the remaining areas after removal of background and plate. We mainly use hue and saturation as the variables to apply the segmentation algorithm with the help of brightness. Thresholds of differences of hue, saturation, and brightness of two succeeding pixels are set based on the analysis of training data. Values of hue, saturation, and brightness of each pixel are compared to the values of its neighbors. If the difference of every component between the pixel and one of its neighbors is smaller than the threshold, the pixel is joined to the neighbor. Alternatively, the pixel will serve as the start of a new region, if at least one of three

components shows a difference between the pixel and any neighbor larger than the threshold. This algorithm tests every pixel and results in an image (except the parts of the background and the plate) which is segmented into several parts according to their color vector values.

2.7 Classification

Now comes the last step to reach our goal: classification. The result from segmentation leaves us several areas which cover the whole image. Our algorithm of classification is suppose to recognize what they represent, e.g. how many kinds of food on the plate? And what are they?

The system calculates the average hue and saturation of each segment, so that each segment can have an average vector in HSB color space. This vector is compared to clusters of food items, the knowledge which we acquired from a number of training images. The decision is made when the location of the average vector match one of the clusters. If the vector does not match any cluster, the decision of "something else" would be given.

Figure 2.1 gives a flow chart of the system.

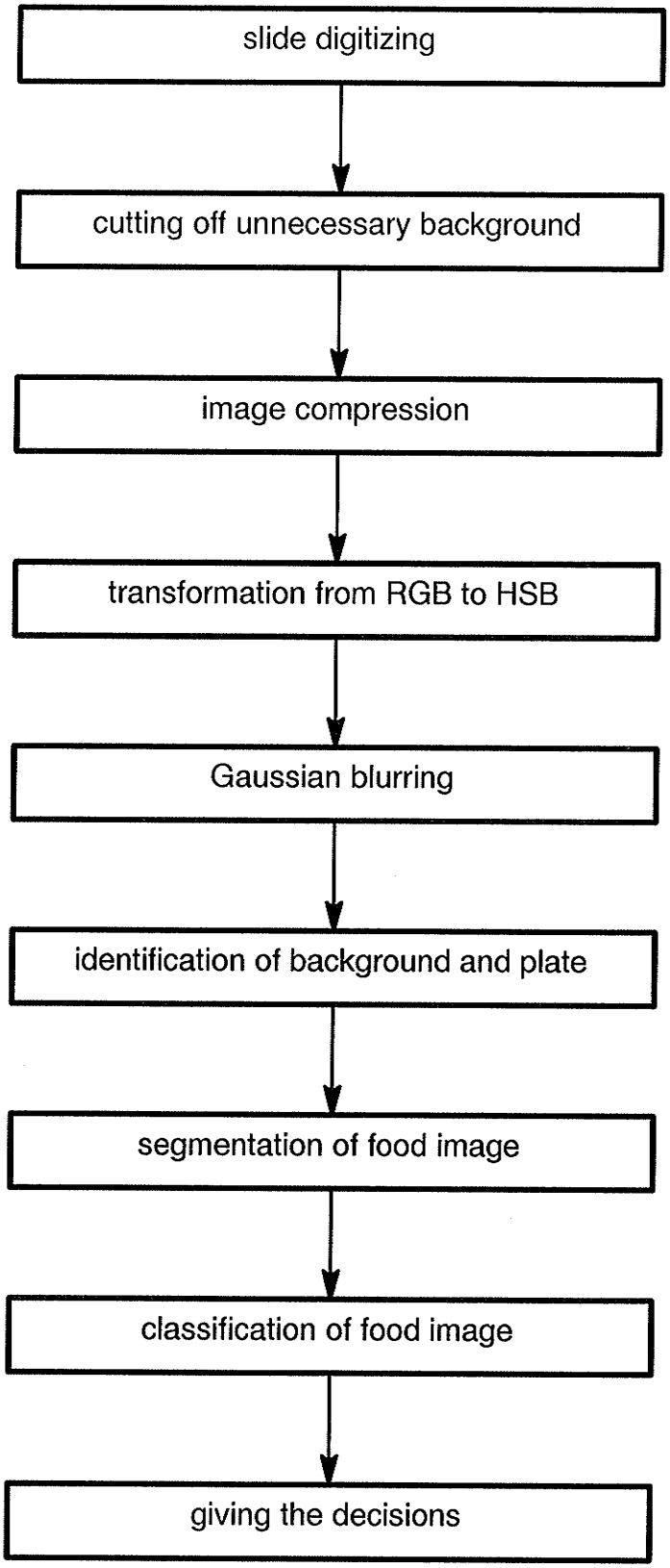


Figure 2.1 Flow Chart

CHAPTER 3 PRE-PROCESSING

3.1 Transformation

Original images digitized from 35mm slides by Barneyscan hardware and software are in RGB form. Three bytes (24 bits) are used to represent each pixel: one for red, one for green and one for blue. So R, G, B respectively could have any value from 0 to 255.

The most important components for segmentation, especially for classification, are hue and saturation, since hue is a color attribute of a color perception that describes a pure color and saturation gives a measure of the degree to which a pure color is diluted by white light. The higher the amount of white light mixed with a hue, the lower the saturation is. So hue combined with saturation could determine most colors. Brightness is the attribute of a visual sensation according to which a given visual stimulus appears to be more or less intense. In some particular case, when hue and saturation together are not enough to determine a color, we need the help of brightness. For example, the plate has the color white, and it is the brightest object in an image. But the hue of pure white is arbitrary. It could be any value with the saturation zero (Figure 1.2). In practice, plates in images are not pure white but close to pure white. Their hues are not reliable, they are sensitive to the light reflecting from surrounding objects. The values of saturation are very low (close to zero). In this case, hue is useless for detecting the plate. Further, we can not detect the plate according to values of saturation only, since other objects (food) might also have very low saturation, which would result in confusion. Brightness can play a role now. None of the

foods could have brightness as high as the plates. We can combine the values of saturation and brightness to detect plates.

Experimental results indicated that HSB color model is more effective than RGB color model in this system. Slides made at different times may have slightly different exposure. When they are digitized to computer, even with adjusted scanning exposure time, they will still have different RGB values for the same food. The reason for this is that R, G, B values are sensitive to the power of light, since the RGB values of a color corresponds directly to the absolute spectral radiant power distribution . Hue and saturation are less sensitive to this variation of color, since they mainly depend on the relative power distribution. The most sensitive component to the variation is brightness, which is not as important as hue and saturation in segmentation and classification; whereas the three components in the RGB model have the same importance.

Table 3.1 gives the experimental result which indicates the sensitivity of R, G, B, and H, S, B to the variation of color caused by different exposures. Here, all variables take the value from 0 to 255 (hue value is mapped from 0^0–360^0 into 0–255).

Checking the means and variances through Table 3.1, we found that for most food items the variances of hue and saturation are smaller than the variances of R, G, and B compared with their means, except the saturations of green–apple and broccoli which are not green at all the areas but yellow somewhere. The variances of brightness of most items are much higher than other variances compared to their means.

These results show that hue and saturation have smaller color variation. Based on these results the HSB color system was chosen. The transformation from RGB to HSB was implemented by Barneyscan. Figure 3.1 and Figure 3.2

are sample images of hue, saturation, and brightness components. Figure 3.1 comes from a unblurred image. Figure 3.2 comes from a Gaussian blurred image.

	$\overline{R}$	$\overline{G}$	$\overline{B}$	$\overline{H}$	$\overline{S}$	$\overline{B}$
tomato1	192	78	54	6	187	192
tomato2	175	72	44	8	195	175
tomato3	148	51	31	6	208	148
tomato4	159	68	51	6	178	159
tomato5	153	52	36	5	196	153
tomato6	209	90	60	7	183	209
tomato7	182	69	45	6	193	182
mean	174	68	45	6	191	174
variance	422	163	88	0	83	422

(a) Tomatoes

	$\overline{R}$	$\overline{G}$	$\overline{B}$	$\overline{H}$	$\overline{S}$	$\overline{B}$
carrot1	151	39	15	6	230	151
carrot2	173	51	18	7	228	173
carrot3	139	33	11	5	235	139
carrot4	141	34	10	6	236	141
carrot5	110	34	9	8	235	110
carrot6	140	35	11	6	236	140
carrot7	149	36	11	5	237	149
carrot8	137	30	10	5	237	137
mean	142	36	11	6	234	142
variance	269	36	8	1	10	269

(b) Carrots

	$\overline{R}$	$\overline{G}$	$\overline{B}$	$\overline{H}$	$\overline{S}$	$\overline{B}$
orange1	229	123	26	19	226	229
orange2	237	133	26	21	227	237
orange3	228	120	23	19	229	228
orange4	227	118	23	19	229	227

orange5	178	93	30	16	219	178
orange6	199	98	26	16	225	199
mean	216	114	25	18	225	216
variance	434	198	6	3	12	434

(c) Oranges

	$\overline{R}$	$\overline{G}$	$\overline{B}$	$\overline{H}$	$\overline{S}$	$\overline{B}$
apple1	153	147	45	41	183	154
apple2	187	173	54	39	184	187
apple3	150	149	50	43	176	156
apple4	130	126	35	41	191	132
apple5	137	137	70	42	143	141
apple6	150	151	73	43	148	155
mean	151	147	54	41	170	154
variance	323	206	179	2	342	291

(d) Green Apples

	$\overline{R}$	$\overline{G}$	$\overline{B}$	$\overline{H}$	$\overline{S}$	$\overline{B}$
broccoli1	13	20	9	69	142	20
broccoli2	31	41	18	65	157	42
broccoli3	19	26	9	62	178	26
broccoli4	15	23	6	61	202	23
broccoli5	16	23	7	60	190	24
broccoli6	11	16	3	58	212	16
broccoli7	19	32	7	64	201	32
broccoli8	18	25	9	65	180	26
broccoli9	17	26	8	64	179	26
mean	17	25	8	63	182	26
variance	29	46	14	9	442	49

(e) Broccoli

	$\overline{R}$	$\overline{G}$	$\overline{B}$	$\overline{H}$	$\overline{S}$	$\overline{B}$
peas1	11	22	6	69	169	22
peas2	17	31	10	68	176	31

peas3	9	18	4	67	193	18
peas4	10	21	5	67	194	21
peas5	9	18	4	65	204	18
peas6	8	16	3	64	217	16
peas7	18	26	8	59	195	27
peas8	10	20	4	64	207	20
peas9	10	21	5	69	198	21
peas10	9	19	5	66	193	19
mean	11	21	5	65	194	21
variance	10	17	4	8	176	18

(f) Green Peas

	$\overline{R}$	$\overline{G}$	$\overline{B}$	$\overline{H}$	$\overline{S}$	$\overline{B}$
potato1	192	176	144	28	72	192
potato2	225	210	180	27	51	225
potato3	234	218	178	31	60	234
potato4	236	218	172	31	70	236
potato5	240	228	187	35	57	240
potato6	235	215	176	28	65	235
potato7	170	160	136	33	57	170
potato8	244	228	179	33	67	244
potato9	220	207	169	32	60	220
potato10	218	202	167	30	60	218
mean	221	206	168	30	61	221
variance	494	438	240	6	38	494

(g) Potatoes

	$\overline{R}$	$\overline{G}$	$\overline{B}$	$\overline{H}$	$\overline{S}$	$\overline{B}$
corn1	180	116	26	24	219	180
corn2	194	126	27	24	221	194
corn3	191	122	27	24	220	191
corn4	175	107	22	22	225	175
corn5	140	95	24	26	216	140
corn6	164	101	20	23	227	164

corn7	192	125	26	24	223	192
corn8	182	114	24	23	223	182
mean	177	113	24	23	221	177
variance	285	113	5	1	11	285

(h) Corns

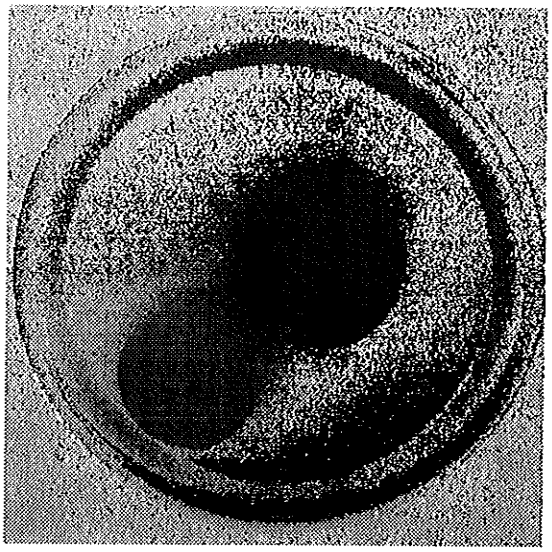
	$\overline{R}$	$\overline{G}$	$\overline{B}$	$\overline{H}$	$\overline{S}$	$\overline{B}$
beef1	48	29	19	14	169	48
beef2	32	16	11	16	170	32
beef3	23	11	7	20	178	23
beef4	21	9	5	15	190	21
beef5	24	10	6	16	188	24
beef6	18	7	4	16	199	18
beef7	32	16	11	21	174	32
beef8	25	12	8	20	180	25
mean	27	13	8	17	181	27
variance	79	42	21	6	97	79

(i) Beef

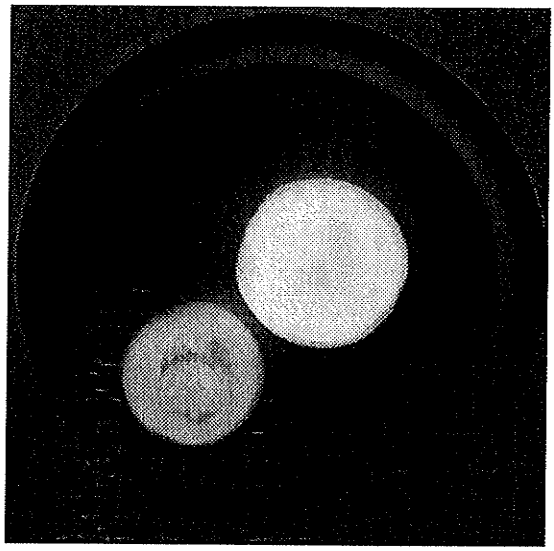
	$\overline{R}$	$\overline{G}$	$\overline{B}$	$\overline{H}$	$\overline{S}$	$\overline{B}$
toast1	224	171	98	24	145	224
toast2	175	122	65	21	163	175
toast3	235	181	108	24	139	235
toast4	152	116	61	25	159	152
toast5	158	108	51	22	177	158
toast6	174	131	70	25	152	174
toast7	229	176	103	24	145	229
mean	192	143	79	23	154	192
variance	1087	836	451	2	146	1087

(j) Toasts

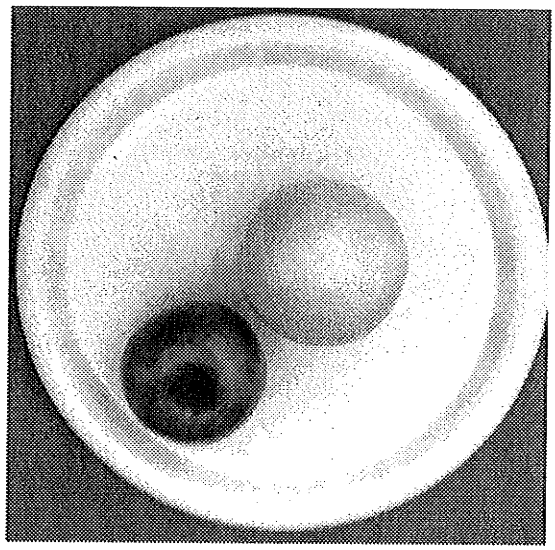
Table 3.1 Sensitivity of R,G,B,H,S, and B(bright) to the Variation of Color



(a) Hue

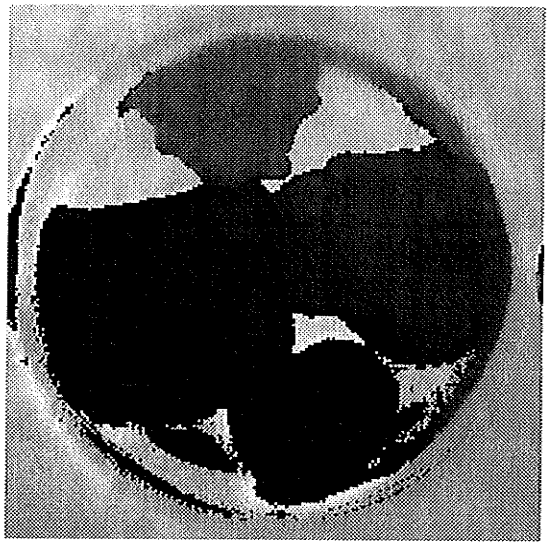


(b) Saturation

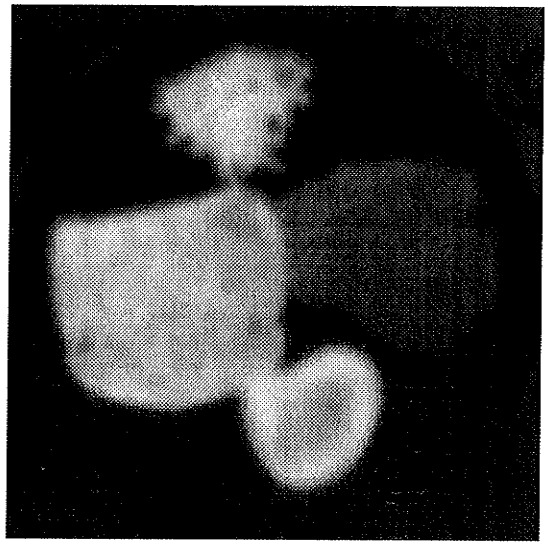


(c) Brightness

Figure 3.1 Images of Three Components of A Food Image



(a) Hue



(b) Saturation

(c) Brightness

Figure 3.2 Images of Three Components of A Blurred Food Image

3.2 Image Compression

An image, after digitizing by Barneyscan and cutting off unnecessary background, still requires about 1.9 megabytes of storage. It would take the computer (Macintosh cxII) too much time to fulfill the process of blurring, segmentation and classification (about 20 minutes). On the other hand, the same image with less resolution is sufficient for the purpose of color segmentation and classification. Therefore, the pre–processing of image compression is introduced into the system.

A simple compression algorithm is applied to the system. Average values of hue, saturation, and brightness (or red, green, and blue) of 4 (2 × 2) connected pixels in the original image are calculated to form the values of hue, saturation and brightness of one pixel in the new image.

If we have an original image with the horizontal size of hsize and the vertical size of vsize, the compressed image will have the new horizontal size of newhsize and the new vertical size of newvsize. Where:

$newhsize = hsize/2$ and

$newvsize = vsize/2$

Let hue[i × hsize+j], saturation[i × hsize+j], and brightness[i × hsize+j] represent the three component values at the position (i,j) in the original image, and let newhue[m × newhsize+k], newsaturation[m × newhsize+k], and newbrightness [m × newhsize+k] represent the values at the position (m,k) in the new image, where i, j, m, and k are integers and

$i = 2 \times m$ and

$j = 2 \times k$

Then

$$newhue[m \times newhsize + k] = \frac{1}{4}\{hue[i \times hsize + j] + hue[i \times hsize + j + 1] + hue[(i+1) \times hsize + j] + hue[(i+1) \times hsize + j + 1]\} \quad (3\text{–}1)$$

$$newsatu[m \times newhsize + k] = \frac{1}{4}\{satu[i \times hsize + j] + satu[i \times hsize + j + 1] + satu[(i+1) \times hsize + j] + satu[(i+1) \times hsize + j + 1]\} \quad (3\text{–}2)$$

$$newbright[m \times newhsize + k] = \frac{1}{4}\{bright[i \times hsize + j] + bright[i \times hsize + j + 1] + bright[(i+1) \times hsize + j] + bright[(i+1) \times hsize + j + 1]\} \quad (3\text{–}3)$$

Theoretically, this method of compression only reduces the high frequency details which are not important for color segmentation and classification, while leaving low frequency information unchanged. This process is experimentally feasible. It would not result in significant image change for the purpose of segmentation and classification, but the storage (both in hard disk and in memory) is reduced from 1.9 Megabytes for the original image to 480 Kbytes for the new image. The time required for all processing (blurring, segmentation and classification) is reduced to less than 4 minutes.

