Tracking Color Objects in Real Time

by

Vladimir Kravtchenko

Diploma in Computer Engineering, I.M. Gubkin State Oil & Gas Academy, 1992

A thesis submitted in partial fulfilment of
the requirements for the degree of

Master of Science

in

The Faculty of Graduate Studies
Department of Computer Science

We accept this thesis as conforming
to the requested standard

. . . . . .

. . . . . .

The University of British Columbia

August 27, 1999

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purpose may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis in whole or in part for financial gain shall not be allowed without my written permission.

Thesis Title: Tracking Color Objects in Real Time

Author: Vladimir Kravtchenko

. . . . . .

(Signature)

. . . . . .

(Date)

Department of Computer Science
The University of British Columbia
2366 Main Mall
Vancouver, BC, Canada
V6T 1Z4

Abstract

The goal of our research is efficient tracking of color objects from a sequence of live images for use in real-time applications including surveillance, video conferencing and robot navigation. In this work we outline the results of our research. First we propose a novel, compact, look-up table color representation of a dielectric object that models the behavior of a color cluster in color space and yields real time performance in segmenting out color object pixels. This representation accounts for non-white illumination, shadows, highlights, variable viewing and camera operating conditions. We then propose a clustering method that uses density and spatial cues to cluster object pixels into separate objects. We also describe a method of identifying objects from the neighboring frames and predicting their future movement. Finally we provide details of a practical implementation of a tracking system based on the proposed techniques.

Table of Contents

List of Figures

Acknowledgments

I owe my gratitude to many people who made my studies at UBC an unforgettable experience. I extend sincere appreciation to Dr. Jim Little, my faculty supervisor, for giving me the freedom to choose a research topic, for his valuable guidance and encouragement, for reviewing my work and suggesting needed improvements. Much appreciation goes to Dr. Toshikazu Wada, Visiting Professor from Kyoto University, for reviewing my thesis and providing both positive criticism and valuable comments. I am also very grateful to Dr. David Lowe for his course that inspired me and resulted in the opportunity to do the project that laid the foundation for my Masters thesis.

Many thanks to the entire LCI community and especially to the "vision gang" - Don Murray, Cullen Jennings, Stewart Kingdon, Rod Barman, Jochen Lang - for their advice and numerous explanations, to Luc Dierckx for his technical support, and, to Ms. Valerie McRae for her endless "maternal care" which went beyond ordinary secretarial duties.

My special thanks to Ms. Mickey Soudack from SFU for her effort in improving my English style and coping with my "a" and "the".

Finally, I would like to thank my wife, Galina, for her love, patience and support throughout my studies at UBC.

Vladimir Kravtchenko

The University of British Columbia
August 1999

Chapter 1

Introduction

A variety of problems of current interest in computer vision require the ability to track moving objects in real time for purposes such as surveillance, video conferencing, robot navigation, etc. The fundamental challenges that drive much of the research in this field are the enormous data bandwidth implied by high resolution frames at high frame rates, and the desire for real-time interactive performance. Numerous innovative methods have been proposed [18, 3, 35, 19, 39, 49, 50]. However, most of these methods utilize sophisticated models such as edges, snakes, splines, templates or computationally expensive eigenimage or condensation algorithms [4, 25]. Although these approaches are broad in their abilities offering reliable object recognition in addition to tracking, they are as yet unable to run on full video resolution images at high frame rates.

Color has been widely used in real-time tracking systems [37, 38, 22, 10]. It offers several significant advantages over geometric cues such as computational simplicity, robustness under partial occlusion, rotation, scale and resolution changes. Although color methods proved to be efficient in a variety of vision applications, there are several problems associated with these methods of which color constancy is one of the most important [14, 15].

In our research we address the question of building an active tracking system consisting of the ordinary off-the-shelf equipment and capable of tracking multiple objects from an NTSC frame sequence in real time. We also address the problem of color constancy for dielectric objects and suggest a novel method for representing the color of a dielectric object for high speed image segmentation. We achieve real-time tracking performance relying on color, density and spatial cues.

In the tracking system implemented, the color *blobs* are being tracked. The notion of blobs as a representation for image features has a long history in computer vision and has had many different mathematical definitions. It may be a compact set of pixels that share a visual property that is not shared by the surrounding pixels. This property could be color, texture, brightness, motion, shading, a combination of these, or any other salient spatio-temporal property derived from the signal, in our case the image sequence. We use the dichromatic reflection properties of a dielectric color material to identify object pixels in the image (Chapter 2). Having done that, we use object pixels distribution information, namely, density and spatial cues, to cluster object pixels into separate objects/blobs (Chapter 3). Finally, we identify objects, and based on their previous history, i.e., position in the neighboring frames, predict their future movement, and, respectively, control the movement of a pan/tilt camera to track the objects (Chapter 4).

Chapter 2

Efficient Color Object Segmentation Using the Dichromatic Reflection Model

2.1 Problem

Color has been widely used in machine-based vision systems for tasks such as image segmentation, object recognition and tracking. It offers several significant advantages over geometric cues and gray scale intensity such as computational simplicity, robustness under partial occlusion, rotation in depth, scale changes and resolution changes. Although color based object segmentation methods proved to be efficient in a variety of vision applications, there are several problems associated with these methods, of which color constancy is one of the most important. A few factors that contribute to this problem include illumination changes, shadows and highlights, inter-reflection with other objects, and camera characteristics. Also, speed of segmentation may become an issue in real-time applications where use of computationally expensive operations during the segmentation process may be restrictive. The problem therefore is how to represent the object color robustly and efficiently, and provide means for a high speed segmentation.

2.2 Brief review of color models commonly used in computer vision for object segmentation

In computers, color pixels usually contain Red, Green and Blue values each measured in 8 bits. A typical color object segmentation would involve a conversion of these values to some color model

parameters, then, comparison of these parameters to the assumed object model invariants. The most popular color models used are RGB, HSV, HLS, HSI and NCC.

A *color model* is a method for explaining the properties or behavior of color within some particular context [21]. The purpose of a color model is to allow convenient specification of colors within some color gamut, where the color gamut is a subset of all visible chromaticities [11]. Hence, no single color model can explain all aspects of color or be used to specify all visible colors. We make use of different models to help describe the different perceived characteristics of color. Our primary interest is the gamut for color cameras, as defined by the Red, Green, and Blue primaries.

2.2.1 The RGB Color Model

The RGB (*Red, Green, Blue*) color model uses a cartesian coordinate system and forms a unit cube shown in Figure 1.