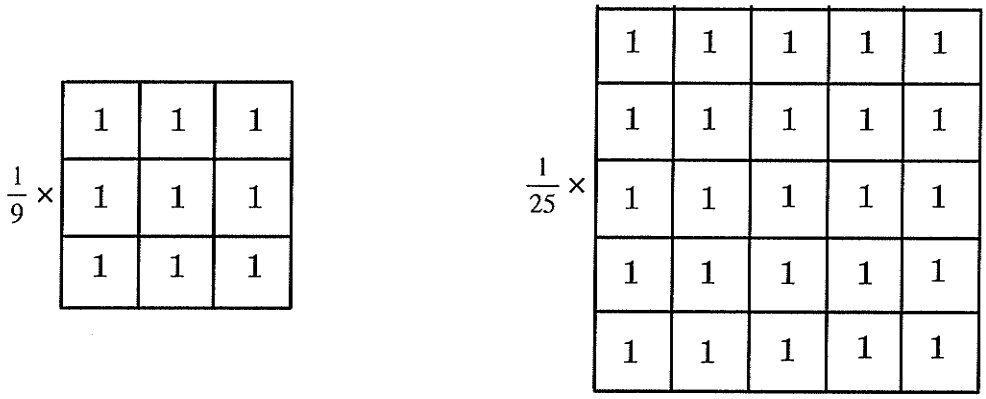
3.3 Image Blurring

Blurring (one of the smoothing filters) is usually used in pre–processing steps such as noise reduction, removal of small details from an image prior to (large) object extraction, and bridging of small gaps in lines or curves.

The objects in the images, which are expected to be segmented are relatively large; there are only 2–5 items of food in an image plus the background and the plate. Too many details could not be beneficial to the process of segmentation and classification and would take a long time to process. So blurring is implemented before segmentation. Another reason to choose blurring is that there are some noise points in images. These arise from the food itself (e.g. shadow areas). However some of the noise appears to arise from film grain resolved by the high resolution scanner (1520 × 1024 pixels, not adjustable). A smoothing filter is important to reduce the effect of these noisy pixels on segmentation.

There are several types of blurring filters. Frequency domain methods were not considered, since the FFT and IFFT transformations should be applied for using these filters. This would make the system more complicated and more time–consuming. Among spatial blurring filters, the simplest one is the lowpass spatial filter, which attenuates or eliminates high–frequency components in the Fourier domain (e.g. details and noise in the spatial domain) while leaving low frequencies (basic information of images) untouched.

In the spatial domain, a convolution of the image with the filter function is carried out. Because of the symmetry of most functions, building a filter is building a mask. A 3×3 mask (smallest) is built as in Figure 3.3(a).



(a) 3x3 mask (b) 5x5 mask

Figure 3.3 Spatial Lowpass Filters of Various Sizes

In order to perform 3×3 mask blurring, the central point of the mask is moved throughout the image pixel by pixel. Once the central point of the mask covers a pixel in an image, the values of H, S, B of the nine pixels (the pixel in the center and its 8 neighbors) are averaged. The average values are used to replace the old values of H, S, B of this Pixel.

To build the blurring filter for the system, the 3×3 mask and the 5×5 mask (Figure 3.3(b)) have been tested. A 5×5 mask can blur the image more strongly, but none of them produce satisfactory results for segmentation. Hence we considered Gaussian blurring as an alternative.

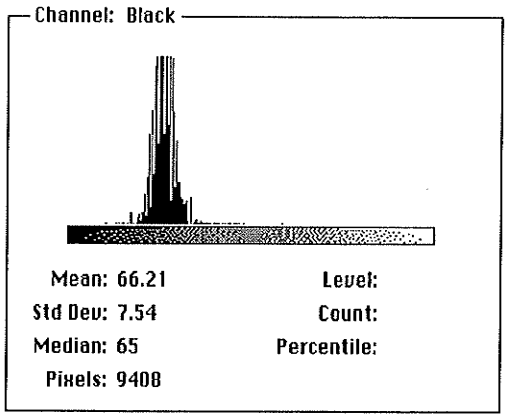
A Gaussian blurring filter is a mask with a kernel of Gaussian shape. The kernel for a 2–dimension of Gaussian smoothing operator in (x,y) coordinates is (where x, y = 0, 1, 2, ...):

$$G(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x^2+y^2)}{2\sigma^2}\right) \qquad (3\text{–}4)$$

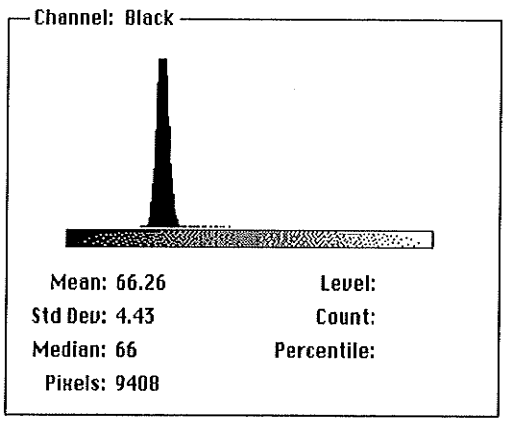
G(x,y) is circularly symmetric, so we can build a mask according to it. The smoothing effect may be controlled through σ . In this system, σ is chosen as 2. So the kernel can be expressed as (3–5):

$$G(x,y) = \frac{1}{8\pi}\exp\left(-\frac{(x^2+y^2)}{8}\right) \tag{3–5}$$

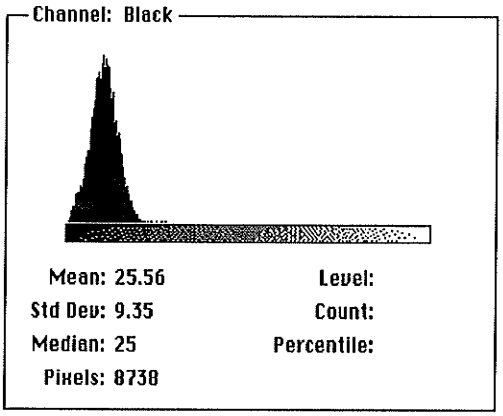
where, the x and y represent the relative coordinates from the center of the mask. An 11×11 mask was built. The mask is moved from the left–top corner to the right–bottom throughout the image pixel by pixel. When the center of the mask is located at a point, all hue, saturation, and brightness values of every pixel covered by the mask are multiplied by the coefficients and added up. This job was done by "Barneyscan". Figure 3.4 shows an example of histograms comparing hue, saturation, and brightness of an unblurred image and a blurred image. We can see from the histograms that the histogram of the blurred images have narrower distributions and sharper peaks. Statistical data shows that the blurred images have much smaller standard deviations. This means that the blurred images are much smoother in color. It is easier to segment blurred images. Although the edges of objects are also blurred, it is still acceptable for segmentation purposes. Therefore, Gaussian blurring was adopted.



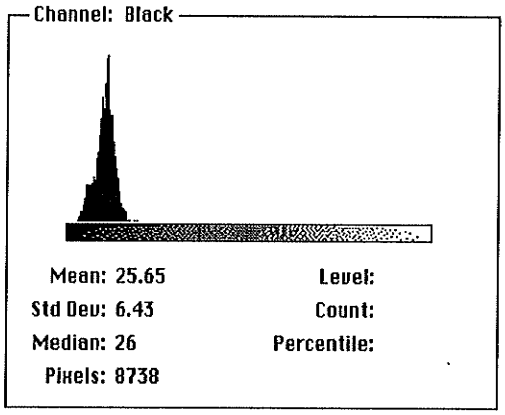
(a) Hue of Broccoli



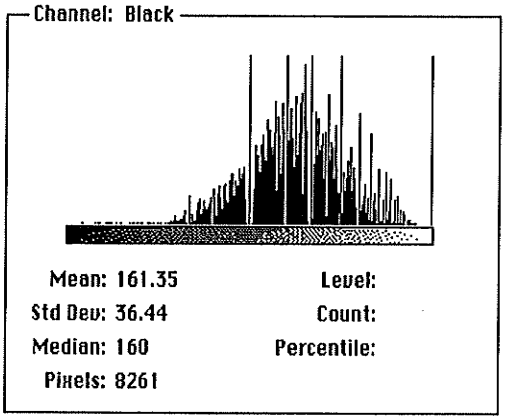
(b) Hue of Blurred Broccoli



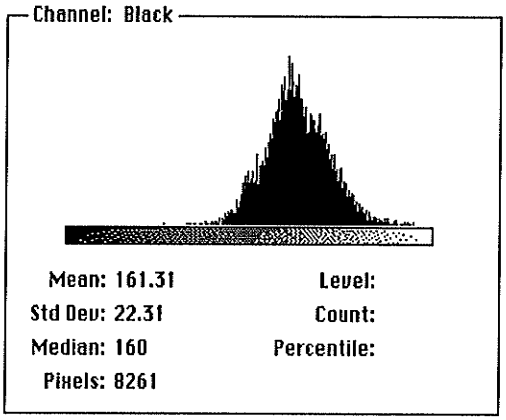
(c) Saturation of Broccoli



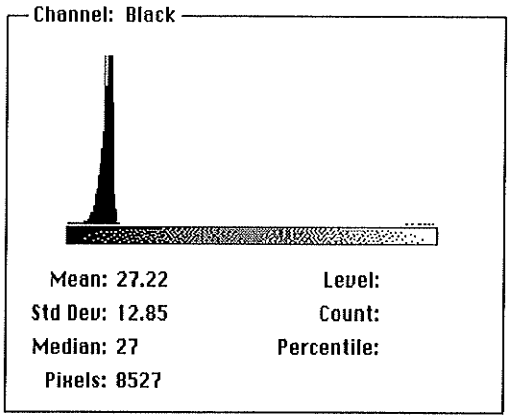
(d) Saturation of Blurred Broccoli



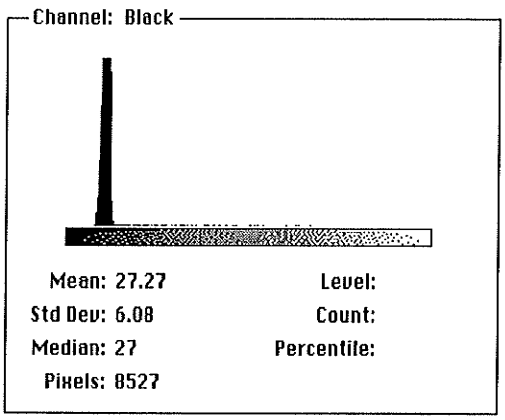
(e) Brightness of Broccoli



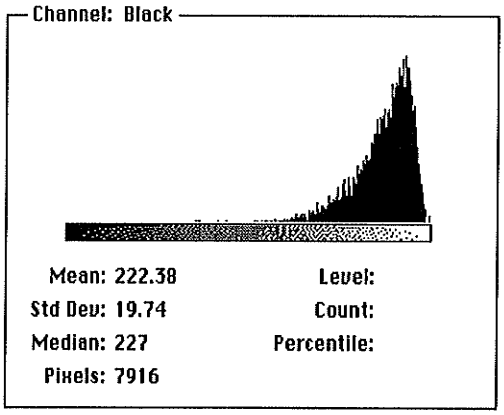
(f) Brightness of Blurred Broccoli



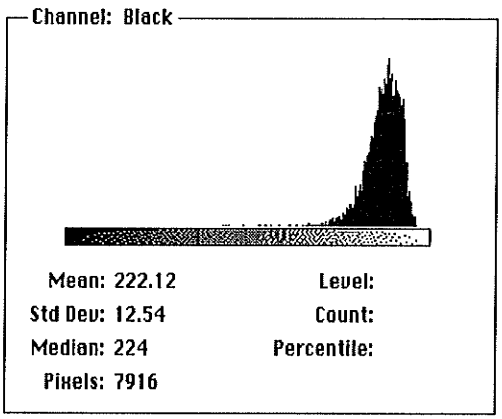
(g) Hue of Corn



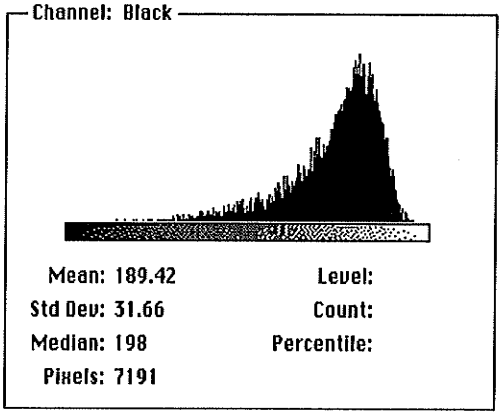
(h) Hue of Blurred Corn



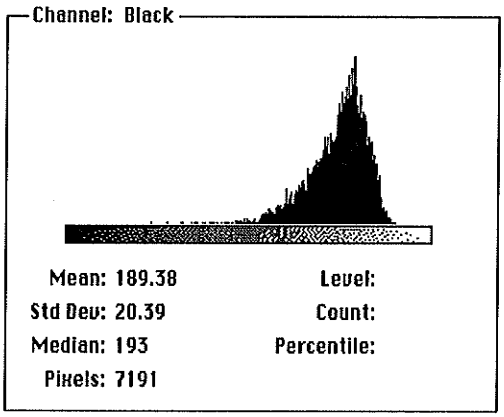
(i) Saturation of Corn



(j) Saturation of Blurred Corn



(k) Brightness of Corn



(l) Brightness of Blurred Corn

Figure 3.4 Comparison of Histograms

CHAPTER 4

TRAINING DATA AND COLOR VARIATION

4.1 Training Data

Color segmentation and, especially, classification of food images rely on the results of training experiments. A number of color images of food have been analysed to get the histograms, means, standard deviations and medians of hue, saturation, and brightness of different items of foods.

The means of histograms of hue, saturation and brightness (e.g. average hue, saturation, and brightness) of each item of food from sample images form a cluster in HSB space. These clusters can be used for the color classification. And the variances and medians can be helpful for setting the thresholds of segmentation.

We have used 30 slides of plates for training. There are 2–5 items in every slide. For every food item in every slide, we calculate its histograms, means, standard deviations, and medians of hue, saturation, and brightness.

Figure 4.1 shows the distribution of means of hue and saturation. Every point represents a food item from a plate. There are 20 points for the tomato, 12 for the carrot, 11 for the orange, 13 for the peas, 18 for the broccoli, 11 for the green apple, 10 for the corn, 22 for the potato, 18 for the beef, 19 for the toast. Some points overlap. Figure 4.1 shows that every item forms a cluster in the hue–saturation plane. We set classification rules later on from this result.

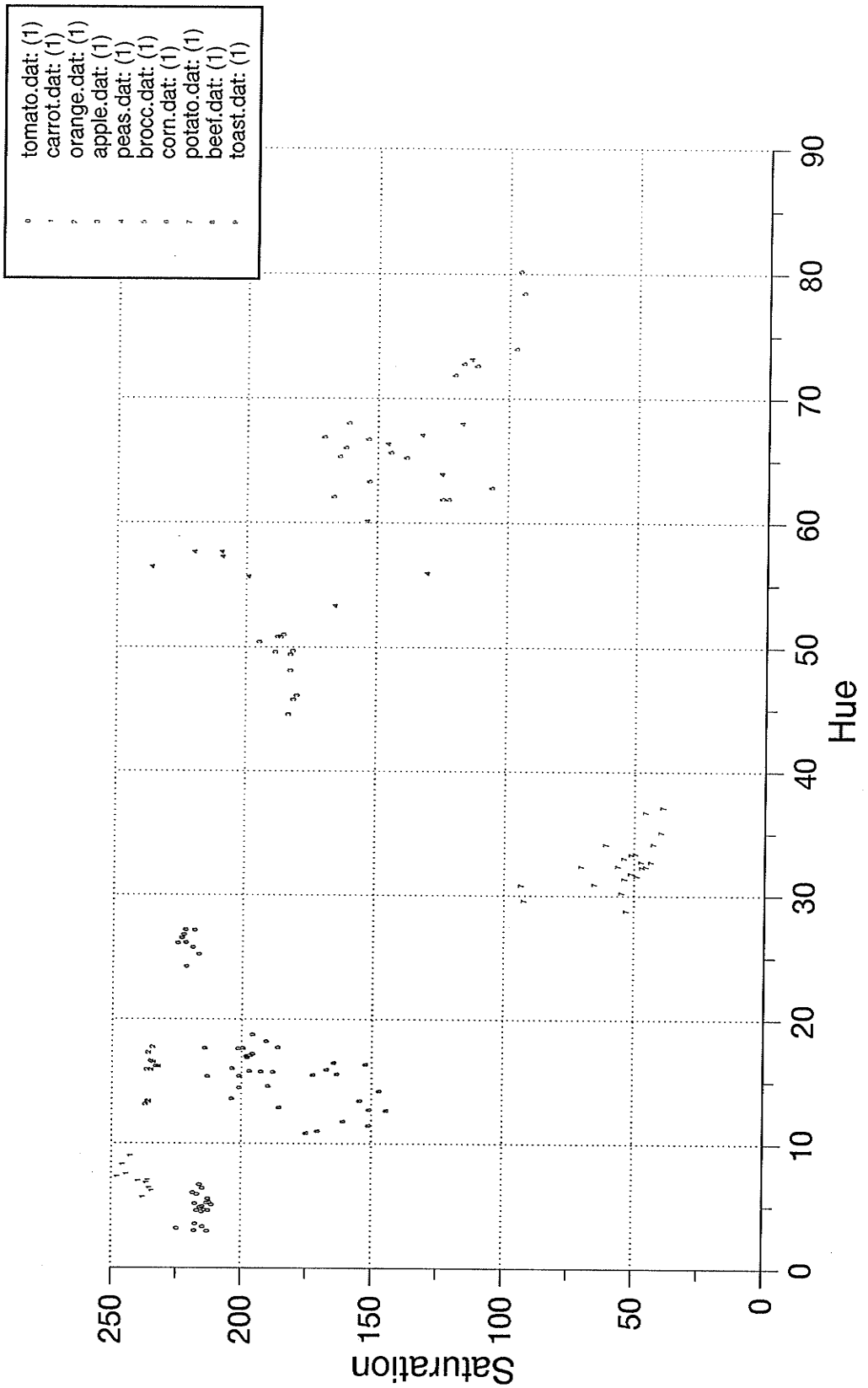


Figure 4.1 Distribution of Means of Hue and Saturation

4.2 Color Variation

Due to different exposure times both in making a slide and in scanning with Barneyscan, the same object could appear in slightly different color. Some tests are made to examine these color variations. The tests are made on a standard color chart, and the colors for testing are typical saturated colors: red, yellow, green, blue. Table 4.1 shows average hue, saturation, and brightness values of different colors in a single slide with different exposure times when scanning with Barneyscan. For this slide, the automatic (suitable) exposure time of Barneyscan is 376 (relative time).

exposure time	150	200	300	400	500
red	8	4	4	0	0
yellow	32	32	33	36	42
green	77	77	78	79	80
blue	167	167	166	166	166

(a) Average Values of Hue

exposure time	150	200	300	400	500
red	251	251	248	247	247
yellow	239	239	236	234	231
green	214	216	211	207	204
blue	232	240	236	238	239

(b) Average Values of Saturation

exposure time	150	200	300	400	500
red	50	54	71	103	139
yellow	107	116	152	219	255

green	28	30	41	64	86
blue	14	15	21	33	46

(c) Average Values of Brightness

Table 4.1 Values of Color Chart for Different Exposure Time of Scanning

We could see from Table 4.1 that hue and saturation values are slightly different but do not change much for most colors (except red), and the most sensitive variable is brightness.

Table 4.2 shows the color variation caused by different exposure when a slide is made. In Table 4.2, the object is the same thing (color chart). But one slide is slightly darker (the automatic exposure time of Barneyscan is 500), and the other is slightly brighter (the automatic exposure time of Barneyscan is 378). The average values of hue and saturation are slightly different.

	magenta	yellow	green	blue
bright slide	250	33	74	154
dark slide	253	27	78	158

(a) Average Values of Hue

	magenta	yellow	green	blue
bright slide	225	234	220	235
dark slide	232	240	230	248

(b) Average Values of Saturation

Table4.2 Values of Color Chart From Slides With Different Exposure Time

Table 4.1 and Table 4.2 show a bigger color variation caused by different exposure time when making a slide than caused by a different exposure time when scanning from a slide to the computer. Even with the adjustment of scanning exposure time, the variation is still visible.

Because the colors for this test are typical saturated colors and the slides are made under very good conditions, the variation is not significant. But for other colors and for slides which are not made under good conditions, especially with non–pure–white light background, color variation is not negligible.

CHAPTER 5 SEGMENTATION

Image Segmentation refers to the decomposition of a scene into its components. It is a key step in image analysis. Segmentation by color is the most important and the most difficult part of the project.

5.1 Segmentation

For image analysis, segmentation is generally the first step and autonomous segmentation is one of the most difficult tasks in image processing. Segmentation subdivides an image into its constituent parts or objects, and stops when the objects of interest in an application have been isolated; segmentation subdivides an image into regions that are uniform and homogeneous with respect to some characteristic. There is no complete theory of image segmentation although there are a variety of techniques for segmenting images. Segmentation techniques differ in the way they emphasize one or more properties and in the way they balance and compromise one desired property against another. The properties on which segmentation techniques are generally based are the properties of gray–level value (for monochrome images), texture, or some basic properties of color vectors in color spaces: RGB, HSB and so on (for color images). The two concepts of discontinuity and similarity of these values or vectors could be used to segment images. Using the discontinuity, we can partition an image based on abrupt changes of these values or vectors. This approach is mainly used for the detection of isolated points, lines and edges in an image. Using the similarity, the region based algorithms such as thresholding, region growing and region

splitting–and–merging are used to segment areas in an image. For region based techniques, we can describe the principle as follow. Let R represent the entire image region. We may view segmentation as a process that partitions R into n subregions: R_1, R_2, ... , R_n, such that [1]

$$(a)\ \bigcup_{i=0}^{i=n} R_i = R \tag{5–1}$$

$$(b)\ R_i \text{ is a connected region}, \quad i = 1, 2, ..., n, \tag{5–2}$$

$$(c)\ R_i \bigcap R_j = \phi \text{ for all } i \text{ and } j, \quad i \neq j, \tag{5–3}$$

$$(d)\ P(R_i) = TRUE \text{ for } i = 1, 2, ..., n, \text{ and} \tag{5–4}$$

$$(e)\ P(R_i \bigcup R_j) = FALSE \text{ for } i \neq j \tag{5–5}$$

where $P(R_i)$ is a logical predicate over the points in set R_i and ϕ is the null set. For monochromic images $P(r_i)$ could be $|G(r_k)–G(r_l)|$ <threshold,where $r_k, r_l \in R_i$, $G(r_k)$ is the grey level at the point r_k. For color images, the $P(R_i)$ could be that most color vectors in R_i are located in a certain area in color space.

In our project, area segmentation is the purpose. To apply thresholding approaches, the histogram of each component should be calculated, then thresholds in each histogram of every variable should be determined. Decisions are made based on these thresholds to find clusters in an image. These are effective approaches when there are only a couple of objects in an image, but cluster seeking becomes an increasingly complex task as the number of variables increases (for color images, there are at least 3 or 2 variables), especially when there are many objective clusters to be sought.

Region growing is a procedure that groups pixels or sub–regions into large regions. The simplest of these approaches is pixel aggregation, which starts with a set of "seed" points and from these grows regions by appending to each seed point those neighboring pixels that have similar properties (such as color). A simple example using this method is shown in Figure 5.1 [1]. Figure 5.1(a) is the

original image with gray levels represented in each point. Using two seeds (3,2) and (3,4) should result in a segmentation consisting of, at most, two regions: R_1 associated with seed (3,2) and R_2 associated with seed (3,4). The property P to be used to include a pixel in either region is that the absolute difference between the gray level of that pixel and the gray level of the seed be less than a threshold T. Figure 5.1(b) shows the result obtained using T=3, and Figure 5.1(c) shows the result when T=8. In Figure 5.1(c), since the threshold (T=8) of the absolute difference is higher than the difference of the two seed pixels, each region which grows from a seed includes the other seed and the all pixels. As a result, there is only one region.

	1	2	3	4	5
1	0	0	5	6	7
2	1	1	5	8	7
3	0	1	6	7	7
4	2	0	7	6	6
5	0	1	5	6	5

(a) Original Image

a	a	b	b	b
a	a	b	b	b
a	a	b	b	b
a	a	b	b	b
a	a	b	b	b

(b) Result With T=3

a	a	a	a	a
a	a	a	a	a
a	a	a	a	a
a	a	a	a	a
a	a	a	a	a

(c) Result With T=8

Figure 5.1 Example of Region Growing Using Known Seed Points

Using above approaches, choosing seed pixels is important, and it is not easy to automatically and appropriately choose seed pixels. In this case, region splitting–and–merging can be a practical method. The principle of these approaches is to subdivide an image into a set of arbitrary, disjointed regions and then merge and /or split the regions in an attempt to satisfy a pre–set criterion condition. It begins with the entire image as the initial segment. Then it successively splits each of its current segments into quarters if the segment is not homogeneous enough as judged by the criterion. Figure 5.2 [1] can simply describe the splitting–and–merging algorithm for a simple monochrome image.

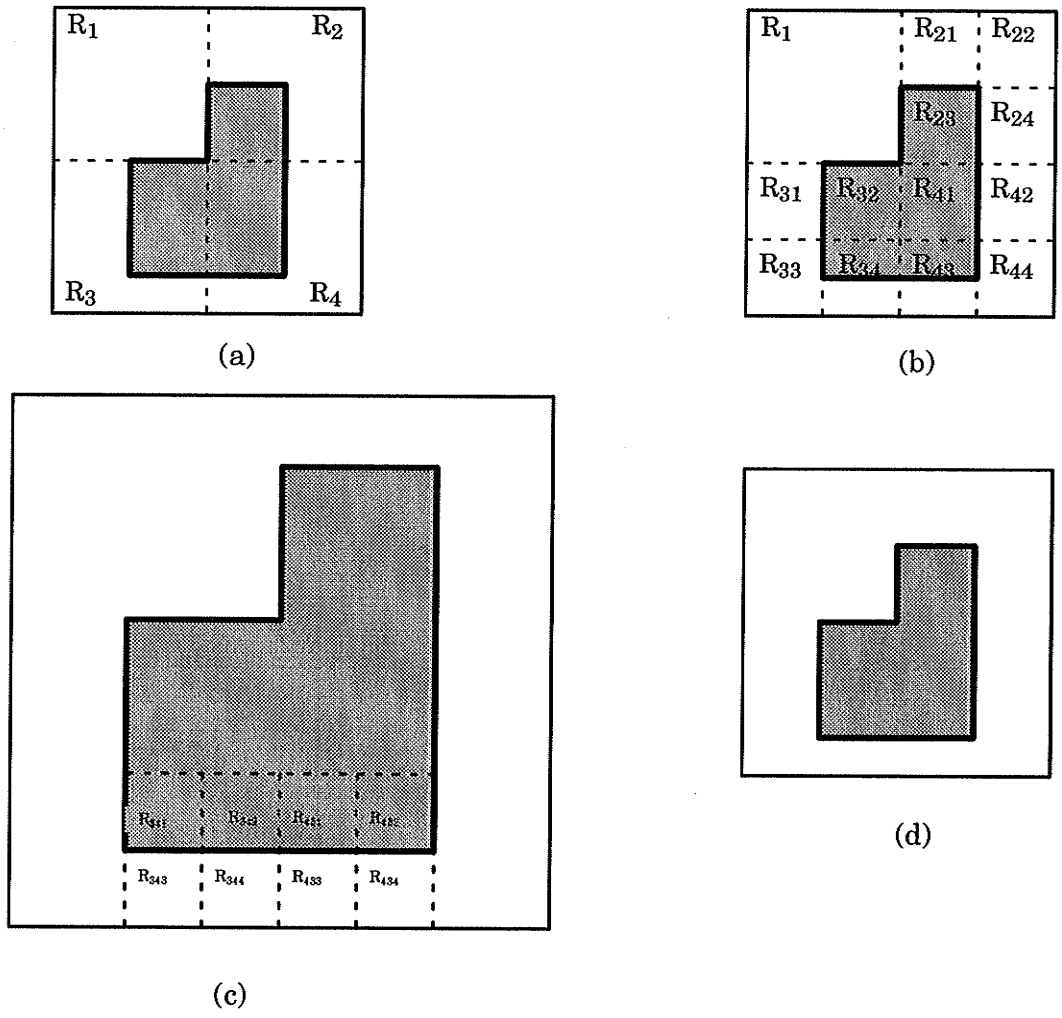


Figure 5.2 Example of Splitting–and–Merging Algorithm

(a). The image is split into R_1, R_2, R_3, R_4;

(b). R_2, R_3, and R_4 are split respectively into R_{21}, R_{22}, R_{23}, R_{24}, R_{31}, R_{32}, R_{33}, R_{34}, R_{41}, R_{42}, R_{43}, R_{44}. Merge R_{21}, R_{22}, R_{24}, R_{42}, R_{44}, R_{31}, and R_{33} to the background (R_1), and merge R_{23}, R_{32}, and R_{41} to the object;

(c). R_{34}, and R_{43} are split into R_{341}, R_{342}, R_{343}, R_{344}, R_{431}, R_{432}, R_{433}, R_{434};

(d). merge R_{341}, R_{342}, R_{431}, and R_{432} to the object, and merge R_{343}, R_{344}, R_{433}, and R_{434} to the background.

5.2 Color Segmentation--Survey of Related Research

The classical method for the segmentation of color images is to segment the individual component images separately, e.g. the red component image, the green component image, and the blue component image. The resultant segmentations are then combined to produce the complete color segmentation. The segmentation of individual component images is usually combined with histogram techniques. The most important aspect of this type of method lies in the techniques that are used to combine the results of component images. Rosenfeld and Kak described color segmentation schemes based on simple thresholds of the color vector components [4]. R. Weill and Y. Nes also applied RGB and rgb system to the separation of fruits and background with the help of intensity histograms [5].

There are some expansions of this type of method. R. B. Ohlander[6] used nine histograms, one for each component, in red, green, and blue(RGB), YIQ, and a perceptual model space based on hue, saturation, and intensity(HSI). He determined the most sharply defined feature, as measured by one of the nine parameters, then obtained a cluster of points that were uniform for the given feature,

applying thresholding on limits provided by the minimal bounding of the best peak. He extracted the region so isolated and eliminated it from further consideration. As a result of this, features that were formerly obscured may become more distinct. This procedure is applied iteratively until there is no prominent peak in any histogram. This approach results in regions that are approximately uniform in all nine components. Another expansion is made in other color spaces such as HSI [9],[10] or L*a*b* [7]. S. H. Ong and C. C. Hew segmented color images by iterative thresholding of hue, saturation and intensity components and reduced fragmentation by a merging procedure after each thresholding[9].

Baker, Hwang and Aggarwal [7] translated the RGB to the CIELAB (L*a*b*) transformation and combined the histograms of L (luminance), C (chroma), H (hue), and scattergram of the a*b* coordinates. This approach needs the transformation from RGB model to the CIELAB model and the calculation of every component's histogram. All schemes above are based on the proper combined thresholds of the component histograms.

Miyake, Saitohn, Yaguchi and Tsukada [8] used an experimental expression about r (r=R/(R+G+B)), g (g=G/(R+G+B)), b (b=B/(R+G+B)) to determine the region of skin color for TV pictures. This kind of method needs a large amount of experimentation for single color extraction to get the means and standard variance of each component.

When there are more than one objects appearing in an image, or if color variation for the same objects on different images exists, a better approach of segmentation might be splitting–and–merging. Trying to recognize a piece of ham on a conveyor belt, F. Diaz Pernas and J. Lopez Coronada [11] developed an analysis process for the recognition of the objects appearing in an image according to color criteria. Due to color variation existing in each area, a splitting–and–merging

algorithm was used so that the grouping of nodes is based on their belonging in one of the three regions of interest (meat, fat and background), but not on their color homogeneity.

In our project, there are 2–5 objects involved and the color variation seems unavoidable. We identify and remove the plate and background according to the clusters in HSB space; and then apply an algorithm, coming from the combination of region growing and splitting–and–merging algorithms based on HSB color space to segment the rest of the image into several parts:objects.

5.3 Color Segmentation

Theoretically, regions of a color segmentation should be uniform and homogeneous with respect to color characteristics. Interiors of regions should be simple and without many small holes. Adjacent regions of segmentation should have significantly different values with respect to the color characteristics. Boundaries of each segment should be simple, not ragged, and spatially accurate.

Achieving all these desired properties is difficult because strictly uniform and homogeneous regions are typically full of small holes and have ragged boundaries. Insisting that adjacent regions have large differences in values can cause regions to merge and boundaries to be lost.

There is no general theory of image segmentation, including color segmentation. Techniques in use have to differ precisely in the way they emphasize one or more of the desired properties and in the way they balance and trade off one desired property against another.

The objective of color segmentation in this system is to segment an image by color into several regions of which each region represents one of the food items

in the image, and each item in the image would have one region represented. Because classification is to follow, a region–based segmentation scheme would be preferred. The system uses two different schemes in two steps: first, remove out the background and the plate and then segment the remaining area.

5.3.1 Identification of Background and Plate

All images processed in this system have almost the same color background (grey) and the same color plate (white). This means that there are two clusters at almost the same positions in color space for every image. One represents the background and the other represents the plate. This makes it possible to detect the two clusters in HSB space and remove the corresponding points from the image. In fact, we can only use saturation and brightness as variables to detect the plate because of the randomness of its hue values. We can use hue, saturation, and brightness as variables to detect the background. In Table 5.1 are the average values of hue and saturation of blurred backgrounds. Values of brightness are from 90–140. In Table 5.2 are the average values of saturation of blurred plates. Values of brightness are from 200–255.

SLIDE	1	2	3	4	5	6	7
$\overline{H}$	164.88	168.24	160.81	142.45	139.57	144.62	162.71
$\overline{S}$	35.24	48.22	37.33	27.94	29.33	24.41	46.07

SLIDE	8	9	10	11	12	13	14
$\overline{H}$	180.76	167.10	190.24	179.71	151.48	160.40	170.77
$\overline{S}$	33.56	23.40	24.07	25.68	27.55	27.40	31.98

SLIDE	15	16	17	18	19	20	
$\overline{H}$	156.75	171.37	163.44	167.01	164.29	177.49	
$\overline{S}$	34.63	33.30	44.75	38.01	46.27	12.97	

Table 5.1 Average Hue and Saturation Values of Blurred Background

SLIDE	1	2	3	4	5	6	7
$\overline{S}$	6.36	19.43	10.81	8.07	8.89	9.17	10.29

SLIDE	8	9	10	11	12	13	14
$\overline{S}$	8.41	3.95	9.67	7.72	8.33	7.24	6.77

SLIDE	15	16	17	18	19	20	
$\overline{S}$	7.53	7.76	10.24	4.74	8.55	6.12	

Table 5.2 Average Saturation Values of Blurred Plate

In this step, classification and segmentation are completed simultaneously. From histogram analysis, the following thresholds are established. For backgrounds:

hue: 100–180

saturation: 0–40

brightness: 90–140 (5–6)

For plates:

saturation: 0–45

brightness: 200–255 (5–7)

Every pixel in the image is checked against these thresholds. The pixels that satisfy the conditions 5–6 are joined the region 'background'. The pixels satisfying the conditions 5–7 are joined the region 'plate'. Pixels in other areas are retained for the next segmentation procedure.

5.3.2 Segmentation of Food Items

Color variation of food items in images could be larger than the color variation of backgrounds and plates. But color differences among different food items should maintain certain values, e.g. different colored items should have some

color differences whenever we make transparencies. Because of the color variation and more than one object to be segmented, the histogram thresholding method is not used, since some foods could have overlapping histograms and a threshold could not be fixed. Instead, an algorithm coming from the combination of region growing and region splitting–and–merging approaches is designed to segment food items.

In this algorithm, every pixel in an image will be compared with its neighborhood with respect to hue, saturation, and brightness. Thresholds of differences of the three values are set on the basis of the training data by calculating the variances inside every item and checking the color values jumping between different items for about 30 slides (see also 4.1). If the differences between the pixel and any of the neighbors are smaller than the thresholds, the pixel will join the region in which the neighbor is located. If none of the neighbors have values close to those of the pixel, the pixel will start a new region.

Using the algorithm in a computer program, comparing pixel values starts from the left–top corner of the image and proceeds to the right–bottom, pixel by pixel and row by row. Every pixel's values are compared with the values of the left neighboring pixels and the above neighboring pixel. If the pixel is close enough to the left neighborhood with respect to the values of hue, saturation, and brightness, it joins the region which the left neighborhood belongs to. If the pixel is close enough to the above one with respect to the three values, it joins the region which the above neighbor belongs to. If the pixel is close to neither the left one nor the above one, a new region starts from this pixel. Figure 5.2 illustrates a simple example of the algorithm.

In Figure 5.2, the comparing starts at the pixel P_1. Region R_1 starts at P_1 since there is no pixel which could be compared with it. P_2 is compared only to P_1, since

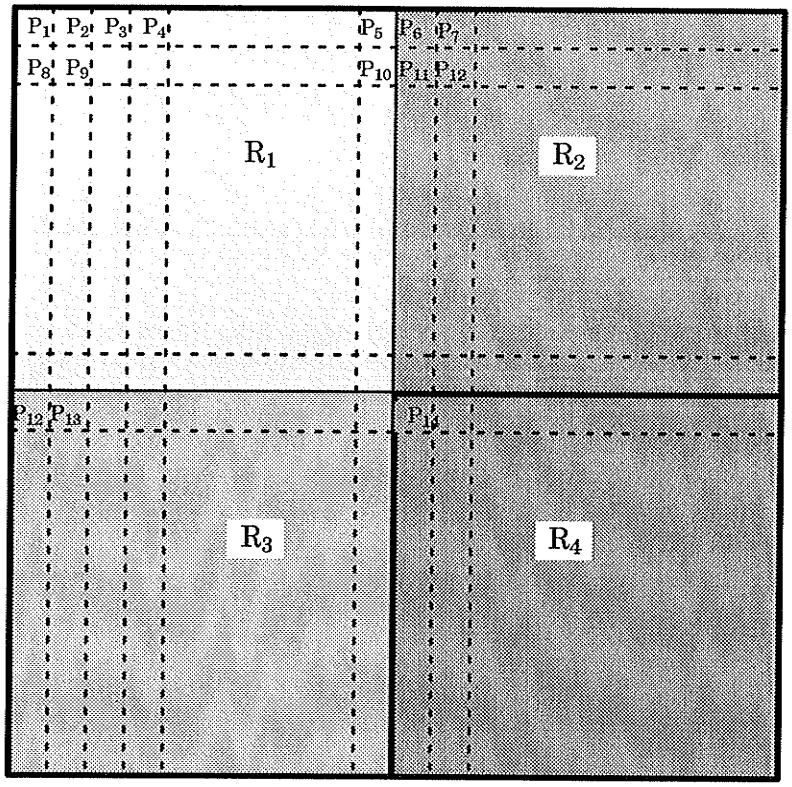


Figure 5.2 Illustration of the Segmentation Algorithm

there is no pixel above P_2. P_2 joins to R_1 because P_1 and P_2 are close enough. Then the same thing is done with P_3, P_4, and the succeeding pixels in the first row until pixel P_6 is reached. Since P_6 is not close to P_5, it starts a new region R_2. The pixels to the right of P_6 in the row join R_2 because they all have values below the R_2 threshold. In the second row, P_8 is compared to P_1 and joins R_1. P_9 is compared to P_8 and P_2 and is also joined R_1 too. So do the succeeding pixels until P_{11}. P_{11} is compared to P_{10} and to P_6. It joins R_2 because it is not close to P_{10} but is close P_6. The succeeding pixels join R_2, since they are close both to the left neighboring pixels and to the above pixels. The pixels in the following row are compared in the same way, except that P_{12} starts region R_3 and P_{14} starts the region R_4.

This simple logic could not segment an image well when the situation shown in Figure 5.3 happens. We suppose there are four regions in Figure 5.3, R_1, R_2,

R_3+R_4, and R_5+R_6. In Figure 5.3, P_1 would start a new region R_3 because it is close neither to the left neighbor nor to the above neighbor (they are in R_1). P_2 would start another new region R_4. But in fact, region R_3 and R_4 are the same thing. For the same reason, every pixel on $line_1$ would start a new region. Therefore R_6 would be segmented into many small vertical regions with one pixel width.