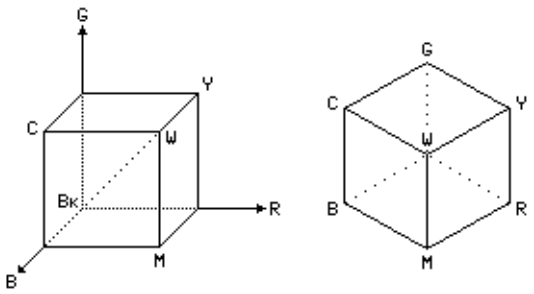

Figure 1. The RGB color model.

The dotted main diagonal of the cube, with equal amounts of *Red*, *Green* and *Blue*, represents the gray levels. This diagonal is further referred to as the *gray diagonal*. The RGB color model is hardware oriented and is used in many image capturing, processing and rendering devices.

2.2.2 The HSV Color Model

The HSV (*Hue, Saturation, Value*) color model suggested by A. Smith [43] is user oriented and is based on the intuitive appeal of the artist's tint, shade, and tone. The subspace within which the model is defined is a hexcone as shown in Figure 2.

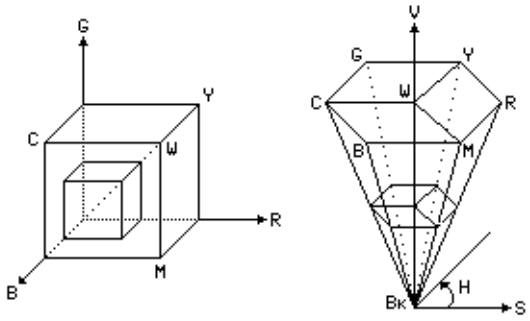

Figure 2. The HSV color model.

The top plane of the hexcone corresponds to $V = 1$, and contains the maximum intensity colors. The point at the apex is black and has the coordinate $V = 0$. Hue is the angle around the vertical (gray) axis, with *Red* at 0°. The notion of Hue is very important and is used in many other color models. Note that complementary colors are 180° opposite one another as measured by Hue. The value of S is a ratio, ranging from 0 on the center line (V axis) to 1 on the triangular sides of the hexcone. The point $S = 0$, $V = 1$ is white. When $S = 0$, H is irrelevant and is undefined.

There is a geometrical correspondence between the HSV and the RGB color models. The top of the HSV hexcone corresponds to the surface seen by looking along the gray diagonal of the RGB color cube from white toward black as shown in Figure 1. The RGB cube has subcubes as shown in Figure 2. Each subcube when viewed along its gray diagonal appears like a hexcone. Each plane of constant V in HSV space corresponds to such a view of a subcube in the RGB space. In other words, the top surface of the RGB subcube projected orthogonally along the gray diagonal onto a plane becomes a plane of a constant V in the HSV hexcone.

2.2.3 The HLS Color Model

The HLS (*Hue, Lightness, Saturation*) color model suggested by W. Ostwald [36] forms a double hexcone subspace as shown in Figure 3.

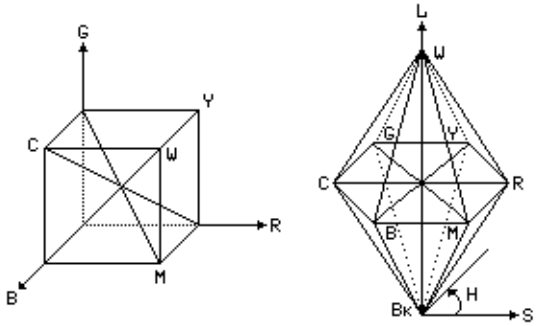

Figure 3. The HLS color model.

The definition of Hue is the same as in the HSV model. Lightness is a gray axis with black at the bottom and white at the top of the double hexcone. Lightness is 0 for black and 1 for white. In fact, one can think of HLS as a deformation of HSV, in which white is pulled upwards to form the upper hexcone. Saturation is measured radially from the vertical gray axis, from 0 on this axis to 1 on the surface of the hexcone. The maximally saturated hues are at $S = 1$, $L = 0.5$.

The geometrical correspondence between the HLS and the RGB color models is as follows. Imagine rubber bands running from the center of the RGB cube to the cube corners (*RGBCMY*), as shown in Figure 3. The $L = 0.5$ plane in the double hexcone corresponds to an umbrella-like surface formed by the rubber bands and seen by looking along the gray diagonal of the RGB color cube. We can resize the RGB cube to form subcubes in two ways: by mowing the white point to black, or moving the black point to white. In the first case, the umbrella-like surfaces obtained will form the lower HLS hexcone.

In the second case, the umbrella-like surfaces will form the upper HLS hexcone. In other words, the umbrella-like surface of the RGB subcube projected orthogonally along the gray diagonal onto a plane becomes a plane of constant *L* in the HLS double hexcone.

2.2.4 The HSI Color Model

The HSI (*Hue, Saturation, Intensity*) color model, suggested by J. Tenenbaum [46] appeals to the user's intuition and is compatible with hardware. The subspace within which the model is defined is the RGB cube as shown in Figure 4.

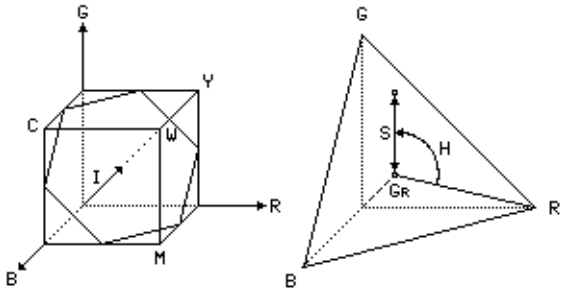

Figure 4. The HSI color model.

Hue is the same as in the HSV model. Intensity changes from 0 to 1 and is defined as $I = (R + G + B) / 3$ which corresponds to the distance along the gray diagonal scaled by a factor of $\sqrt{3}$, from the black corner of the cube to the plane in which the RGB point lies. This plane of constant intensity is perpendicular to the gray diagonal. To calculate Saturation, the *RGB* values are normalized to *rgb*. The *rgb* point lies on the $R + G + B = 1$ plane. Saturation is measured radially from the gray point in this plane, where that point is at 0 to 1 on the surface of the RGB cube as shown in Figure 4.

2.2.5 The NCC Color Model

The NCC (*Normalized Color Components*) color model is a simple model which eliminates the intensity component of color. The subspace within which the model is defined is the *rg* plane as shown in Figure 5. $r = R / (R + G + B)$, $g = G / (R + G + B)$.