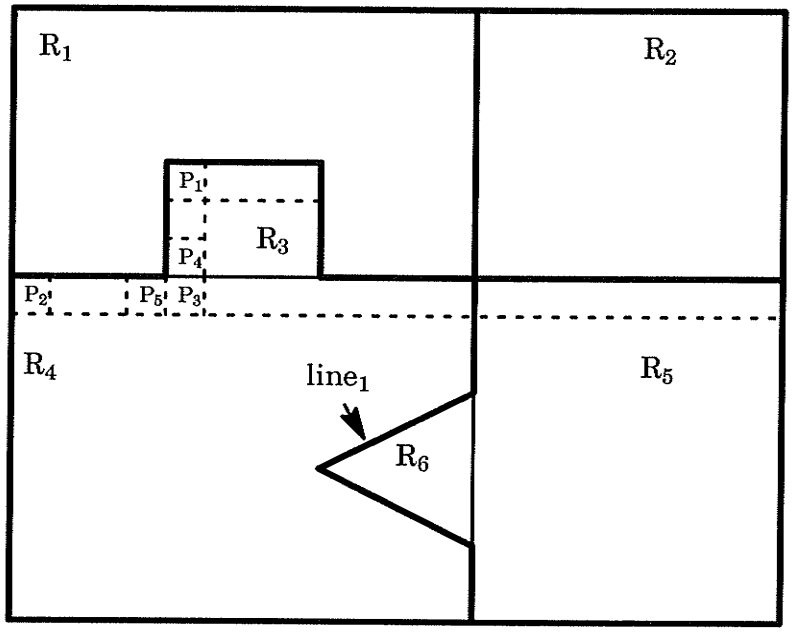


Figure 5.3 Check–Back Algorithm

A check–back algorithm is applied to solve this problem. In this algorithm, every pixel, except those without left neighbors or above neighbors, is compared with its left neighborhood and its above neighborhood. The pixels that are close enough in their values are given a similar sign to indicate that they are in the same region. If it is close both to the left neighboring pixel and to the one above, the sign of the left neighboring pixel is compared to the sign of the above neighboring pixel. If the two signs are the same, the pixel will take the same sign, meaning that the pixel joins the region. This causes no problem. If the two signs

are different, a problem occurs. This means that the two close pixels are in two different regions. In this case, the sign of the above neighboring pixel is given to the pixel, and all pixels with the same sign as the left neighboring pixel are checked out and given the sign of the above neighboring pixel.

In Figure 5.3, P_1 starts R_3 and P_2 starts R_4. When P_3 is compared to its above neighboring pixel P_4 and the left neighboring pixel P_5, it is found that P_3 is close both to P_4 and to P_5 and that the sign of P_4 is different from the sign of P_5. P_3 takes the sign of P_4, and all pixels which already have a sign are checked. Any pixel with the sign of P_5 changes its sign to the sign of P_4. Therefore the region R_4 disappears, and all pixels that used to be in region R_4 join R_5.

Figure 5.4 illustrates the complete algorithm of segmentation, where Hth, Sth, and Bth are thresholds of hue, saturation, and brightness respectively.

The first steps are for identifying background and plate using the rules (5–6) and (5–7). The next steps segment the remaining area by examining the difference of the values of hue, saturation, and brightness between a pixel and its upper neighbor and left neighbor. If the difference between the pixel and its upper neighbor is smaller than the thresholds, it joins the region to which the upper neighbor belongs. If this difference is larger than the thresholds, but the difference between the pixel and its left neighbor is smaller than the thresholds, it joins the region to which the left neighbor belongs. If neither is smaller than the thresholds, the pixel starts a new region. If both the difference are smaller than the thresholds, but the regions to which the upper neighbor and the left neighbor belong, respectively, are different, the check–back algorithm is applied and the region to which the left neighbor belongs joins the region to which the upper neighbor belongs. The pixel joins the same region. The procedure starts at the

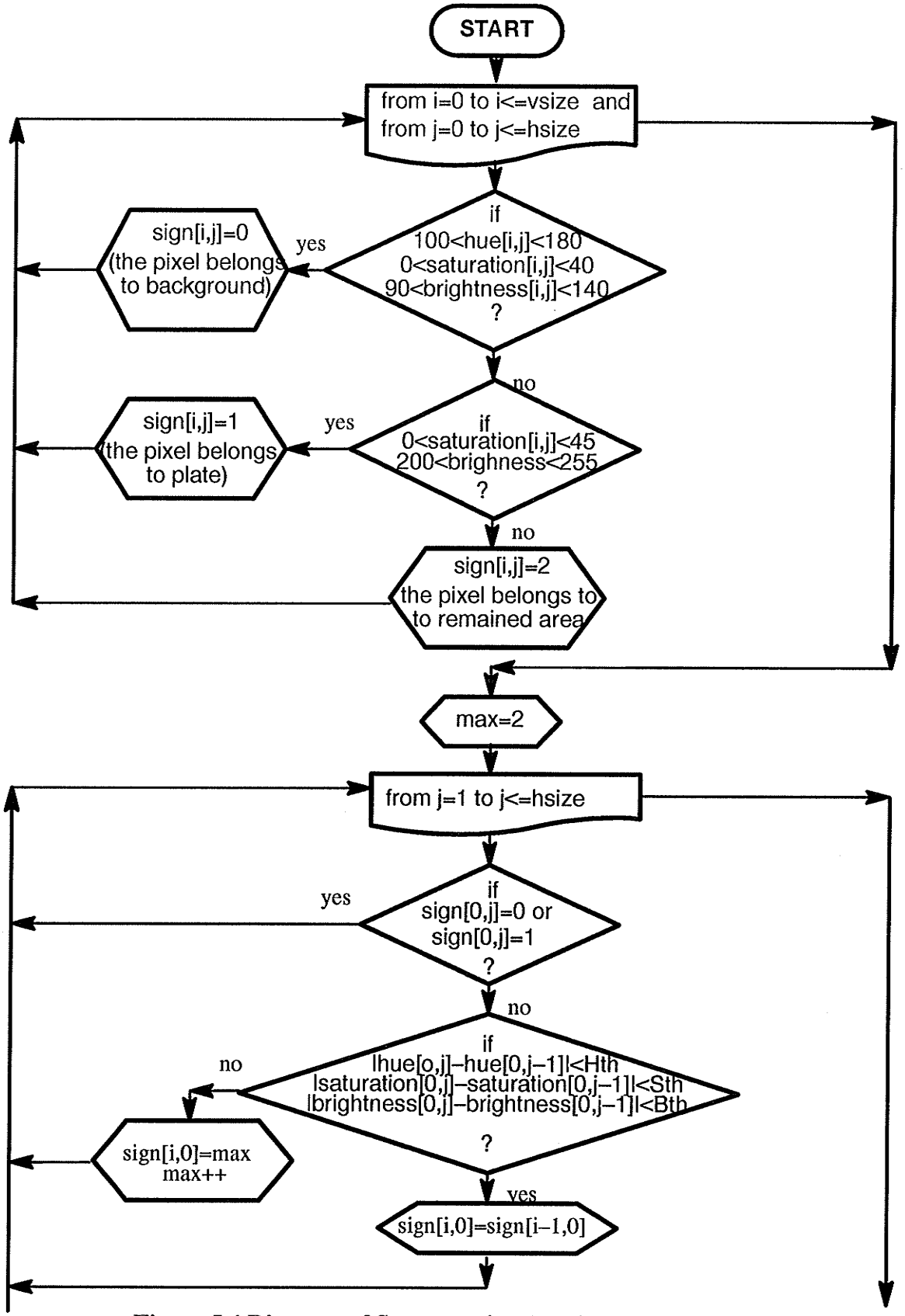


Figure 5.4 Diagram of Segmentation (continued on pp. 48–49)

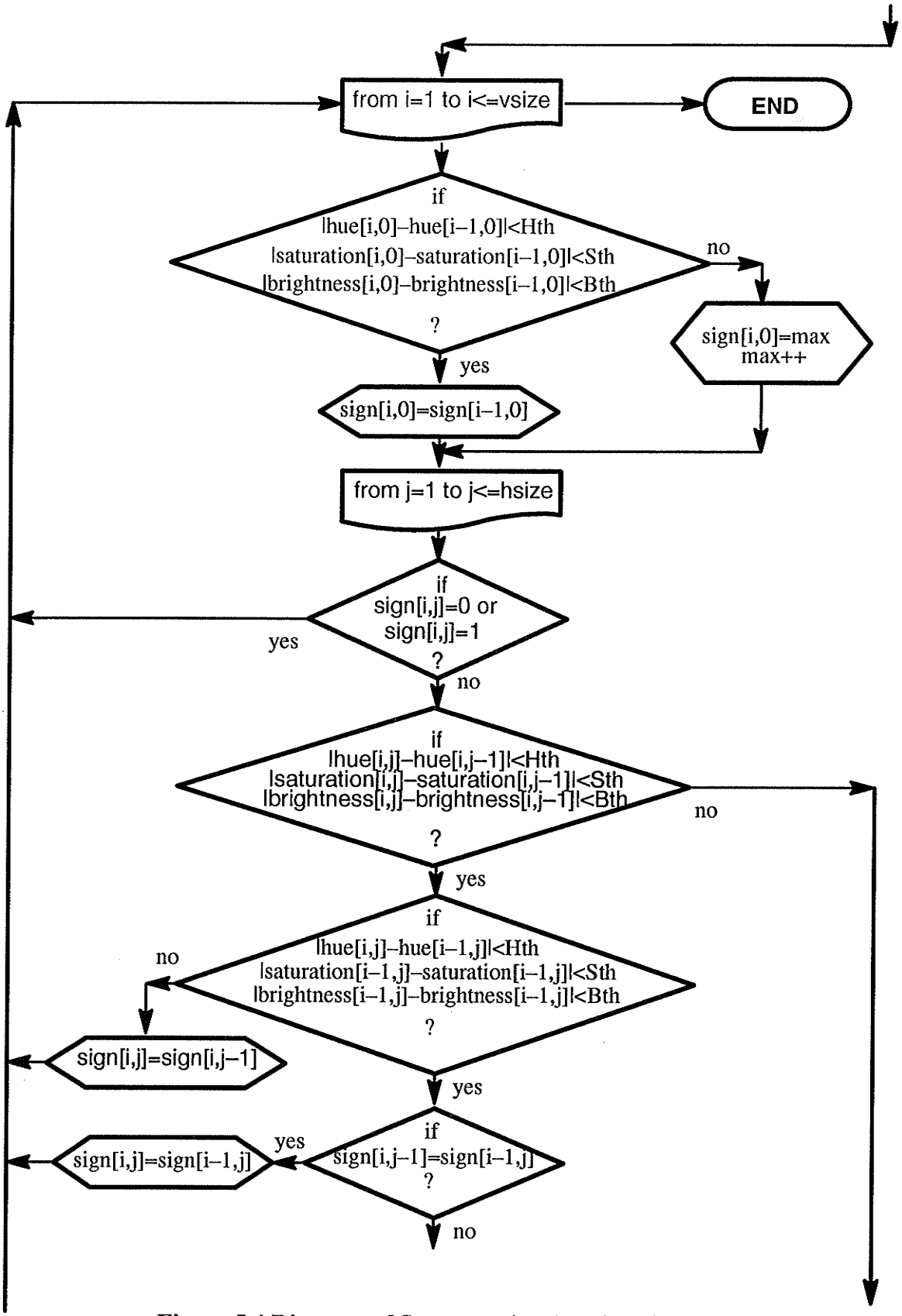


Figure 5.4 Diagram of Segmentation (continued on pp. 49)

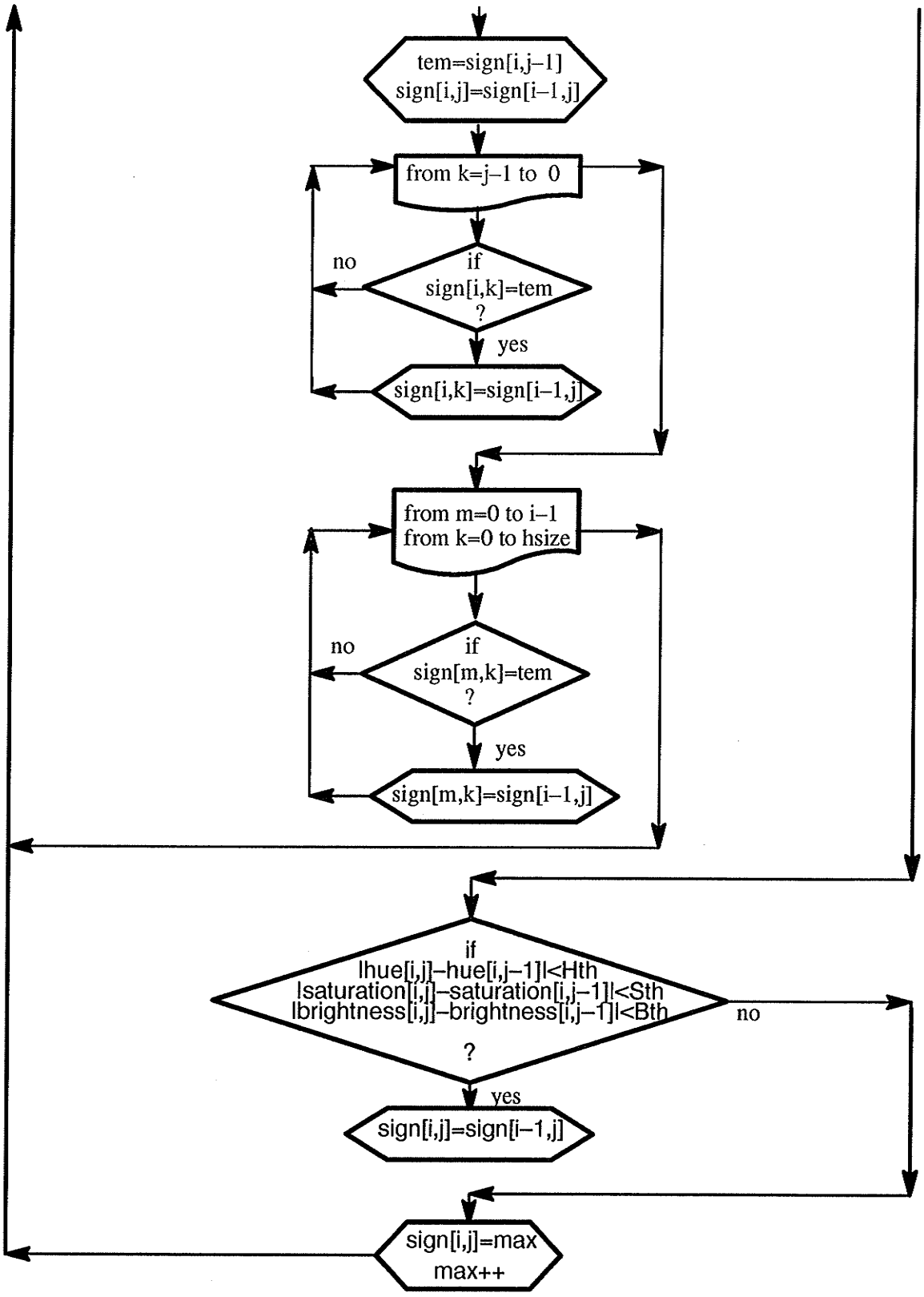
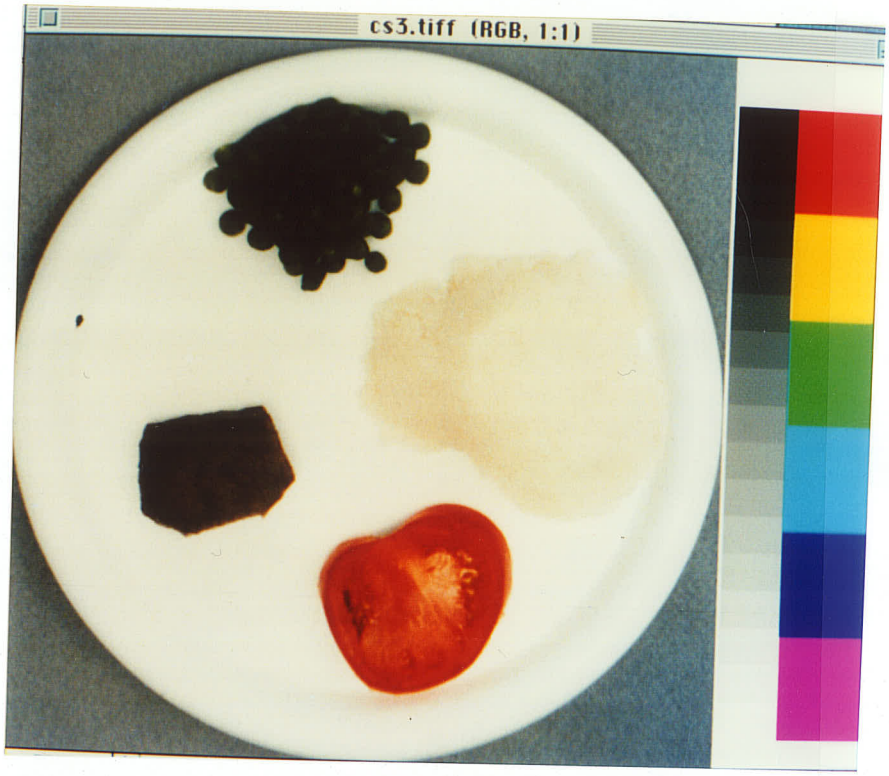


Figure 5.4 Diagram of Segmentation

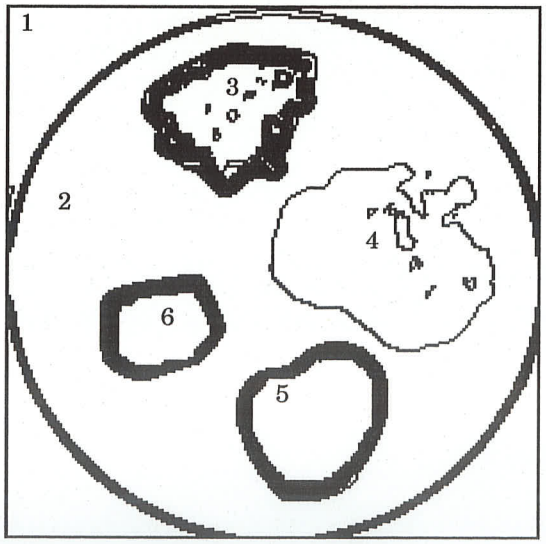
upper–left corner of the image and ends at the bottom–right corner. The examination is implemented pixel by pixel.

Figure 5.5(b) (an image as the result of segmentation) is the result of the segmentation of Figure 5.5(a) (a food image).The segments are background(1), plate(2), peas(3), potato(4), tomato(5), and beef(6).

More segmentation results are shown in Appendix B. In Figure 5.5(b) and segmentation results in Appendix B, a black pixel indicates that the color change between that point and its neighbor is large than the threshold. A white pixel indicates a small change or on change.



(a) the Image



(b) the Segmentation

Figure 5.5 Segmentation of Color Image

CHAPTER 6 CLASSIFICATION

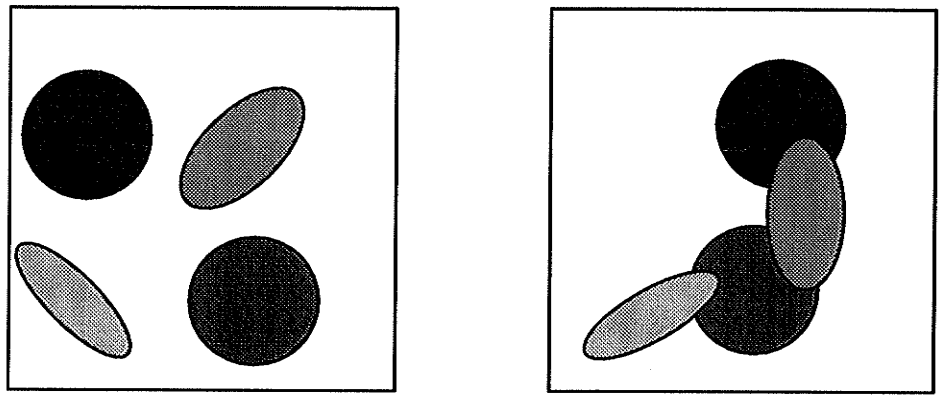
6.1 Classification

Classification and segmentation processes have closely related objectives. Classification, which classifies the object into one of several categories, can lead to segmentation, and vice–versa. Classification of pixels in an image is another form of component labeling that can result in segmentation of various objects in the image, which is what we have done for removal of background and plate.

Classification is generally implemented in a feature space. A feature can be any variable, with which we identify one object from another, such as gray levels, shapes, sizes, or colors. A set of P features is extracted from the image. These features may either be pixel–based or object–based. In the pixel–based case, it is often impossible to perform a segmentation with a single feature. More than one feature for each pixel is needed to separate different classes of objects from each other and from the background. In the object–based classification, the objects could already be separated from the background. We can calculate all the average values of pixel–based features over the whole area of the object. These features could be gray level value, color value (RGB or HSB), or local orientation. Based on these, the shape of the objects could be described. The object–based classification is preferable, if possible, since much less data must be handled; that is only one set of P features for each object detected.

The set of P features forms a P–dimensional space, which is denoted as the feature space. Each pixel or object is represented as a feature vector in this space. If the features represent an object class well, all feature vectors of the ob-

jects from this class should lie close to each other in the feature space. By statistical calculation, we can build a P–dimensional histogram. A narrow peak in the histogram represents a cluster. It will be possible to separate the objects into the given object classes if the clusters for the different object classes are well separated from each other, as shown in Figure 6.1(a). With less suitable features, the clusters overlap each other (Figure 6.1(b)). In this case, an error–free classification is not possible [18].



(a) Well Separated Object Classes (b) Overlapping Object Classes

Figure 6.1 A Two–dimensional Feature Space With Four Object Classes [18]

If we perform classification by analyzing the structure of the feature space, one object is thought of as a pattern in the feature space. There are two basic approaches to classification: supervised and nonsupervised. By supervised classification, we determine the clusters in the feature space with known objects before hand and find out the number of classes, and their location and extension in the feature space. With unsupervised classification, no knowledge is presumed about the objects to be classified, even the number of classes. The result comes from the analysis of the clusters in the feature space and the separation of the

clusters. This method is more objective, but it may result in a less favorable separation.

There are learning methods if the feature space is updated by each new object which is classified. Learning methods can compensate any temporal trends in the object features. Such trends may be due to simple reasons such as changes in the illumination which could easily occur in an industrial environment because of changes in daylight, aging or dirtying of illumination systems [18].

There are different classification techniques. As for supervised classification, which our system belongs to, the simplest classification method is the look–up method, known as labeling. By this method, we give a number indicating the object class to each point in the discrete feature space. This approach is difficult only if the distributions of two classes overlap. When this happens, we could either take the class which shows the higher probability at this point or argue that an error–free classification is not possible with this feature. The same attribute is given to all the points in the feature space which do not belong to any object class. Then the only thing that we have to do is to look up which class a feature vector belongs to. We regard the feature space as a multi–dimensional look–up table. This method is the best with respect to the computing time. However, concerning the memory needed to store the feature space it is not so advantageous. A three dimensional $64 \times 64 \times 64$ feature space already requires 1/4 Megabyte memory. Consequently, the look–up method is only feasible for low–dimensional feature spaces.

All other classification methods model the pattern classes in the feature space to reduce storage requirements. The box method approximates a class by a surrounding box. If this method does not work, we can use the minimum distance method to determine which class a feature vector belongs to: we compute the dis-

tance of the feature vector to all cluster centers and choose the class which has the minimum distance. The maximum probability method and other methods could be used when the above methods do not work well [18].

6.2 Color Classification--Survey of Related Research

Color classification is usually combined with color segmentation, and uses the result of segmentation. In this case, a color space could be a feature space. An intuitive method for color classification is the detection and analysis of vector clusters in color space (it could be any space which can represent a color image appropriately). By the analysis of histograms of every component, the clusters which represent certain objects could be found. By comparing the properties of clusters with previous knowledge (from a training procedure), decisions could be made.

Tominaga [12][13][14] did color classification of natural color images with a similar method based on uniform color spaces, which partitioned color image data into a set of uniform color regions. The ability to classify spatial regions of the measured image into a small number of uniform regions can also be useful for several essential problems of color image analysis including color segmentation. The input image data are mapped from device coordinates (RGB) into an approximately uniform perceptual color space (L*a*b*). Colors are classified by means of cluster detection in the uniform color space. The process is composed of two stages of basic classification and reclassification. The basic classification is based on histogram analysis to detect color data, which are extracted for effective discrimination of clusters. At the reclassification stage, the representative colors extracted by the color classification are reclassified on a color distance.

There is also a scheme of color classification which is not based on 3–D color space. J. Parkkinen and T. Jaaskelainen[15][16][17] proposed a vector subspace method of color classification which used the whole color spectrum instead of three–parameter methods (RGB, HSB, etc.). To use this method, sometimes resulting in improved accuracy, we should calculate the whole color spectrum of each image and each probable item which could exist in the images.

6.3 Classification--Program and Result

In this system, we apply color classification to an image after it is segmented into several subregions which could represent some food items. So the classification is object–based. It is also supervised, since we have training data (beforehand knowledge) before the classification is applied to images.

The simplest clustering method is applied for this system. There are ten items to be identified. They are tomato, carrot, orange, broccoli, green peas, green apple, potato, corn, roast beef, and toasted bread. Average values of hue and saturation are calculated for each item in every image. About ten to twenty samples were checked before we set the parameters as shown in Table 6.1.

As a first parameter we consider the hue. According to their distribution, we can divide the ten items into six groups. Group 1 includes the tomato and the carrot. Group 2 includes the orange, the beef, and the toast bread. Group 3 includes the corn. Group 4 includes the potato. Group 5 includes the green apples. Group 6 includes the broccoli and the green peas. Items in different groups do not overlap. But items in the same groups have some areas overlapped. So hue alone is not sufficient to separate all items. Similarly, saturation alone is not sufficient to separate all items either. However, using both parameters, we can sep-

arate almost all of these ten items. In group 1, although they have some common area in hue, their saturation values are different. In group 2, orange, beef, and toast also have different distributions of saturation values. However in group 6, broccoli and green peas not only have the same hue area but also have the same saturation area. By observing their brightnesses and histograms, we found out that they even have same brightness area and similar histograms. With this result, we can only conclude that broccoli and green peas could not be separated only by color. This conclusion is not surprising, since it is even difficult to separate them by human eye only by color.

	Tomato	Carrot	Orange	G–Apple	Broccoli
Hue	3–7	6–9	12–18	45–50	62–82
Saturation	210–225	230–250	225–240	180–195	90–170

	G–Peas	Potato	Corn	R–Beef	T–Bread
Hue	52–74	29–38	24–28	11–17	14–19
Saturation	115–240	40–90	215–230	140–180	185–220

Table 6.1 Parameters for Classification of Food

The following are the conditions used in the program for color classification shown in Table 6.1.

$$Tomato: \quad 3 \le hue \le 7 \quad \&\& \quad 210 \le saturation \le 225 \tag{6–1}$$

$$Carrot: \quad 6 \le hue \le 9 \quad \&\& \quad 230 \le saturation \le 250 \tag{6–2}$$

$$Orange: \quad 12 \le hue \le 18 \quad \&\& \quad 225 \le saturation \le 240 \tag{6–3}$$

$$G-Apple: \quad 45 \le hue \le 50 \quad \&\& \quad 180 \le saturation \le 195 \tag{6–4}$$

$$Broccoli: \quad 62 \le hue \le 82 \quad \&\& \quad 90 \le saturation \le 170 \tag{6–5}$$

$$G-Peas: \quad 52 \leq hue \leq 74 \quad \&\& \quad 115 \leq saturation \leq 240 \tag{6–6}$$

$$Potato: \quad 29 \leq hue \leq 38 \quad \&\& \quad 40 \leq saturation \leq 90 \tag{6–7}$$

$$Corn: \quad 24 \leq hue \leq 28 \quad \&\& \quad 215 \leq saturation \leq 230 \tag{6–8}$$

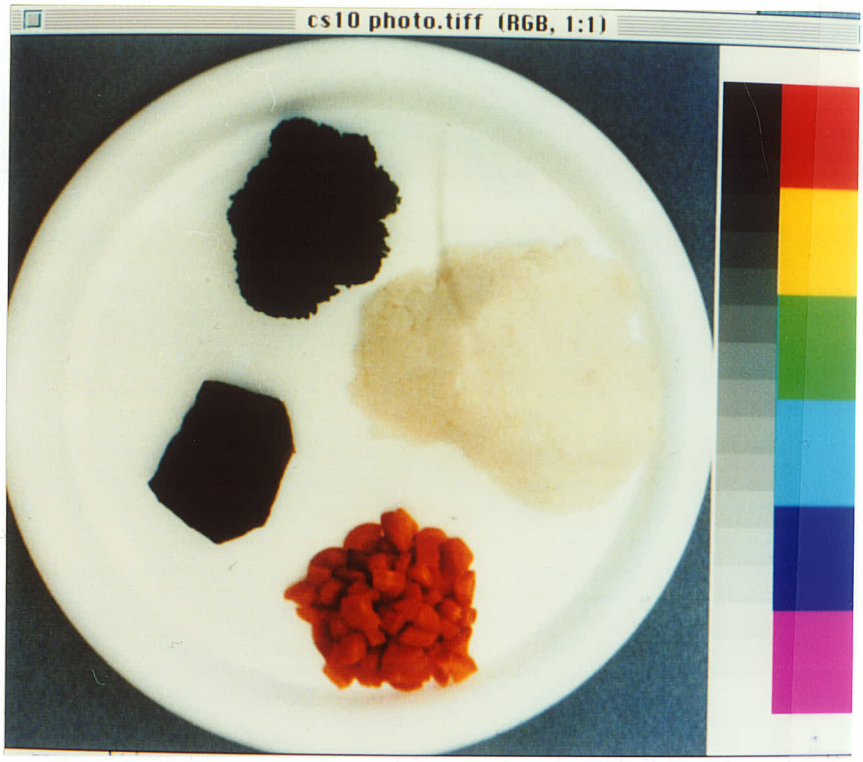
$$Beef: \quad 11 \leq hue \leq 17 \quad \&\& \quad 140 \leq saturation \leq 180 \tag{6–9}$$

$$T-Bread: \quad 14 \leq hue \leq 19 \quad \&\& \quad 185 \leq saturation \leq 220 \tag{6–10}$$

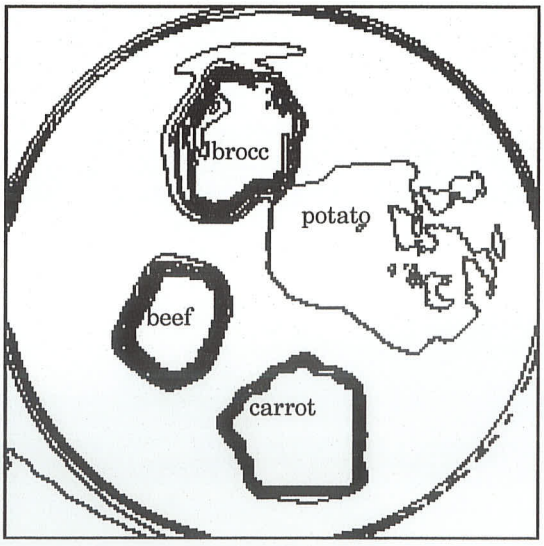
Figure 6.2(b) is the result of the segmentation and classification of Figure 6.2(a).

Since there are still some details left after blurring, the result of segmentation contains some small regions. Because very small sizes of food are not expected to be served, these small regions are only the subregions of some large regions, which represent food items. They are not important for classification. So we ignore them and only calculate the average hue and saturation values of large regions. The average values do not change much in the absence of those small regions. Therefore, the result of classification is not significantly affected by ignoring these small parts. A threshold is set to decide if a region is small enough to be ignored. Threshold for this system is 200 pixels.

Ten slides are tested by the segmentation and classification program. Figure 6.3 shows the rules in the Hue–Saturation plane for classification expressed by (6–1)–(6–10) and clusters of every object from every slide. They group well except broccoli and peas, where 1 is tomato; 2 carrot; 3 orange; 4 green apple; 5 broccoli; 6 peas; 7 potato; 8 corn; 9 beef; 10 toast.



(a) image



(b) result of segmentation

Figure 6.2 Segmentation and Classification of Image by Color

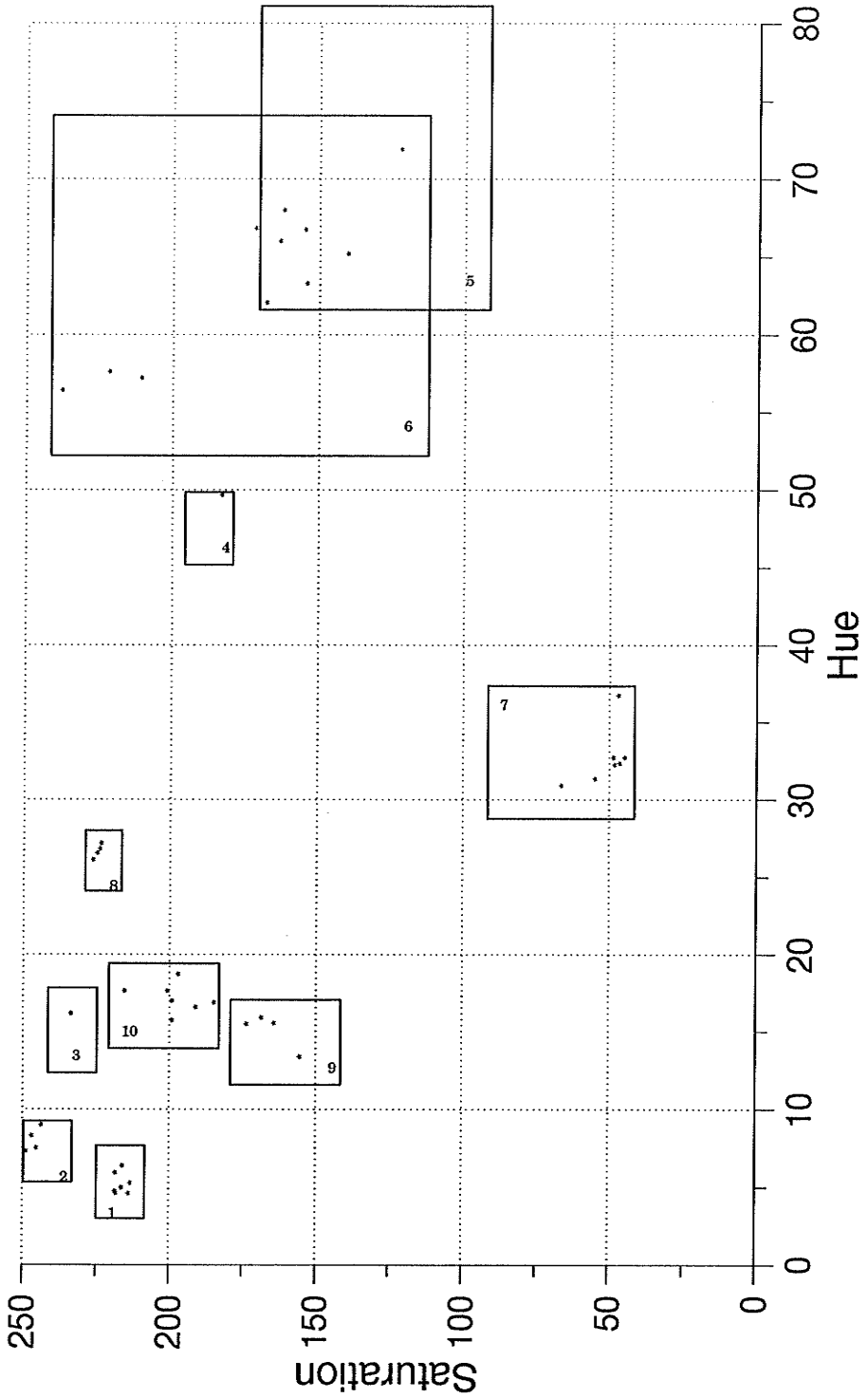


Figure 6.3 Clusters of Segments

CHAPTER 7 CONCLUSION AND DISCUSSION

7.1 Conclusion

The objective of this project is to study the possibility of identifying food items by color.

The first question presented to us was which color system is the best. The intuitive choice is the RGB system, since it comes directly from the digitization. By checking more than fifty color transparencies of food plates, we found out that the HSB system is more reliable. Using the HSB color system can save processing time too, because most of the classification (except plate and background) can be implemented without the parameter B.

The second problem addressed was segmentation: the processing time for segmentation is too long and the result of segmentation contains too much detail. Therefore this would cause a problem: how to classify so many small regions, especially when a small region can not represent a food item. In this case, a blurring filter was chosen to blur the details and smooth the images. The result is very satisfactory. We can get the results of segmentation with very good quality after blurring.

Segmentation and classification by color are the major tasks in this project. After examining thirty transparencies, thresholds of hue, saturation and brightness differences were set for segmentation, and parameter conditions were set for classification. The programs developed for color segmentation and classification all worked successfully. They provide a method to segment and classify objects in an image by color. The result shows that segmenting and classifying food

items in plate images by color is a feasible method. Combined with more features of images, it could lead to a successful automatic food identification in images.

7.2 Discussion and Recommendation

The goal for this project has been achieved. However, there are still problems remaining to be studied in further work.

1. Different background light could make the color of images very different. Therefore, the colors of food items could have big variations, which could make the classification difficult, or even impossible. A color adjustment pre–processing could reduce the effect of this color variation on segmentation and classification. But this procedure could not eliminate the effect totally. The other method may by more effective: to design simple, but special photographic equipment to make transparencies, which have fixed background light for every exposure. Dr. Sevenhuysen already had special equipment for this purpose. Only a little improvement is need to avoid color variation.

2. Some food items have almost the same colors. It is even difficult for human eyes to identify them only by color. In this case, segmentation and classification only by color is not enough. Other features, such as sizes, shapes, and textures, should be introduced for the segmentation and classification. For most food items one more feature added into the segmentation and classification system could be enough. The only cost is a little speed reduction and a larger memory requirement, which would not cause a big problem for the purpose of segmentation and classification.

REFERENCE

[1]. Rafael C. Gonzalez, and Richard E. Woods, *Digital Image Processing*, Addison–Wesley Publishing Co.Ltd, 1992.

[2]. G. P. Sevenhuysen, and L. A. Wadsworth, "Food Image Processing: A Potential Method for Epidemiological Surveys", *Nutrition Reports International*, Vol. 39, 1989.

[3]. G. P. Sevenhuysen, W. V. Steveren, K. Dekker, and E. Spronck, " Estimates of Daily Individual Food Intakes Obtained by Food Image Processing", *Nutrition Research*, Vol. 10, 1990.

[4]. A. Rosenfeld, and A. C. Kak, *Digital Picture Processing*, Academic Press, New York, 1982.

[5]. R. Weill, and Y. Nes, "Use of Color Vision for Citrus Plucking", *Proceedings of the IFIP TC5 / WG 5.3 International Conference*, 1990.

[6]. R. B. Ohlander, "Analysis of Natural Scenes", Doctoral Dissertation, *Carnegie–Mellon University*, Pittsburgh, PA. 1975.

[7]. D. C. Baker, S. S. Hwang, and J. K. Aggarwal, "Detection and Segmentation of Man Made Objects in Outdoor Scenes: Concrete Bridges", *Optical Society of America*, Vol.6, No.6, 1989.

[8]. Y. Miyake, H. Saitoh, H. Yaguchi, and N. Tsukada, "Facial Pattern Detection and Color Correction From Television Picture for Newspaper Printing", *Journal of Imaging Technology*, Vol.16, No.5, 1990.

[9]. S. H. Ong, and C. C. Hew, "Segmentation of Color Images Based on Iterative Thresholding and Merging", *ICIP 92. Proceedings of the 2nd Singapore International Conference on Image Processing*, 1992.

[10]. D.C. Tseng, and D. H. Cheng, "Color Segmentation Using Perceptual Attributes", *11th IAPR International Conference on Pattern Recognition*, Vol.3, 1992.

[11]. F. Diaz Pernas, and J. Lopez Coronada, "A New Method for Recognition Using Color Image Segmentation", *Proceedings of the 16th International Conference on Image Analysis and Processing*, 1992.

[12]. S. Tominaga, "Color Classification of Natural Color Images", *Color Research & Application*, Vol.17, Iss.4, 1992.

[13]. S. Tominaga, "A Color Classification Method for Color Images Using A Uniform Color Space", *IEEE Comput. Soc. Press, 1990.*

[14]. S. Tominaga, "Color Classification of Color Images Based On Uniform Color Spaces", *Proceedings of The SPIE–the International Society for Optical Engineering, 1990.*

[15]. T. Jaaskelainen, S. Toyooka, S. Izawa, and Kadono, "Color Classification by Vector Subspace Method and Its Optical Implementation Using Liquid Crystal Spatial Light Modulator", *Optics Communications, Vol.89, Iss.1, 1992*

[16]. J. Parkkinen, and T. Jaaskelainen, "Color Vision:Machine and Human", *Optical Engineering*, Vol.1199, Iss.pt.2, 1989.

[17]. J. Parkkinen, E. Oja, and T. Jaaskelainen, "Color Analysis by Learning Subspaces and Optical Processing", *IEEE International Conference on Neural Networks*, Vol.2, 1988.

[18]. B. Jahne, *Digital Image Processing*, Springer–Verlag, 1991.

[19]. *Barneyscan Manual*, Bayneyscan Cooperation, Berkeley, California, 1989.

APPENDIX A DESCRIPTION OF BARNEYSCAN

Barneyscan for the Macintosh is a color desktop scanning system which will quickly transform 35mm slides into digital information. We can immediately display this information as an image on a Macintosh II monitor. With software support supplied by Barneyscan, we can manipulate, adapt, and store the image information. Then we can readily transfer it to other applications, to merge into publications, presentation graphics, an image data–base, or in other chosen context.