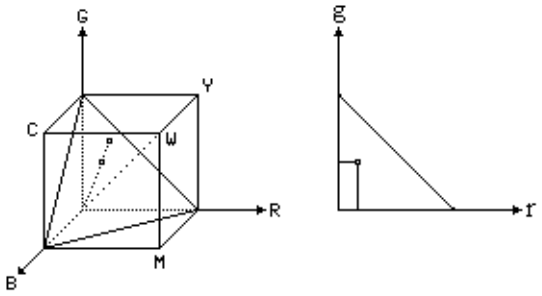

Figure 5. The NCC color model.

The RGB point is perspectively projected onto the $R + G + B = 1$ plane, then this plane is orthogonally projected along the B axis onto the RG side of the RGB cube, as shown in Figure 5.

2.2.6 The HDI Color Model

In addition to the above, the HDI (*Hue, Distance, Intensity*) color model is suggested in this paper and is used for most illustrations herein. The rationale for the suggestion is to preserve the geometrical distances in units used to measure *R*, *G* and *B* values and avoid a problem inherent to some of the above color models in describing saturated pixels. The subspace within which the model is defined is the RGB cube as shown in Figure 6.

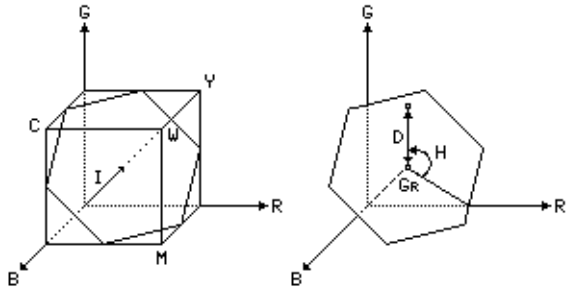

Figure 6. The HDI color model.

Hue is the same as in the HSV model. Intensity is defined as $I = (R + G + B) / \sqrt{3}$. This is the true unscaled distance along the gray diagonal from the black corner of the cube to the plane in which the RGB point lies. This plane of constant intensity is perpendicular to the gray diagonal. Distance is the distance from the RGB point to the gray diagonal. Distance is not a ratio. Distance and Intensity are measured in the same units as are *R*, *G*, and *B*.

Other color models are used in computer vision and graphics but are not reviewed herein. Instead we concentrate on the advantages and disadvantages of the above mentioned models in representing the color of a dielectric object under various illumination and viewing conditions.

2.2.7 Pixel Distribution in Real Settings

To demonstrate how image pixels of a dielectric object are distributed in the space defined by each color model, several experiments in real settings were performed. The following equipment was utilized: a Sony EVI-D30 CCD color camera providing an NTSC signal via S-Video connection, a Matrox Meteor NTSC frame grabber, and a PC with an Intel Pentium II - 233 MHz processor running Linux. Lighting was of two types, fluorescent ceiling lamps and an incandescent desk lamp.

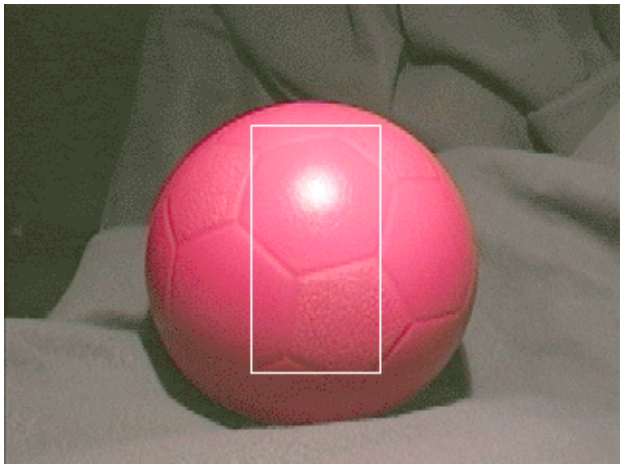

Figure 7. A sample area of the red ball.

In our first set of experiments a red ball made of a soft plastic, a Sony camera and indoor fluorescent illumination were used. Firstly, the camera operating parameters (iris, gain, and shutter) were manually adjusted to avoid saturation of pixels in R, G, and B bands. The distribution of pixels from the framed area on the ball in Figure 7 is shown in Figure 8.

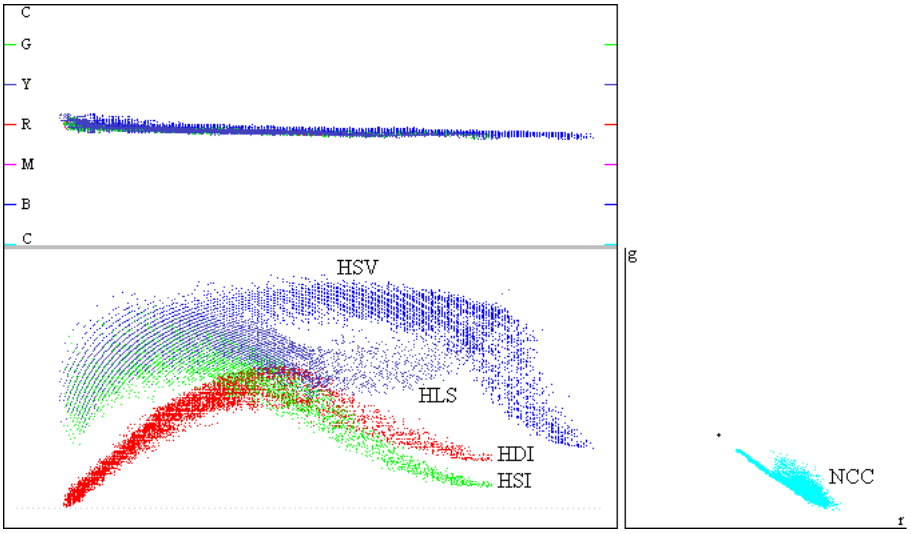

Figure 8. Red ball pixel distribution without color saturation.

The top plot is *H* vs *V* for HSV, *H* vs *L* for HLS, *H* vs *I* for HSI, and *H* vs *I* for HDI. The bottom plot is *S* vs *V* for HSV, *S* vs *L* for HLS, *S* vs *I* for HSI, *D* vs *I* for HDI, and *g* vs *r* for NCC.

Secondly, the camera operating parameters were automatically adjusted by the camera and some object pixels were saturated in one or more color bands as shown in Figure 9. The saturated pixels came mostly from the area of a bright specular reflection on the ball.