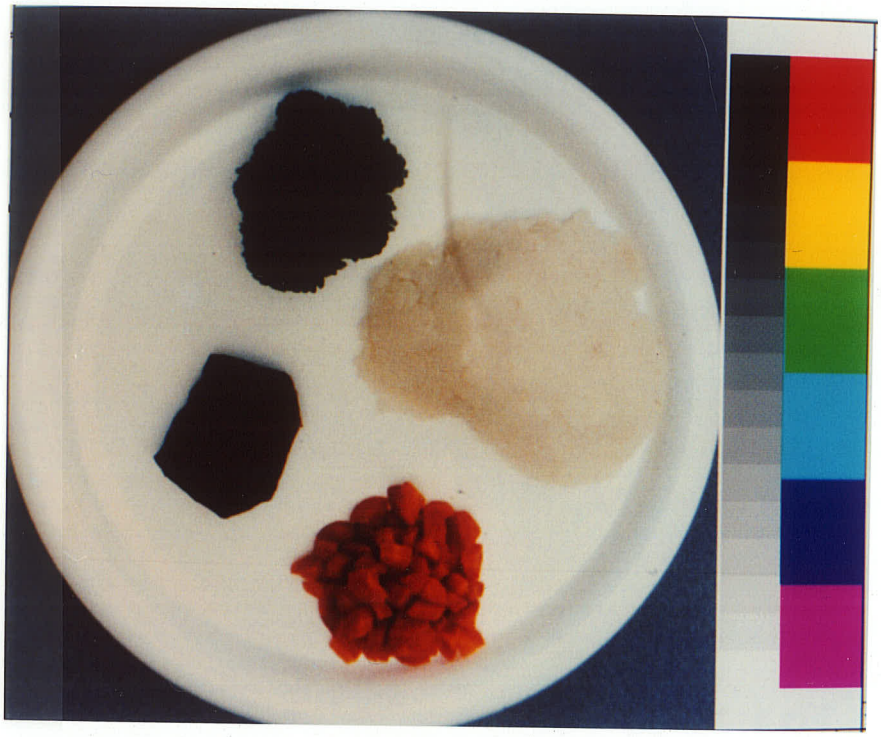
The process of scanning a slide casts a narrow band of light through a slide, then through a colored or neutral gray filter. The portions of spectrum that are not absorbed by the slide or filter reach a photo–detector consisting of a stack of 1024 sensors. Each sensor responds to the light that has passed through an area of the slide about a thousandth of an inch square. Every sensor reading is assigned a numerical value on a scale of 0–255; corresponding to no light received and the maximum received, respectively. The slide is then moved over very slightly, and the next vertical slice of the slide is scanned. This process is repeated 1520 times, until the entire width of the slide has been scanned. As a result, we get an image with resolution 1024×1520 which can not be adjusted. A color scan uses three color filters in succession to measure the red, green and blue components of the image. The full color image seen on the screen combines the data from the red, green and blue scan.

Barneyscan image editing software can do most of things which an image processing software can generally do. It can display an image (either in color or gray scale) on screen with magnification ratio from 16:1 to 1:16. It is available to read gray–level value or color values (RGB) and position information on the image.

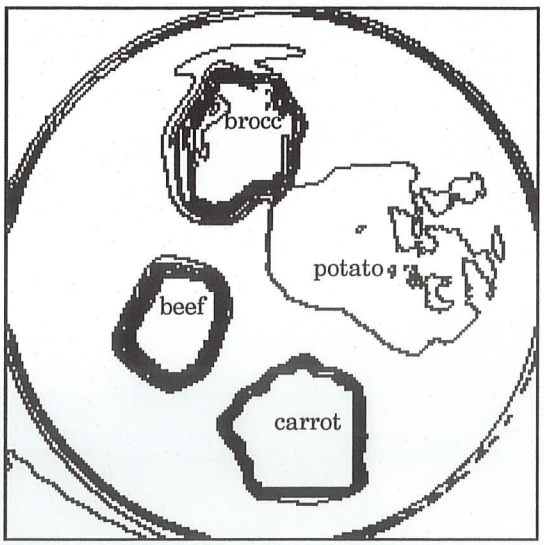
It can make color changes to images, including adjustment of brightness, contrast, gamma, and color balance. There are some built–in filters we can use directly to modify images, including an additive noise filter, smoothing filters, sharpening filters, and so on. Different types of image (RGB, HSB, HSL—L is lightness) can be converted between each other. All these image types contain more than one channel. Barneyscan can split or merge channels of an image and do some editing on a single channel. From the tool palette, we can choose an editing function from the grabber to scroll the image, the zoom tool to zoom in or zoom out the image, the eraser to erase part of the image, the smudge tool to simulate the effect of dragging a finger through wet paint, the blur tool to blur a part of the image, the sharpen tool to sharpen part of an image, the eyedropper to select the current foreground and background colors from colors in an image, the pencil to create either freehand or straight lines, the paint brush to paint the foreground color into the image, the airbrush to lay down a diffused spray of paint on an image, the line tool to draw a straight line on an image, the rubber stamp tool to pick up a sample of a particular part of an image and place an exact copy or a modified version of that part on the same image or some other image, the paint bucket to fill areas with the foreground color, and the blend tool to create a gradient fill.

The functions used for this project are: capturing the image through scanning, converting an RGB image to an HSB image, and using a Gaussian blur filter. The author developed the other programs for this project [19].

APPENDIX B SEGMENTATION RESULTS

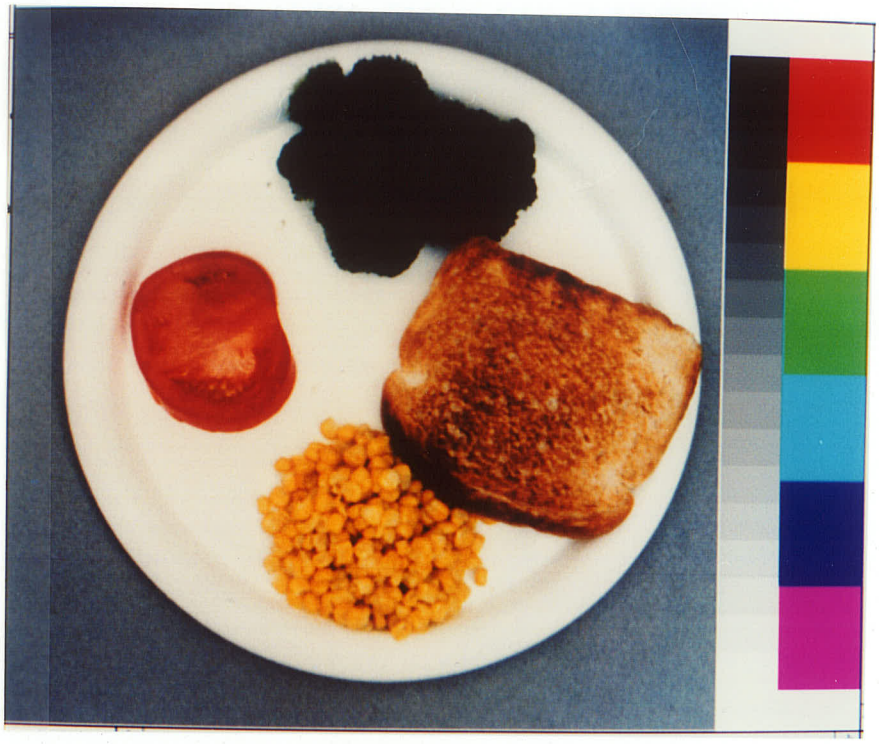


(a) The Image

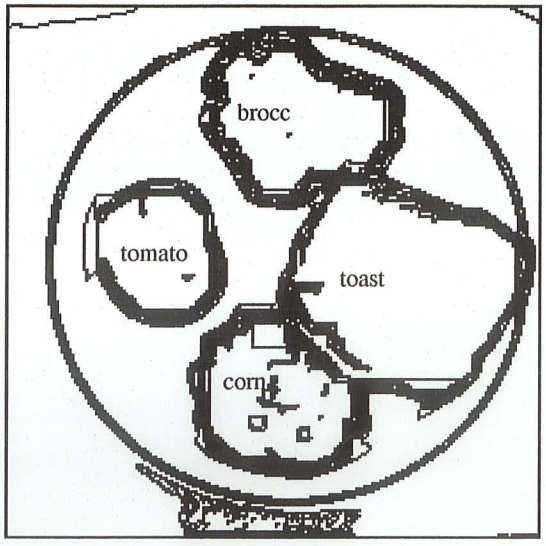


(b) The Segments

(I)

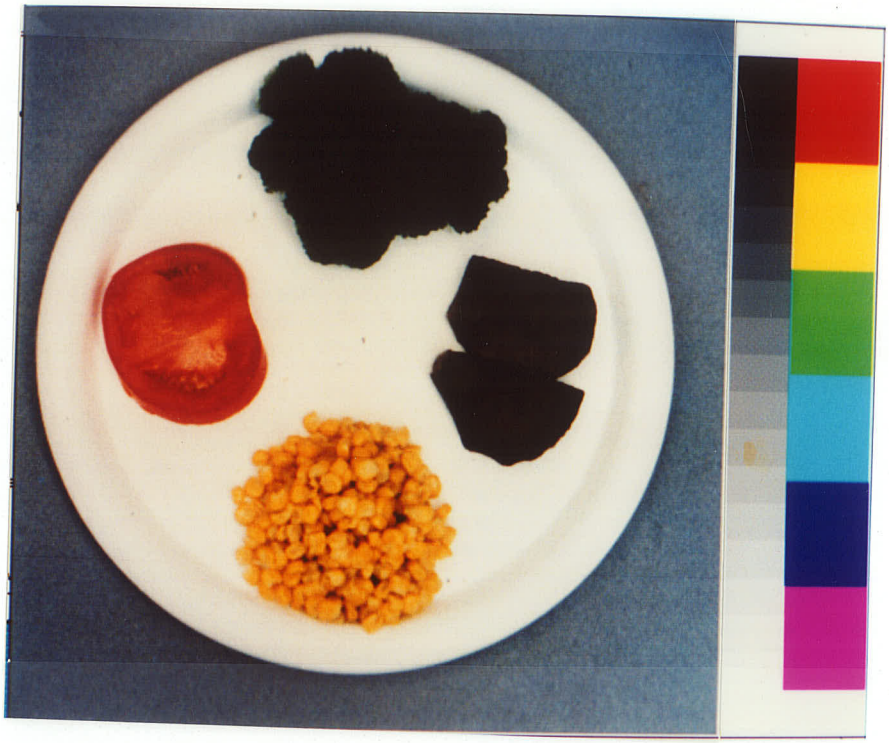


(a) The Image

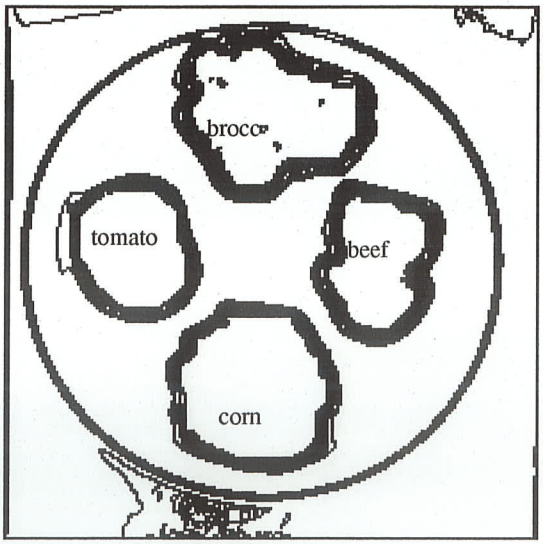


(b) The Segments

(II)

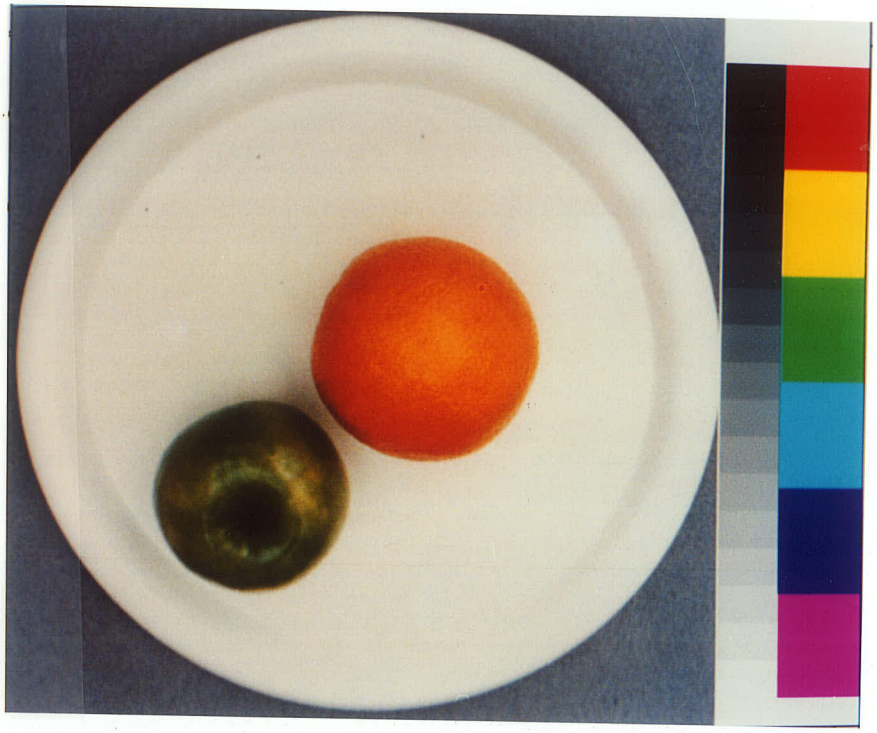


(a) The Image

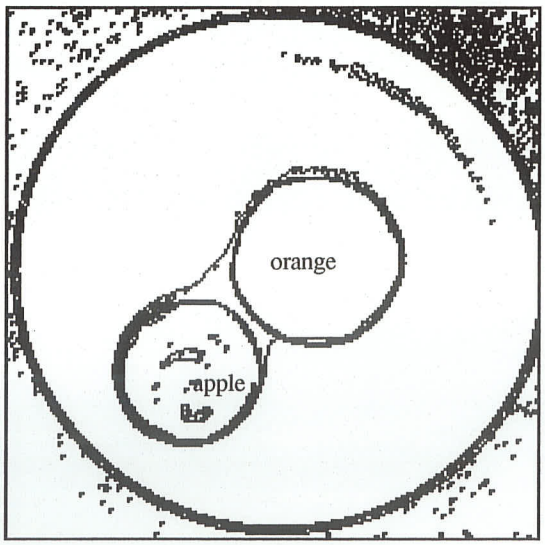


(b) The Segments

(III)

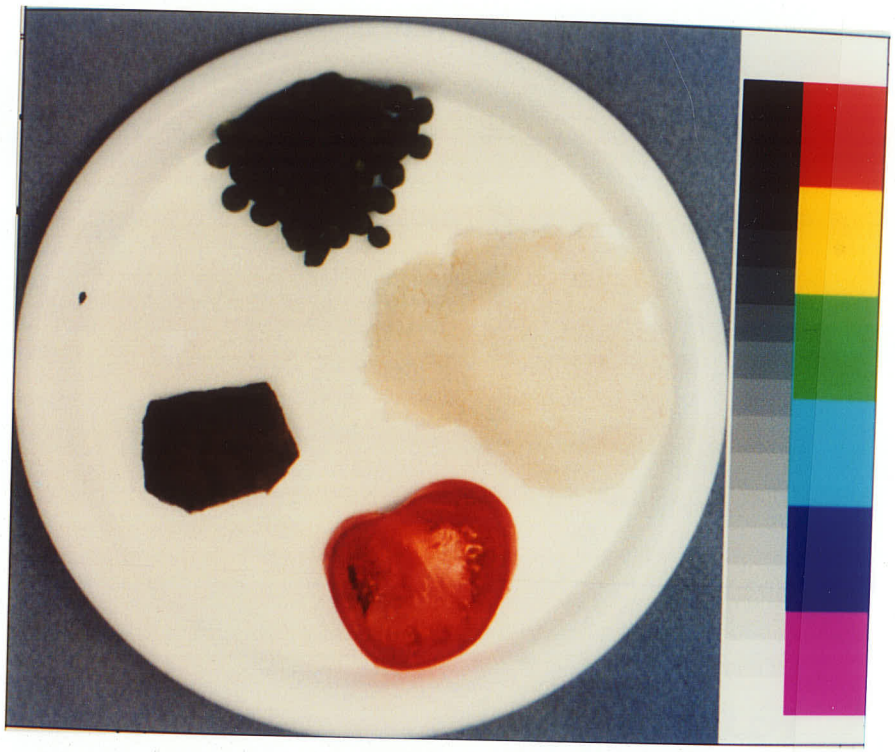


(a) The Image

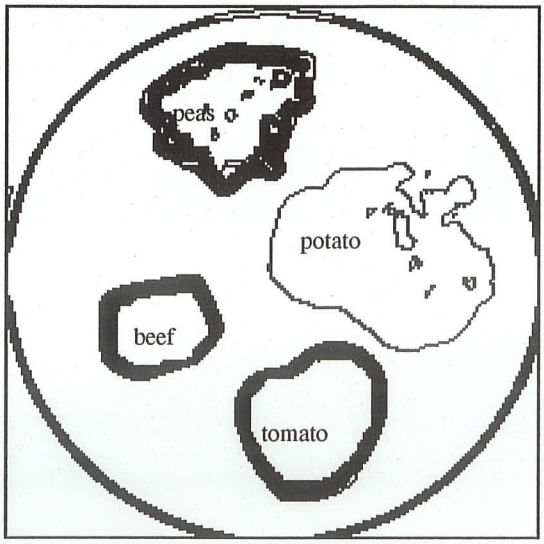


(b) The Segments

(IV)

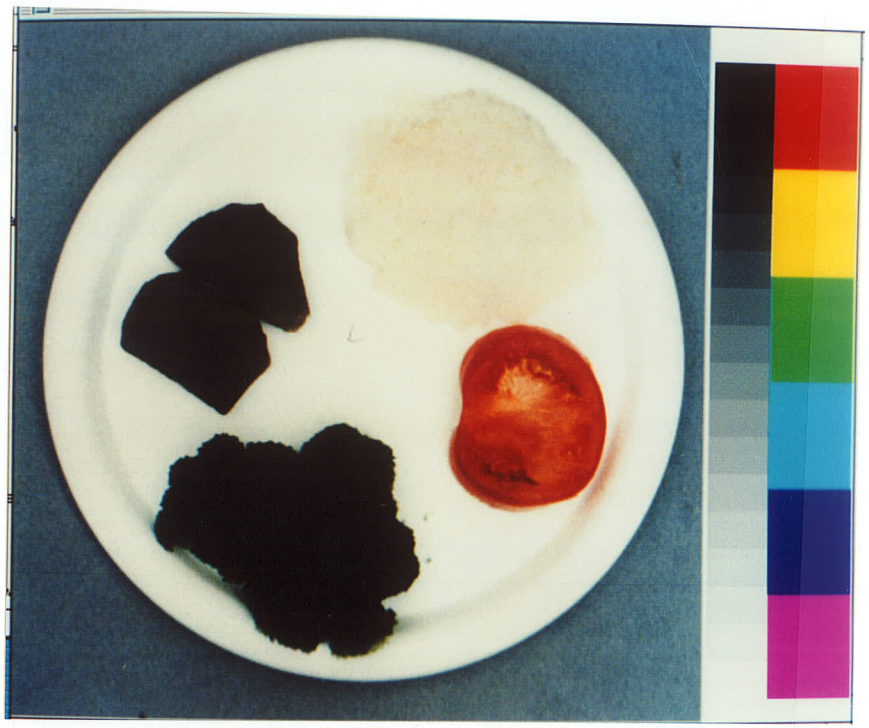


(a) The Image

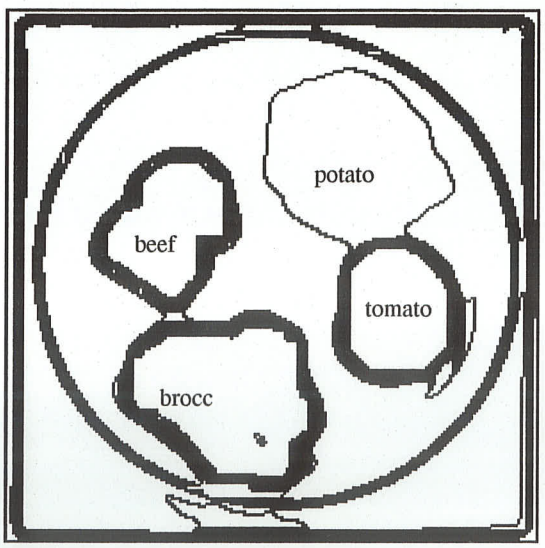


(b) The Segments

(V)



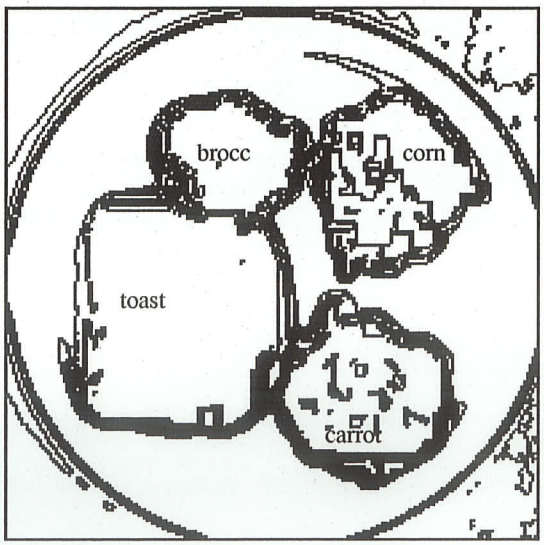
(a) The Image



(b) The Segments

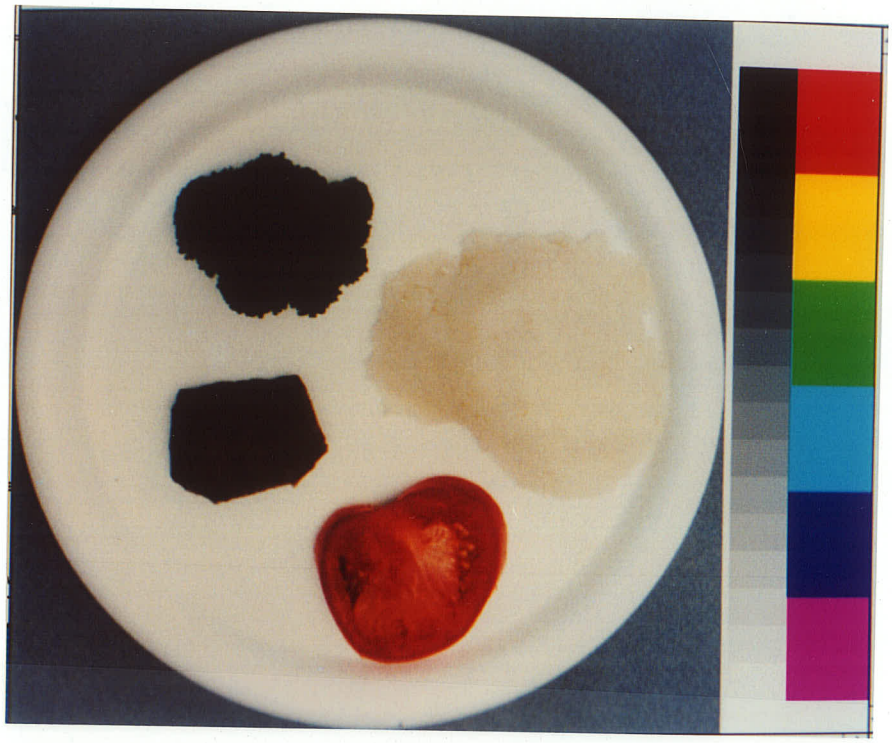
(VI)

(a) The Image

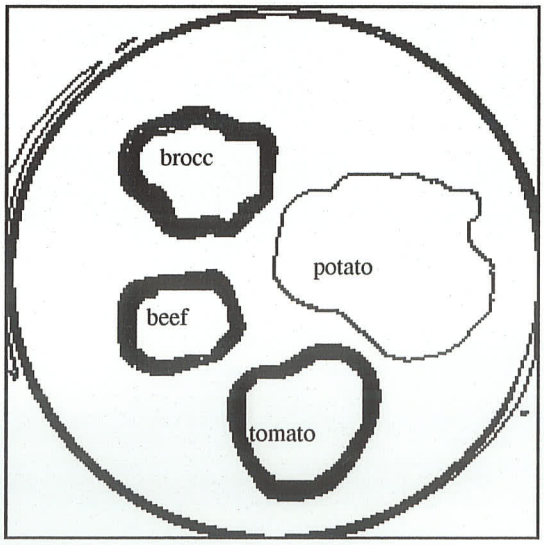


(b) The Segments

(VII)

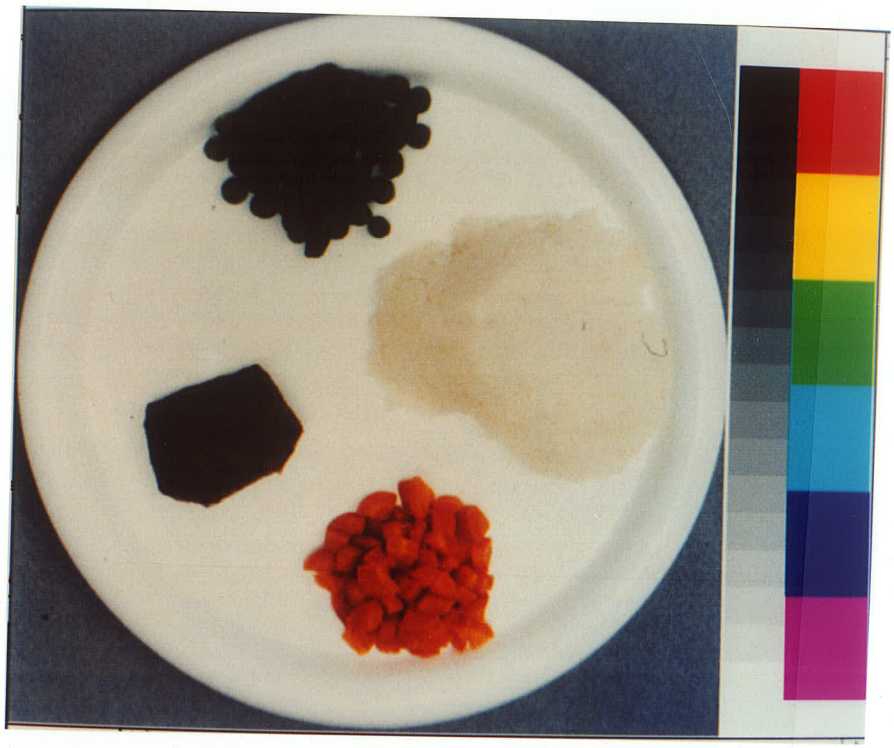


(a) The Image

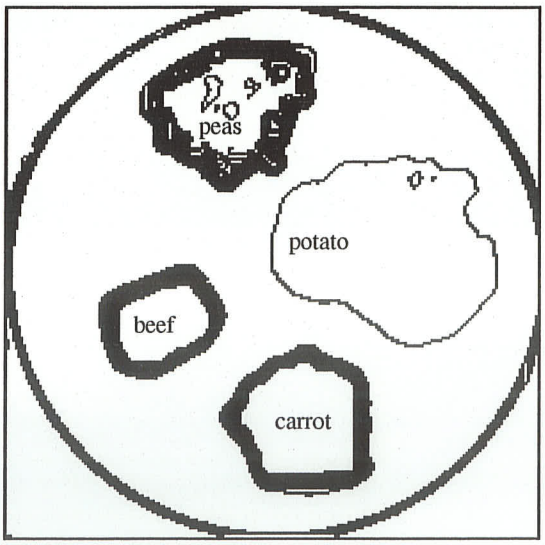


(b) The Segments

(VIII)

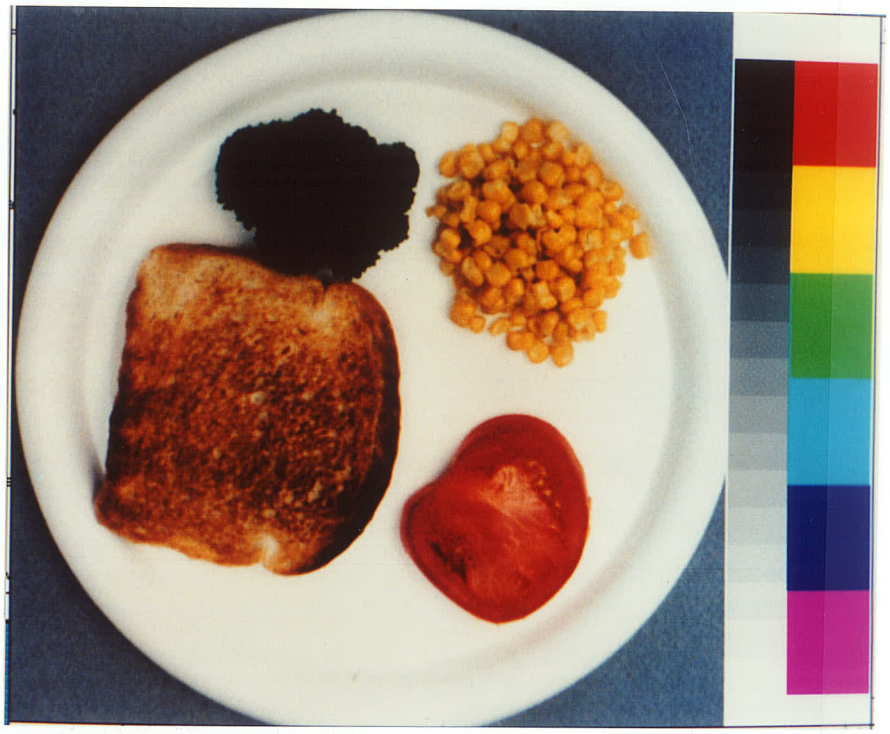


(a) The Image

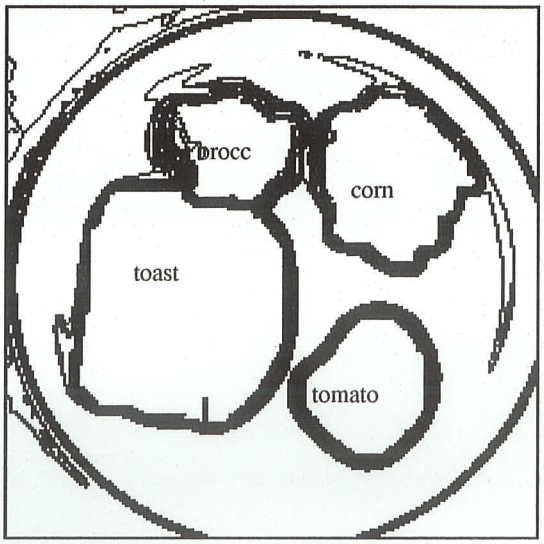


(b) The Segments

(IX)



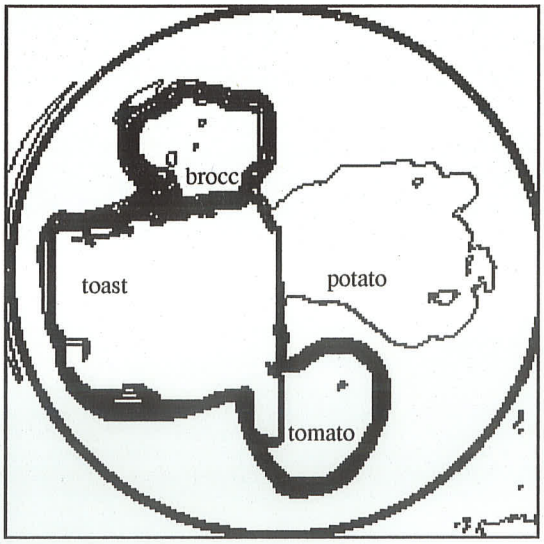
(a) The Image



(b) The Segments

(X)

(a) The Image



(b) The Segmentation

(XI)

APPENDIX C.1 PROGRAM COMP4.C

```
/*****************************************************************************
*
*                                    COMP4.C
*
* Get R, G, B values of an image from the non–interleaved raw file.
* Compress the image into a quarter of the original size by calculating the average R, G, B values of 4 (2 × 2)
* connected pixels.
* The horizonal size and vertical size of the image should be the same.
* This is a program for Macintosh II
*
*                                    By Ling Chen
*                                       1994
*
*****************************************************************************/

#include <math.h>
#include <stdio.h>

#define HEADER_SIZE                512
#define BASE_RES_ID                400
#define NIL_POINTER                0L
#define MOVE_TO_FRONT              –1L
#define REMOVE_ALL_EVENTS          0

#define ERROR_ALERT_ID             BASE_RES_ID+1
#define CANT_OPEN_FILE             BASE_RES_ID
#define GET_EOF_ERROR              BASE_RES_ID+1
#define HEADER_TOO_SMALL           BASE_RES_ID+2
#define OUT_OF_MEMORY              BASE_RES_ID+3
#define CANT_READ_HEADER           BASE_RES_ID+4
#define CANT_READ_PICT             BASE_RES_ID+5
#define CANT_WRITE_HEADER          BASE_RES_ID+6
#define CANT_WRITE_PICT            BASE_RES_ID+7
#define CANT_CLOSE_FILE            BASE_RES_ID+8
#define NOTHING_WRONG              BASE_RES_ID+9
#define CANT_GET_F_INFO            BASE_RES_ID+10
#define CANT_CREATE_FILE           BASE_RES_ID+11

#define NIL_PRPORT                 NIL_POINTER
#define NIL_IOBUFFER               NIL_POINTER
#define NIL_DEVBUF                 NIL_POINTER

#define NIL_STRING                 "\P "
#define IGNORED_STRING             NIL_STRING
#define NIL_FILE_FILTER            NIL_POINTER
```

```
#define NIL_DIALOG_HOOK                 NIL_POINTER
#define DONT_SCALE_OUTPUT               NIL_POINTER
#define HOPELESSLY_FATAL_ERROR          "\PGame over, man!"


void                    ToolBoxInit();
void                    WindowInit();
void                    GetRAWName()
void                    PrintRAWFile();
void                    ErrorHandler();

PicHandle               newPICTHand;
PicHandle               gThePicture;
WindowPtr               gPictureWindow;
short int               globalRef;
int                     PICTCount;


/*****************************************************************************/

main()
{
        SFReply                 reply;

        ToolBoxInit();
        WindowInit();
        GetRAWName( &reply );

        if ( reply.good )       /* The User didn't hit Cancel when asked for a file name */
        {
                PrintRAWFile( &reply);

                MoveTo(320,360);
                DrawString("\p press button!");

                while(!Button());
        }
}       /*************************endofmain()*********************************/


/*-------------------------------------------------------------------------------*/

ToolBoxInit()           /*—— initial the ToolBox ——*/
{
        InitGraf (&thePort );
        InitFonts();
        FlushEvents( everyEvent, REMOVE_ALL_EVENTS );
        InitWindows();
        InitMenus();
        TEInit();
        InitDialogs( NIL_POINTER );
        InitCursor();
```

```
}           /*------------------------ end of ToolBoxInit() ------------------------*/

/*::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/

WindowInit()            /*:::: initial  and open the window ::::*/
{
  gPictureWindow=GetNewWindow( BASE_RES_ID, NIL_POINTER, (WindowPtr)MOVE_TO_FRONT );
    ShowWindow( gPictureWindow );
    SetPort( gPictureWindow );
}           /*:::::::::::::::::::::::::::::::::::::::::::::::::: end of WindowInit() ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/


/*=============================================================================================*/

GetRAWName( replyPtr )          /*==== choose an image raw file ====*/
SFReply                 *replyPtr;
{
            Point               myPoint;
            SFTypeList          typeList;
            int                 numTypes;


            myPoint.h=100;
            myPoint.v=100;
            typeList[0] = 'RRRR';
            numTypes = 1;
            SFGetFile( myPoint,IGNORED_STRING,NIL_FILE_FILTER,numTypes,typeList,
                        NIL_DIALOG_HOOK,replyPtr );
}           /*============================== end of GetRAWName() ================*/


/*********************************************************************************************/

void PrintRAWFile( replyPtr)        /** get the R, G, B values; compress the image; save the new image **/
SFReply                             *replyPtr;
{
            short int               temcolor,srcFile,newsrcFile,vrefnum;
            long                    pictSize,n,new,npictSize,nlongcount,longcount;
            long                    i,j,m, k,cc,vv,hh,hsize,vsize,nhsize,nvsize;
            RGBColor                cPix;
            unsigned char           tem,*red,*green,*blue,*s1, *s2;
            SFReply                 *replyPtr2;
            OSErr                   myErr;
            Point                   myPoint;


            if ( FSOpen( (*replyPtr).fName, (*replyPtr).vRefNum, &srcFile ) !=noErr )
            {
                        FSClose( srcFile );
                        ErrorHandler( CANT_OPEN_FILE);
            }
```

```
if ( GetEOF( srcFile, &pictSize ) !=noErr )
{
        FSClose( srcFile );
        ErrorHandler( GET_EOF_ERROR);
}

n=pictSize/3;
hsize=(long) sqrt(n);
vsize=(long) sqrt(n);

if ( ( red=(unsigned char*)NewPtr(n) )==NIL_POINTER )
{
        FSClose( srcFile );
        ErrorHandler( OUT_OF_MEMORY);
}

if ( ( green=(unsigned char*)NewPtr(n) )==NIL_POINTER )
{
        FSClose( srcFile );
        ErrorHandler( OUT_OF_MEMORY);
}

if ( ( blue=(unsigned char*)NewPtr(n) )==NIL_POINTER )
{
        FSClose( srcFile );
        ErrorHandler( OUT_OF_MEMORY);
}

longcount=n*sizeof(unsigned char);


        if ( FSRead( srcFile, &longcount, red)!=noErr)
        {
                FSClose( srcFile );
                ErrorHandler( CANT_READ_PICT);
        }

        if ( FSRead( srcFile, &longcount, green)!=noErr)
        {
                FSClose( srcFile );
                ErrorHandler( CANT_READ_PICT);
        }

        if ( FSRead( srcFile, &longcount, blue)!=noErr)
        {
                FSClose( srcFile );
                ErrorHandler( CANT_READ_PICT);
        }

FSClose( srcFile );
```

```
nhsize=hsize/2;
nvsize=vsize/2;
new=n/4;
npictSize=new*3;
nlongcount=new*sizeof(unsigned char);

for ( i=0; i<nvsize; i++)    /***** start to compress *****/
{
   for ( j=0; j<nhsize; j++)
   {
            vv=2*i;
            hh=2*j;

            red[i*nhsize+j]=(red[vv*hsize+hh]+red[(vv+1)*hsize+hh]+red[vv*hsize+hh+1]
                     +red[(vv+1)*hsize+hh+1])/4.;
            green[i*nhsize+j]=(green[vv*hsize+hh]+green[(vv+1)*hsize+hh]
                     +green[vv*hsize+hh+1]+green[(vv+1)*hsize+hh+1])/4.;
            blue[i*nhsize+j]=(blue[vv*hsize+hh]+blue[(vv+1)*hsize+hh]
                     +blue[vv*hsize+hh+1]+blue[(vv+1)*hsize+hh+1])/4.;
   }
}

MoveTo(2,15);
DrawString("\p Compression is done! ");          /**** end of compression ****/


myPoint.h=200;          /***** start to save *****/
myPoint.v=200;
SFPutFile( myPoint,"\psave the file as: ",(*replyPtr).fName,NIL_DIALOG_HOOK,replyPtr2 );

if ((*replyPtr2).good )     /***** The User didn't hit Cancel *****/
{
         if(Create((*replyPtr2).fName, (*replyPtr2).vRefNum, '8BIM', 'RRRR')!=noErr)
                  ErrorHandler( CANT_CREATE_FILE);

         if ( FSOpen( (*replyPtr2).fName, (*replyPtr2).vRefNum, &newsrcFile )!=noErr )
         {
                  FSClose( newsrcFile );
                  ErrorHandler( CANT_OPEN_FILE);
         }

         if ( FSWrite( newsrcFile, &nlongcount, red)!=noErr)
         {
                  FSClose( newsrcFile );
                  ErrorHandler( CANT_WRITE_PICT);
         }

         if ( FSWrite( newsrcFile, &nlongcount, green)!=noErr)
         {
                  FSClose( newsrcFile );
                  ErrorHandler( CANT_WRITE_PICT);
         }
```

```
                    if ( FSWrite( newsrcFile, &nlongcount, blue)!=noErr)
                    {
                            FSClose( newsrcFile );
                            ErrorHandler( CANT_WRITE_PICT);
                    }

                    if ( SetEOF( newsrcFile, npictSize)!=noErr)
                    {
                            FSClose( newsrcFile );
                            ErrorHandler( GET_EOF_ERROR);
                    }

        FSClose( newsrcFile );
        }       /**** end of saving ****/

}       /********************************* end of PrintRAWFile() **********************/

/*-------------------------------------------------------------------------------------*/

void ErrorHandler( stringNum )          /*------- give the warning message and exit the procedure -------*/
int                                     stringNum;
{

        StringHandle errorStringH;

        if ( (errorStringH = GetString( stringNum )) == NIL_POINTER )
         ParamText( HOPELESSLY_FATAL_ERROR, NIL_STRING, NIL_STRING, NIL_STRING);

        else
        {
                HLock((Handle)errorStringH);
                ParamText( *errorStringH, NIL_STRING, NIL_STRING, NIL_STRING );
                HUnlock((Handle) errorStringH );
        }

        StopAlert( ERROR_ALERT_ID, NIL_POINTER );

        ExitToShell();
}       /*------------------------------------ end of ErrorHandler ------------------------------------*/

                                /**** END ****/
```