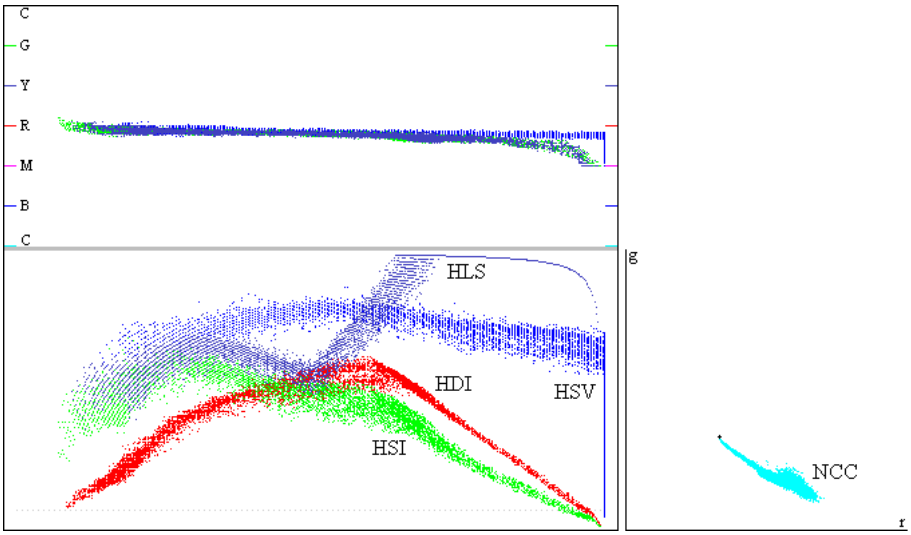

Figure 9. Red ball pixel distribution with color saturation present.

Figures 8 and 9 show that *RGB* values vary a lot and that no single RGB triplet or a spherical cluster in the RGB space can describe the color of the red ball. It is evident that the Hue of the red ball is changing with Value/Lightness/Intensity in the HSV/HLS/HSI/HDI color models. Researchers often make an implicit assumption that the hue does not change with intensity but in real environments this is usually not the case. The Hue changes slowly from red to magenta in our example. The reason for this will become evident later in this paper.

As indicated by the vertical lines in Figure 9, there is an inherent problem within the HSV model in describing the behavior of *H* and *S* when $V = 1$. When *V* reaches 1 it cannot increase further. However *H* and *S* still change. The HLS model almost solves this problem.

When $S = 1$, Hue is represented uniquely. In theory, the $S = 1$ line should stay horizontal, and, at $L = 1$, S and H are undefined. However, since we are dealing initially with integer RGB numbers and limited bit per integer resolution, the to be expected straight line is bent into a curve approaching 0 at $L = 1$. As shown in Figure 9, the HSI model, with its definition of S, solves these problems, but the user may have difficulty seeing from the HSI distribution how pixels are initially spread in the RGB space. The NCC model is used to remove the intensity component from pixel color and to represent an object as a symmetrical cluster in the *rg* space. As also seen in Figure 9, the shape of the cluster is neither spherical nor symmetrical.

As mentioned before, most illustrations in this paper are based on the HDI color model defined above. The distribution of pixels from the red ball at automatic exposure is shown separately in Figure 10.

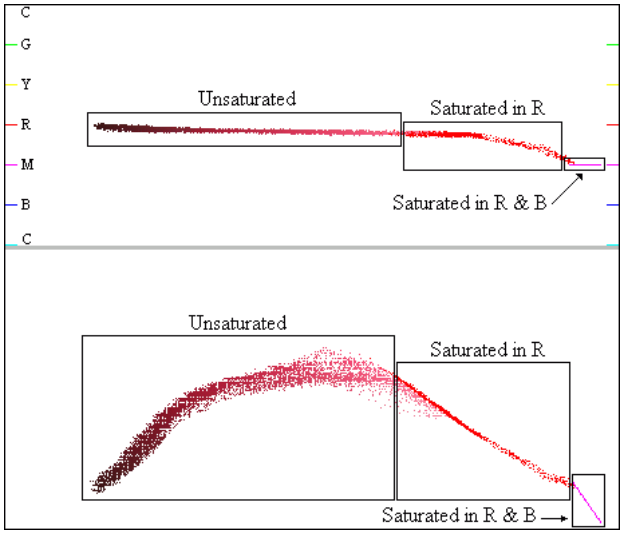

Figure 10. Red ball pixel distribution at automatic exposure.

The cluster in this figure is formed from three distinct parts: unsaturated pixels that lie inside the RGB cube, pixels saturated in one color band that lie on the side of the RGB cube, and pixels saturated in two color bands that lie on the edge of the RGB cube. One intuitively sees that the

cluster lies more or less in the plane. The top plot would correspond to the edge view of this plane, and the bottom plot would correspond to the view perpendicular to this plane. Several questions arise from these plots: why Hue is changing with Intensity, why the color cluster lies more or less in the plane, and why the cluster resembles a boomerang shape. In further sections we try to answer these question.

2.3 The Dichromatic Reflection Model

The use of the *Dichromatic Reflection Model* suggested by S. Shafer [42] helps solve a problem associated with the illumination changes, namely, shadows and highlights, and also takes into account the color (spectral distribution) of the illuminant. Although problems with object inter-reflections and spectral changes of the illuminant still have to be addressed, the implemented segmentation system using this model showed more robustness and higher speed compared to either the popular color model or to pixel clustering approaches.

The dichromatic reflection model describes the light which is reflected from a point on a dielectric, nonuniform material as a mixture of the light reflected at the material surface and the light reflected from the material body. The first, the surface reflection component, generally has the same spectral power distribution as the illumination and appears as a highlight on the object. The second, the body reflection component, provides the characteristic object color and exhibits the properties of object shading. Thus, according to the dichromatic reflection model, all pixels belonging to a color object may be described as a linear combination of two vectors.

The details of the above may be explained thustly. The dielectric materials such as plastic, paper, or ceramics consist of an approximately transparent medium and some imbedded pigments (colorant), as shown in Figure 11.

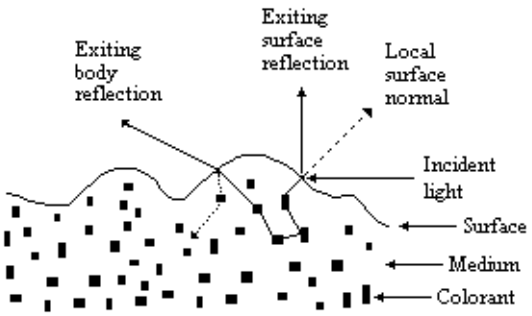

Figure 11. Light reflected of dielectric materials.

Since the refraction index of the dielectric material is generally different from that of the surrounding medium, some percentage of incident light is reflected at the material surface as shown in Figure 11. This light reflection is referred to as *surface reflection*. Some percentage of the incident light penetrates into the material body, travels through the medium hitting pigments. When light penetrates through the pigment particle, some wavelengths are partially or entirely absorbed. The remaining light returns to the surrounding medium and undergoes further scattering and absorption. One part of this light travels back to the material surface and after partial reflection goes back into the material. Another part of this light refracted by the material surface exits the material as shown in Figure 11. This light reflection (i.e. light exiting the material) is referred to as *body reflection*.

In theory, the direction of surface reflection is defined by the angle of perfect mirror reflection. However, because of the roughness of the material surface, the reflection is scattered to some degree around the mirror angle. The amount and color of the surface reflected light depends on the optical properties of the material and the surrounding medium. Since Fresnel's coefficients of most media change very little over the visible spectrum, it may be assumed that the refraction indexes are constant and that the light reflected at the surface is the same color as the illuminating light [28].

The direction, amount and color of the body reflection depends on many factors such as material medium, scattering and absorption properties of the pigments, their shape and distribution. If pigments are distributed randomly in the material, the light will exit in random directions from the body, and, on average, the same amount and color of the incident light will be absorbed everywhere in the material before the light exits. In this case, the light reflected from the material body will have the same color over the entire surface. If the exiting light is uniformly distributed, this distribution may be described by Lambert's law. For many applications this is an acceptable approximation.

As stated by the dichromatic reflection model, the reflected light from every point on the dielectric object can be described as a linear combination of the surface reflection vector $C_S(\lambda)$ and the body reflection vector $C_B(\lambda)$, as seen in Figure 12.

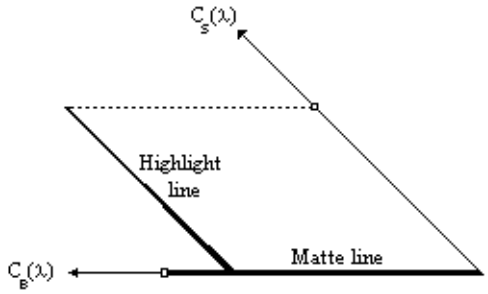

Figure 12. A color cluster defined by the body and the surface reflection vectors.

Highlight pixels form a *highlight line* in the direction of the surface reflection vector. Matte pixels form a *matte line* in the direction of the body reflection vector. The length and position of both lines is defined by a variety of factors which we discuss later.

CCD color cameras use a finite set of samples to describe color from a continuous light spectrum. Such sample measurements are obtained by filtering the light spectrum and integrating over the filtered spectrum. This process is called a *spectral integration*. In CCD cameras, red, green, and blue filters are usually used to reduce the infinite dimensional vector space of spectral power distributions to a three-dimensional RGB color space. This transformation is a linear transformation [41]. With a three-dimensional RGB color space, the body reflection and surface reflection vectors span a *dichromatic plane* containing a parallelogram defined by these vectors. The color cluster lies within this parallelogram. The colors of all rays reflected from the dielectric object then form a planar *color cluster* in the RGB space. This is shown in Figure 13.

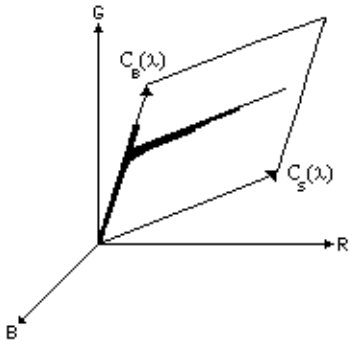

Figure 13. A dichromatic plane in RGB space.

The dielectric object pixels form color clusters shaped as a skewed "T" or a skewed "L" lying in the dichromatic plane in the RGB space. The geometry of the clusters may vary and is defined by the illumination and viewing conditions, as well as by the characteristics of the camera. The angle of the cluster depends on the body and surface reflection vectors. The position of the highlight line relative to the matte line depends on the viewing geometry. If the specular spot is located in the area of maximum body intensity, the cluster looks like a skewed "L". If the specular spot is located off the area of maximum body intensity, the cluster looks like a skewed "T" [27].

To verify the dichromatic reflection model and to determine how the distribution of pixels changes with the illumination of various colors, a second set of experiments was undertaken. In each experiment of this set a table lamp was used to which red, green, and blue Kodak color filters were applied. These filters had the same transparency (magnitude) but different colors. The distribution of pixels of the red ball at fixed camera exposure and using various filters is shown in Figure 14.

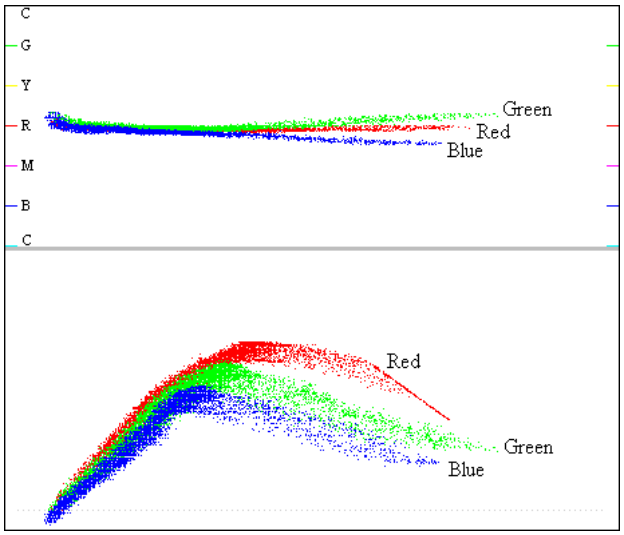

Figure 14. Filter response of the red ball.

The red ball illuminated with red light does not show any changes in Hue. In this case, the body reflection color vector and the surface reflection color vector are co-planar with the gray diagonal of the RGB cube, resulting in the horizontal Hue line. With the green color filter, the tendency of Hue to become green when Intensity increases can be observed. With the blue color filter, the tendency of Hue to become blue when Intensity increases can be observed. The Hue curves for green and blue filters are inclined respectively. The bottom plot of Figure 14 demonstrates the likely decreasing spectral responsivity of the camera to short wavelengths so that red is sensed brighter than blue.

In the second experiment of this set, to amplify the effect, a glossy bluish mouse pad shown in Figure 15 was used. The response to the color filters is shown in Figure 16.

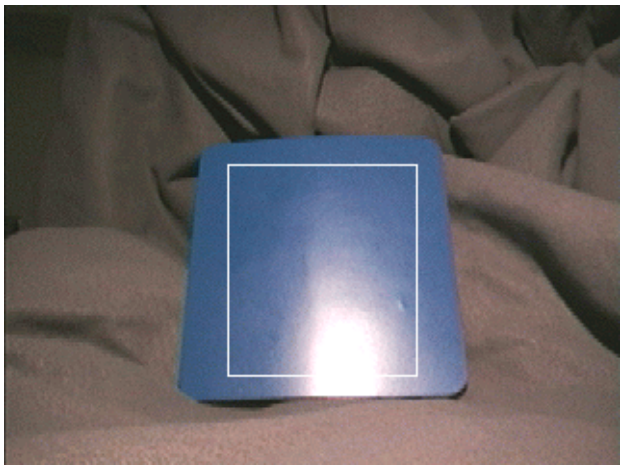

Figure 15. A sample area of the mouse pad.