APPENDIX C.2 PROGRAM SEG–SAVE–CLASS.C

```
/*****************************************************************************
*
*                                SEG–SAVE–CLASS.C
*
* Get an image from a 'PICT' file and display it
* Read H, S, B data from the noninterleaved raw file
* Segment the image according to the algorithm and the thresholds
* Save the result of the segmentation
* Classify and identify every segment
*
*                                By Ling Chen
*                                   1994
*****************************************************************************/

#include <math.h>

#define HEADER_SIZE                512
#define BASE_RES_ID                400
#define NIL_POINTER                0L
#define MOVE_TO_FRONT              -1L
#define REMOVE_ALL_EVENTS          0

#define ERROR_ALERT_ID             BASE_RES_ID+1
#define CANT_OPEN_FILE             BASE_RES_ID
#define GET_EOF_ERROR              BASE_RES_ID+1
#define HEADER_TOO_SMALL           BASE_RES_ID+2
#define OUT_OF_MEMORY              BASE_RES_ID+3
#define CANT_READ_HEADER           BASE_RES_ID+4
#define CANT_READ_PICT             BASE_RES_ID+5
#define CANT_WRITE_HEADER          BASE_RES_ID+6
#define CANT_WRITE_PICT            BASE_RES_ID+7
#define CANT_CLOSE_FILE            BASE_RES_ID+8
#define NOTHING_WRONG              BASE_RES_ID+9
#define NEWHAND_WRONG_FIRST        BASE_RES_ID+10
#define CANT_CREATE_FILE           BASE_RES_ID+11

#define NIL_PRPORT                 NIL_POINTER
#define NIL_IOBUFFER               NIL_POINTER
#define NIL_DEVBUF                 NIL_POINTER

#define NIL_STRING                 "\P"
#define IGNORED_STRING             NIL_STRING
#define NIL_FILE_FILTER            NIL_POINTER
#define NIL_DIALOG_HOOK            NIL_POINTER
#define DONT_SCALE_OUTPUT          NIL_POINTER
```

```
#define HOPELESSLY_FATAL_ERROR          "\PGame over, man!"

void        ToolBoxInit();
void        WindowInit();
Rect        PrintPICTFile();
void        PrintRAWFile();
void        GetPICTName();
void        GetRAWName();
void        ErrorHandler();

PicHandle   newPICTHand;
PicHandle   gThePicture;
WindowPtr   gPictureWindow;
short int   globalRef;
int         PICTCount;

/*****************************************************************************/

main()
{
            SFReply                 reply1, reply2;
            PicHandle               PICTHand;
            PicHandle               newHand;
            Rect                    location;

            ToolBoxInit();
            WindowInit();

            GetPICTName( &reply1 );
            GetRAWName( &reply2);

            if ( reply1.good )              /**** The User didn't hit Cancel ****/
            {
                        PrintPICTFile( &reply1);
            }
            if ( reply2.good )              /**** The User didn't hit Cancel ****/
            {
                        PrintRAWFile( &reply2);
                        MoveTo(500,400);
                        DrawString("\ppress button");
                        while(!Button());
            }
}           /****************************** end of main() ************************/

/*-------------------------------------------------------------------------------*/

ToolBoxInit()   /*—— initiate the ToolBox ——*/
{
            InitGraf (&thePort );
```

```
        InitFonts();
        FlushEvents( everyEvent, REMOVE_ALL_EVENTS );
        InitWindows();
        InitMenus();
        TEInit();
        InitDialogs( NIL_POINTER );
        InitCursor();
}       /*------------------------------end of ToolBoxInit()----------------------*/

/*=========================================================================*/

WindowInit()    /*==== initiate and open the window ====*/
{
 gPictureWindow=GetNewWindow( BASE_RES_ID, NIL_POINTER, (WindowPtr)MOVE_TO_FRONT );
  ShowWindow( gPictureWindow );
  SetPort( gPictureWindow );
}       /*======================= end of WindowInit() =======================*/

/*.........................................................................*/

void GetPICTName( replyPtr )    /*.... choose an image name ....*/
SFReply                         *replyPtr;
{
        Point                   myPoint;
        SFTypeList              typeList;
        int                     numTypes;

        myPoint.h=100;
        myPoint.v=100;
        typeList[0] = 'PICT';
        numTypes = 1;
        SFGetFile( myPoint,IGNORED_STRING,NIL_FILE_FILTER,numTypes,typeList,
                        NIL_DIALOG_HOOK,replyPtr );
}/*...........................end of GetPICTName...........................*/

/*:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/

Rect PrintPICTFile( replyPtr)   /*:::: get the image from the 'PICT' file and display it ::::*/
SFReply         *replyPtr;
{
        PicHandle               thePict;
        short int               srcFile;
        char                    pictHeader[ HEADER_SIZE ];
        long                    pictSize, headerSize;
        unsigned char           *s1, *s2, *s3, *s4;

        if ( FSOpen( (*replyPtr).fName, (*replyPtr).vRefNum, &srcFile ) !=noErr )
        {
                FSClose( srcFile );
```

```
                    ErrorHandler( CANT_OPEN_FILE );
        }
        if ( GetEOF( srcFile, &pictSize ) !=noErr )
        {
                    FSClose( srcFile );
                    ErrorHandler( GET_EOF_ERROR);
        }
        headerSize=HEADER_SIZE;
        if ( FSRead( srcFile, &headerSize, pictHeader ) !=noErr )
        {
                    FSClose( srcFile );
                    ErrorHandler( CANT_READ_HEADER);
        }
        if ( ( pictSize -=HEADER_SIZE )<=0)
        {
                    FSClose( srcFile );
                    ErrorHandler( HEADER_TOO_SMALL);
        }
        if ( ( thePict=(PicHandle)NewHandle( pictSize) )==NIL_POINTER )
        {
                    FSClose( srcFile );
                    ErrorHandler( OUT_OF_MEMORY);
        }
        HLock((Handle)thePict);
        if ( FSRead( srcFile, &pictSize, *thePict )!=noErr)
        {
                    FSClose( srcFile );
                    ErrorHandler( CANT_READ_PICT);
        }
        FSClose( srcFile );
        DrawPicture( thePict, &(**(thePict )).picFrame );
        HUnlock((Handle)thePict);
        DisposeHandle((Handle)thePict);
        return;
}       /*::::::::::::::::::::::::::::::::::::::::::::::::: end of PrintPICTFile() ::::::::::::::::::::::::::::::::::::::::::::::::::::::/

/*-------------------------------------------------------------------------------------------------------------------------------*/
void GetRAWName( replyPtr )     /*------ choose an image HSB raw file ------*/
SFReply                         *replyPtr;
{
        Point                   myPoint;
        SFTypeList              typeList;
        int                     numTypes;
```

```
                myPoint.h=100;
                myPoint.v=100;
                typeList[0] = 'HHHH';
                numTypes = 1;
                SFGetFile( myPoint,IGNORED_STRING,NIL_FILE_FILTER,numTypes,typeList,
                                NIL_DIALOG_HOOK,replyPtr );
}               /*-------------------------------endofGetRAWName()-------------------------*/

/*****************************************************************************************/

void PrintRAWFile( replyPtr)    /** get the H, S, B values; segment the image; save the result; classify **/
SFReply         *replyPtr;
{
                short int       srcFile, newsrcFile, *charact, vrefnum;
                long            pictSize,longcount,nn,m,k,i,j,n, averh, avers, averb, *sign, reg;
                long            hsize,vsize, Hthreshold, Sthreshold, Bthreshold, totalh, totals, totalb;
                RGBColor        cPix;
                unsigned char   *hue, *satu, *bright, *seg, *s1, *s2, max,tem,above;
                SFReply         *replyPtr2;
                Point           myPoint;

                if ( FSOpen( (*replyPtr).fName, (*replyPtr).vRefNum, &srcFile ) !=noErr )
                {
                                FSClose( srcFile );
                                ErrorHandler( CANT_OPEN_FILE);
                }

                if ( GetEOF( srcFile, &pictSize ) !=noErr )
                {
                                FSClose( srcFile );
                                ErrorHandler( GET_EOF_ERROR);
                }               n=pictSize/3;


                hsize=sqrt(n);
                vsize=sqrt(n);


                if ( ( hue=(unsigned char*)NewPtr(n) )==NIL_POINTER )
                {
                                FSClose( srcFile );
                                ErrorHandler( OUT_OF_MEMORY);
                }

                if ( ( satu=(unsigned char*)NewPtr(n) )==NIL_POINTER )
                {
                                FSClose( srcFile );
                                ErrorHandler( OUT_OF_MEMORY);
                }
```

```
        if ( ( bright=(unsigned char*)NewPtr(n) )==NIL_POINTER )
        {
                        FSClose( srcFile );
                        ErrorHandler( OUT_OF_MEMORY);
        }
        m=n*sizeof(short int);
        if ( ( charact=(short int*)NewPtr(m) )==NIL_POINTER )
        {
                        FSClose( srcFile );
                        ErrorHandler( OUT_OF_MEMORY);
        }
        m=2*hsize*sizeof(long);
        if ( ( sign=(long*)NewPtr(m) )==NIL_POINTER )
        {
                        FSClose( srcFile );
                        ErrorHandler( OUT_OF_MEMORY);
        }
        longcount=n*sizeof(unsigned char);
        if ( FSRead( srcFile, &longcount, hue)!=noErr)
        {
                        FSClose( srcFile );
                        ErrorHandler( CANT_READ_PICT);
        }
        if ( FSRead( srcFile, &longcount, satu)!=noErr)
        {
                        FSClose( srcFile );
                        ErrorHandler( CANT_READ_PICT);
        }
        if ( FSRead( srcFile, &longcount, bright)!=noErr)
        {
                        FSClose( srcFile );
                        ErrorHandler( CANT_READ_PICT);
        }
        FSClose( srcFile );

for (i=0; i<2*hsize; i++)               /*** start to pick up the background and the plate ***/
                sign[i]=0;
for (i=0; i<vsize; i++)
{
    for (j=0; j<hsize; j++)
    {
        if ( (hue[i*hsize+j]>100) && (hue[i*hsize+j]<230) &&(satu[i*hsize+j]<50)
                        && (bright[i*hsize+j]>90) && (bright[i*hsize+j]<140 ) )
```

```
        {
                        charact[i*hsize+j]=0;
                        sign[0]++;
        }
        else
        {
            if ( (satu[i*hsize+j]<35) && (bright[i*hsize+j]>200) )
            {
                        charact[i*hsize+j]=1;
                        sign[1]++;
            }
            else
            {
                        charact[i*hsize+j]=2;
            }
        }
    }
}               /**** end of pick up ****/

max=2;          /**** start to segment ****/
Hthreshold=10;
Sthreshold=15;
Bthreshold=20;
for (j=1; j<hsize; j++)
{
    if ( (charact[j]!=0) && (charact[j]!=1) )
    {
        if(  (abs(hue[j]-hue[j-1]) < Hthreshold) && (abs(satu[j]-satu[j-1]) < Sthreshold) &&
        (abs(bright[j]-bright[j-1]) < Bthreshold) && (charact[j-1]!=0) && (charact[j-1]!=1) )
                        charact[j]=charact[j-1];
        else
        {
                        charact[j]=max;
                        max++;
        }
        sign[charact[j]]++;
    }
}

for (i=1; i<vsize; i++)
{
```

```
if ( (charact[i*hsize]!=0) && (charact[i*hsize]!=1) )
{
    if( ( abs(hue[i*hsize]–hue[(i–1)*hsize]) < Hthreshold )
                    && (abs(satu[i*hsize]–satu[(i–1)*hsize]) < Sthreshold)
                    && (abs(bright[i*hsize]–bright[(i–1)*hsize]) < Bthreshold)
                    && (charact[(i–1)*hsize]!=0) && (charact[(i–1)*hsize]!=1) )

                    charact[i*hsize]=charact[(i–1)*hsize];
    else
    {
                    charact[i*hsize]=max;
                    max++;
    }
    sign[charact[i*hsize]]++;
}
for (j=1; j<hsize; j++)
{
    if ( (charact[i*hsize+j]!=0) && (charact[i*hsize+j]!=1) )
    {
        if ( (abs(hue[i*hsize+j]–hue[i*hsize+j–1]) <Hthreshold )
                        && (abs(satu[i*hsize+j]–satu[i*hsize+j–1]) < Sthreshold)
                        && (abs(bright[i*hsize+j]–bright[i*hsize+j–1]) < Bthreshold)
                        && (charact[i*hsize+j–1]!=0) && (charact[i*hsize+j–1]!=1) )
        {
            if ( (abs(hue[i*hsize+j]–hue[(i–1)*hsize+j]) <Hthreshold )
                            && (abs(satu[i*hsize+j]–satu[(i–1)*hsize+j]) < Sthreshold)
                            && (abs(bright[i*hsize+j]–bright[(i–1)*hsize+j]) < Bthreshold)
                            && (charact[(i–1)*hsize+j]!=0) && (charact[(i–1)*hsize+j]!=1) )
            {
                if ( charact[i*hsize+j–1]==charact[(i–1)*hsize+j] )
                {
                                charact[i*hsize+j]=charact[i*hsize+j–1];
                                sign[charact[i*hsize+j]]++;
                }
                else
                {
                                tem=charact[i*hsize+j–1];
                                above=charact[(i–1)*hsize+j];
                                charact[i*hsize+j]=above;
                                sign[above]++;
                    if( (tem!=0) && (tem!=1) )
                    {
                                    sign[tem]=0;
```

```
                for (m=0; m<i; m++)
                {
                    for (k=0; k<hsize; k++)
                    {
                        if (charact[m*hsize+k]==tem)
                        {
                                        charact[m*hsize+k]=above;
                                        sign[above]++;
                        }
                    }
                }
                for (k=0; k<j; k++)
                {
                    if ( charact[i*hsize+k]==tem )
                    {
                                    charact[i*hsize+k]=above;
                                    sign[above]++;
                    }
                }
            }
        }
        else
        {
                    charact[i*hsize+j]=charact[i*hsize+j-1];
                    sign[charact[i*hsize+j]]++;
        }
    }

    else
    {
        if ( ( abs(hue[i*hsize+j]-hue[(i-1)*hsize+j]) <Hthreshold )
                    && (abs(satu[i*hsize+j]-satu[(i-1)*hsize+j]) < Sthreshold)
                    &&(abs(bright[i*hsize+j]-bright[(i-1)*hsize+j])<Bthreshold)
                    && (charact[(i-1)*hsize+j]!=0)
                    && (charact[(i-1)*hsize+j]!=1) )
                    charact[i*hsize+j]=charact[(i-1)*hsize+j];
        else
        {
                    charact[i*hsize+j]=max;
                    max++;
        }
        sign[charact[i*hsize+j]]++;
```

```
                }
            }
        }
    }
}/********** end of the segmentation ************/

/********** start to draw the edge **************/

if ( ( seg=(unsigned char*)NewPtr(n) )==NIL_POINTER )
                ErrorHandler( OUT_OF_MEMORY);

for (i=0; i<n; i++)
                seg[i]=255;

for ( i=0; i<vsize; i++)
{
                seg[i*hsize]=0;
                seg[(i+1)*hsize-1]=0;
}

for (j=0; j<hsize; j++)
{
                seg[j]=0;
                seg[(vsize-1)*hsize+j]=0;
}

MoveTo(250, 0);
LineTo(450, 0);
LineTo(450, 200);
LineTo(250, 200);
LineTo(250, 0);

cPix.red=0;
cPix.green=0;
cPix.blue=65055;

for (i=0; i<vsize; i++)
{
    for (j=0; j<hsize; j++)
    {
                    nn=i*hsize+j;
        if ( (charact[nn]!=charact[nn-1]) || (charact[nn]!= charact[(i-1)*hsize+j]) )
        {
                        seg[nn]=0;
```

```
                    SetCPixel( j+250, i, &cPix);
        }
    }
}                   /******** end of the drawing *********/

DisposePtr( (Ptr) seg );


myPoint.h=10;    /**** start of saving ****/
myPoint.v=210;
SFPutFile( myPoint,"\psave the file as: ",(*replyPtr).fName,NIL_DIALOG_HOOK,replyPtr2 );

if ((*replyPtr2).good )              /**** The User didn't hit Cancel ****/
{
    if (Create((*replyPtr2).fName, (*replyPtr2).vRefNum, '8BIM', 'WWWW')!=noErr)
                    ErrorHandler( CANT_CREATE_FILE);
    if ( FSOpen( (*replyPtr2).fName, (*replyPtr2).vRefNum, &newsrcFile )!=noErr )
    {
                    FSClose( newsrcFile );
                    ErrorHandler( CANT_OPEN_FILE);
    }

    if ( FSWrite( newsrcFile, &longcount, seg)!=noErr)
    {
                    FSClose( newsrcFile );
                    ErrorHandler( CANT_WRITE_PICT);
    }

    if ( SetEOF( newsrcFile,longcount)!=noErr)
    {
                    FSClose( newsrcFile );
                    ErrorHandler( GET_EOF_ERROR);
    }

    FSClose( newsrcFile );
}                   /**** end of saving ****/

tem=0;              /******* start classification *********/
for (i=0; i<vsize; i++)
{
    for (j=0; j<hsize; j++)
    {
                    reg=0;
                    totalh=0;
                    totals=0;
```

```
                totalb=0;
                nn=charact[i*hsize+j];
if ( sign[nn]>100 )
{
    for ( m=i; m<vsize; m++ )
    {
        for ( k=0; k<hsize; k++)
        {
            if (charact[m*hsize+k]==nn )
            {
                                totalh+=hue[m*hsize+k];
                                totals+=satu[m*hsize+k];
                                reg++;
            }
        }
    }


    MoveTo(5,220+15*tem);

    averh=totalh/reg;
    avers=totals/reg;


    if ( (averh>=3) && (averh<=7) && (avers>=210) && (avers<=225) )
                        DrawString("\p Tomato!");
    else
    {
        if ( (averh>=6) && (averh<=9) && (avers>=230) && (avers<=250) )
                            DrawString("\p Carrot!");
        else
        {
            if ( (averh>=12) && (averh<=18) && (avers>=225) && (avers<=240) )
                                DrawString("\p Orange!");
            else
            {
                if ( (averh>=45) && (averh<=50) && (avers>=180) && (avers<=195) )
                                    DrawString("\p Broccoli!");
                else
                {
                    if ( (averh>=52) && (averh<=74) && (avers>=115) && (avers<=240) )
                                        DrawString("\p Green Peas!");
                    else
                    {
```

```
                                if ( (averh>=29) && (averh<=38) && (avers>=40) && (avers<=290) )
                                        DrawString("\p Potato!");
                                else
                                {
                                    if ( (averh>=24) && (averh<=28) && (avers>=215)
                                                && (avers<=230) )
                                                DrawString("\p Corn!");
                                    else
                                    {
                                        if ( (averh>=11) && (averh<=17) && (avers>=140)
                                                    && (avers<=180) )
                                                    DrawString("\p Beef!");
                                        else
                                        {
                                            if ( (averh>=14) && (averh<=19) && (avers>=185)
                                                        && (avers<=220) )
                                                        DrawString("\p Toast Bread!");
                                            else
                                            {
                                                        DrawString("\p Something Else!");
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }

                tem++;
                sign[nn]=0;
            }
        }
    }
    DisposePtr( (Ptr) hue );
    DisposePtr( (Ptr) satu );
    DisposePtr( (Ptr) bright );


}               /********************** end of GetRAWName() ****************************/
```

```
/*****************************************************************************/
void ErrorHandler( stringNum )     /***** give the warning message and exit the procedure *****/
int                stringNum;
{
                   StringHandle errorStringH;

        if ( (errorStringH = GetString( stringNum )) == NIL_POINTER )
                           ParamText( HOPELESSLY_FATAL_ERROR, NIL_STRING, NIL_STRING,
                           NIL_STRING);
        else
        {
                           HLock((Handle)errorStringH);
                           ParamText( *errorStringH, NIL_STRING, NIL_STRING, NIL_STRING );
                           HUnlock((Handle) errorStringH );
        }

        StopAlert( ERROR_ALERT_ID, NIL_POINTER );
        ExitToShell();

}                  /***************************** end of ErrorHandler ************************/


                                   /**** END ****/
```