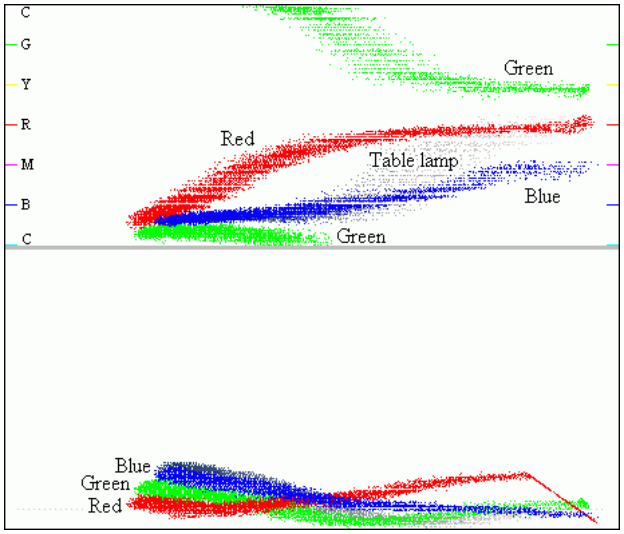

Figure 16. Filter response of the mouse pad.

Note that with no filters applied the Hue tends to become red. It turned out to be that the table lamp used had illumination color close to red.

This explains getting yellow Hue (between red and green) when the green filter was applied and getting magenta Hue (between red and blue) when the blue filter was applied.

The dichromatic reflection model answered two of the three previously considered questions: why Hue is changing with Intensity and why the color cluster lies more or less in the plane. To answer the question of why the cluster resembles a boomerang shape, we have to address the limitations inherent in many digital cameras that introduce distortion to the linearity of the dichromatic reflection model. These issues are discussed in the next section.

2.4 Camera Limitations

The dichromatic reflection model provides a good theoretical background as to how pixels of a dielectric object are distributed in color space. Such a distribution in real settings will now be addressed. In all the above experiments a color CCD camera (Sony EVI-D30) was used. Thus further analysis of camera limitations and their effect on pixel distribution will concentrate on the issues occurring with this camera.

2.4.1 Limited Dynamic Range

Cameras have a limited dynamic range to describe the intensity of the incoming light. Typically, color pixels contain Red, Green and Blue values each measured in 8 bits. This restricts the analysis of light to an RGB cube shown in Figure 1. If the incoming light is too bright at some pixels and exceeds the dynamic range, the camera can neither sense nor represent it adequately so that some or all color bands become saturated to a maximum possible value. In the RGB space this causes the color cluster to bend along the sides and edges of the RGB cube. This process is referred to as *color clipping*.

2.4.2 1-CCD Array

The color CCD camera used in our experiments has a 1-CCD array. This means that CCD elements of three types are evenly distributed in the array with each type sensitive to a particular spectral band corresponding to red, green and blue. Since a physical pixel may not contain CCD elements of all three types, the hardware approximates the value of the absent element based on the values of this element in the neighboring pixels where the element is present. Such approximation introduces error in the color representation of a pixel, and, in relation to the color cluster, it causes pixels to shift in various direction from their true locations.

2.4.3 NTSC and S-Video

The camera we used provided a NTSC video signal to a frame grabber via S-Video port. The NTSC signal uses YUV color system to transmit information about the intensity (Y) and color (UV) of a pixel with a resolution ratio of 2:1:1 respectively. The original RGB signal was converted by the camera into YUV and then back into RGB by the frame grabber. There is an imprecise representation of RGB values in the output image as a result of two transformations involving color resolution reduction. In relation to the color cluster, it causes pixels to form into sparse groups and to shift in various direction from their true locations.

2.4.4 Spatial Averaging

It is common in CCD cameras to use the *spatial averaging* between neighboring pixels to reduce variance in the sensing characteristics of the array. This causes smoothing of saturation, blooming and other undesirable effects over the image. In relation to the color cluster it causes pixels to shift in various direction from their true locations.

2.4.5 Gamma-Correction

Most modern CCD sensors have a linear response to the incident light flux. However, since the luminous output of the display devices is related by a power law to the driving voltage, $L = kV^{\gamma}$, it is common practice in the television industry to produce a signal proportional to $Y^{(1/\gamma)}$ in the cameras, with 2.2 as a typical value for γ. This is known as *gamma-correction*. As a result, the camera output is related by this inverse power law to the incident flux [28]. Gamma correction introduces curvature into the color space, distorting the linear properties of the dichromatic reflection model. A typical curve for gamma-correction may be found in Figure 10, in the unsaturated pixel section. Ideally, it should be a line.

One may wish to verify whether or not a matte line corresponding to the body reflection vector is transferred into a planar gamma-correction curve. Consider the normal to the plane formed by the mate line and the gray diagonal of the RGB cube.

The plane normal (A,B,C) may be found as a cross product of the body reflection vector (R,G,B) and the gray diagonal (1,1,1).

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ R & G & B \end{bmatrix} = 0 \quad \Rightarrow \quad \begin{aligned} A &= B - G \\ B &= R - B \\ C &= G - R \end{aligned}$$

If gamma-correction is introduced, the mate line (kR, kG, kB) with $k \in [0; +\infty]$ will be transformed into $((kR)^{\gamma}, (kG)^{\gamma}, (kB)^{\gamma})$ curve and A, B, and C will change to A_g, B_g, and C_g as follows:

$$\begin{aligned} A_g &= (kB)^{\gamma} - (kG)^{\gamma} \\ B_g &= (kR)^{\gamma} - (kB)^{\gamma} \\ C_g &= (kG)^{\gamma} - (kR)^{\gamma} \end{aligned} \quad \Rightarrow \quad \begin{aligned} A_g &= k^{\gamma} * (B^{\gamma} - G^{\gamma}) \\ B_g &= k^{\gamma} * (R^{\gamma} - G^{\gamma}) \\ C_g &= k^{\gamma} * (G^{\gamma} - R^{\gamma}) \end{aligned}$$

Consider variable X to be k^{γ}, and constants m_1, m_2, m_3 to be

$$m_1 = (B^{\gamma} - G^{\gamma}) \qquad A_g = X * m_1$$
$$m_2 = (R^{\gamma} - G^{\gamma}) \quad \text{then} \quad B_g = X * m_2$$
$$m_3 = (G^{\gamma} - R^{\gamma}) \qquad C_g = X * m_3$$

We can remove X from the equations, since it does not change the direction of the (A_g, B_g, C_g) vector but only varies its magnitude. Thus

$A_g = m_1$ where m_1, m_2, m_3 are constants
$B_g = m_2$
$C_g = m_3$

Therefore it has been proved that the matte line is transformed into a planar gamma-correction curve.

Unfortunately, the dichromatic plane spanned by the body reflection and the surface reflection vectors will not always be transformed into a plane. This will be the case if the body and surface reflection vectors, gray diagonal and one edge of the RGB cube are all co-planar, but chances of this occurring in a real setting are very low. More often the dichromatic plane will be transformed into a non-planar *dichromatic surface*. More than one planar segment may be required to approximate this surface. In relation to the color cluster, gamma-correction transforms the theoretical skewed "T" or a skewed "L" cluster into a boomerang shape cluster observed in our plots.

2.4.6 Sensitivity to the Infrared Light

Camera responsivity to the infrared range adds some value to the intensity of color pixels. Infrared, if present, may wash out the colors in the image and cause a shift of the whole color cluster in the color space. To solve this problem, many cameras have in-built infrared supressors. However, some of these do not completely block infrared.

2.4.7 Spectral Responsivity

Because of the ramp-like spectral responsivity of CCD cameras within the visible spectrum, the blue band registers much less intensity than the red band [33]. This may be seen in the bottom plot of Figure 14, where filters of the same magnitude but different colors were applied. This effect lowers signal-to-noise ratio for short waves and causes a shift of the whole color cluster in the color space.

2.4.8 Chromatic Aberration

Chromatic aberration is an intrinsic property of a lens. It exists because the index of refraction of optical components varies as a function of wavelengths. Thus, for a fixed focal length, only one color band of an image can be perfectly in focus. The images in two other bands correspond to a slightly magnified or shrunk version of the undistorted image. This causes color changes in the outer image areas which are far from the optical axis and in the areas with a rapid color change such as edges [34]. Color aberrations causes the pixels of the color cluster to drift in various directions.

2.4.9 Camera Noise

Noise, a common problem inherent to electronic devices, affects pixel values and causes them to fluctuate within a certain range. The color images we processed contained non-uniformly distributed noise. According to the specification, the color camera we used had a S/N ratio of 48 dB. The frame grabber had on the average 1.5 bad bits out of 8 with the least significant bit reflecting more noise than data. The fluctuations are not always visible to a human eye, but we have to take them into account in the context of the considered dichromatic reflection model. In theory, noise introduces thickness of the dichromatic plane, and, in practice, it introduces thickness of the dichromatic surface.

2.5 Camera Operating Parameters

Automatic exposure is a common feature in many modern cameras. It automatically controls *iris* (aperture), *shutter* (exposure) and *gain* to optimally adjust brightness of the picture depending on the illumination conditions and brightness of the seen objects. The camera was tested to see how these adjustments affect the distribution of pixels in the color space. In the third set of experiments we used the red ball made of a soft plastic, a Sony color camera and indoor fluorescent illumination. Firstly, the shutter was manually fixed at 60 Hz, gain at 9 dB, and the iris was changed from F2.0 to F4.8 (from close to open). The effect of these changes is shown in Figure 17. The bigger the iris the brighter the image. Note that the distributions comply to the dichromatic reflection model and that there is no unexpected change in Hue.

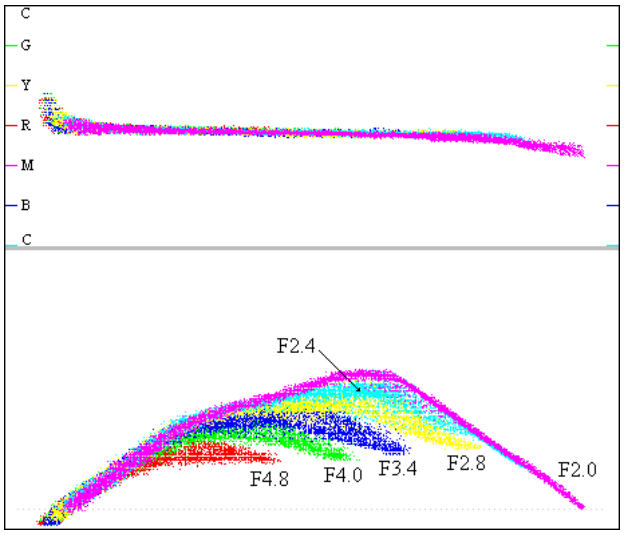

Figure 17. Iris response.

Secondly, the iris was manually fixed at F1.8, gain at 6 dB, and the shutter was changed from 60 to 500 Hz. The effect of the changes is shown in Figure 18. The faster the shutter the darker the image. Note that the distributions again comply to the dichromatic reflection model and that there is no unexpected change in Hue.

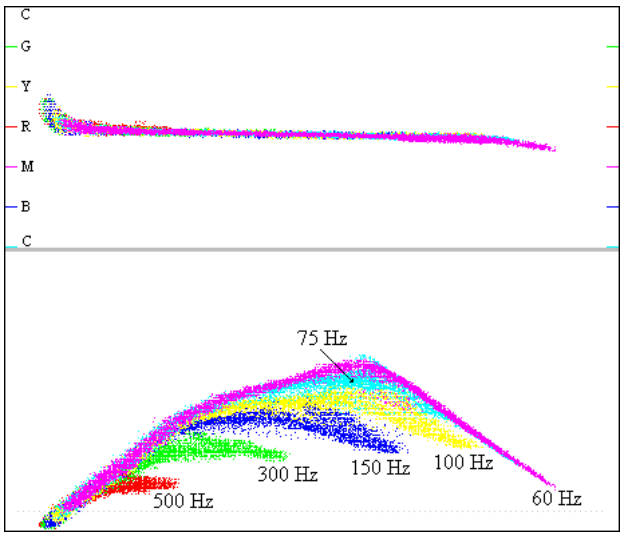

Figure 18. Shutter response.

In the third experiment the iris was manually fixed at F2, shutter at 60 Hz, and the gain was changed from -3 to 12 dB. The effect of the changes is shown in Figure 19. At the beginning the color cluster is scaled until it hits the surface of the RGB cube. The cluster then flattens slightly, getting closer to the gray diagonal. Note again that the distributions comply to the dichromatic reflection model and that there is no unexpected change in Hue.

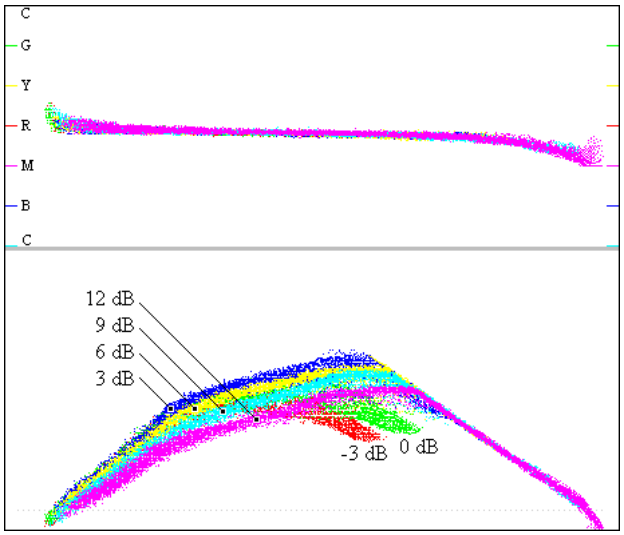

Figure 19. Gain response.

2.6 Approximating the Dichromatic Surface

In previous sections the factors that influence the distribution of pixels in the color space were discussed. From this discussion and the experiments performed it is known that a dichromatic plane is transformed into a non-planar dichromatic surface. The problem is how to approximate this surface. One way that may be suggested is statistical. This involves sampling the color clusters formed by an object viewed at various illumination conditions (same spectra but different intensity), and viewing conditions and accumulating the sampled data in an array. Alternatively, a principal components analysis might be applied to the color clusters of the same object at various illuminations and various viewing conditions to get eigenvectors. The cross product of the two biggest eigenvectors will determine the normal of the plane approximating the dichromatic surface. A third, more simple approximating technique of fitting two planar slices to the dichromatic surface is suggested here and tried to see if satisfactory results could be achieved.

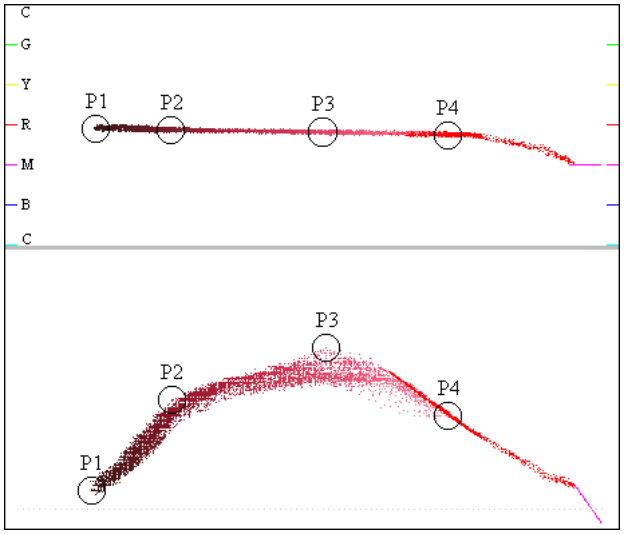

Figure 20. Reference points on the color cluster.

The color cluster shown in Figure 20 is typical for the red ball used in our experiments as viewed by the camera with automatic exposure at the indoor fluorescent illumination.

Initially three points were selected from the unsaturated pixels of the color cluster. In the HDI model two points correspond to the following: P1 is the point with minimum Intensity, P4 is the point with maximum Intensity. P3 is the furthest from the P1-P4 line. In a rough approximation, P1-P3 corresponds to the body reflection vector, and P3-P4 corresponds to the surface reflection vector. A plane was fitted using points P1-P3-P4. However, many pixels in the area between points P1 and P3 turned out to be off the plane. Then point P2 was introduced as the furthest unsaturated pixel from the P1-P3 line. The second plane was fitted using points P1-P2-P3. This considerably increased the recognition of the object pixels. In fact, often a specular spot occupies a relatively small area of the object surface, and fitting a separate plane to the planar gamma-curve that corresponds to the matte line will cover the matte pixel majority with higher accuracy.

The thickness (Th) of the fitted planes was set according to the average deviation (Ea) of unsaturated pixels from the assigned regions, P1-P2-P3 for the body reflection plane, and P1-P3-P4 for the surface reflection plane respectively. The thickness was chosen to be Th = Ea * 2. The thickness is in the same measurement units as *R, G, B* in the RGB model and *D, I* in the HDI model. Experimental results are provided in the *Result*s section of this paper.

In theory, the color cluster lies in a restricted area, to be exact, within the parallelogram defined by the body and surface reflection vectors. In practice, the area within the dichromatic surface must be restricted. For that purpose the directions of P1-P2 and P3-P4 lines as shown in Figure 21 were used.

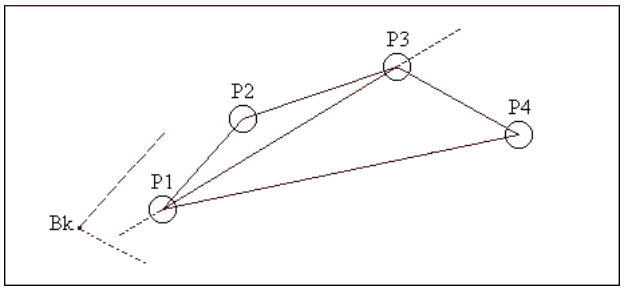

Figure 21. Area restriction of the dichromatic surface.

For plane P1-P2-P3 we calculated two perpendicular planes, the first running through P1-P3, the second running through the black point of the RGB cube parallel to P1-P2 line. A slice of the P1-P2-P3 plane restricted by the two perpendicular planes and the surface of the RGB cube was used for further consideration. By analogy, for the P1-P3-P4 plane two perpendicular planes were calculated, the first running through P1-P3, the second running through the black point of the RGB cube parallel to P3-P4 line. A slice of the P1-P3-P4 plane restricted by the two perpendicular planes and the surface of the RGB cube was used for further consideration.

It is hard to restrict the side farthest from the black point because the varying camera operating parameters may force the border to be pushed to the surface of the RGB cube. For this reason we let this surface represent the border.

2.7 A Look-Up Table Representation of the Object Color

As mentioned before, in color digital images, pixel values usually contain Red, Green and Blue values each measured in 8 bits. A typical color object segmentation would involve a conversion of these values to some color model parameters, then comparison of these parameters to the assumed object model invariants, with Hue being one of the most common. The disadvantages of this approach are: a slowdown due to an intensive use of computationally expensive operations, such as division, square root and trigonometric functions for every analyzed pixel, and, in general, an imprecise representation of the object color at various viewing conditions and at non-white illumination. An ideal way of segmenting a color object in the image would be to interpret a pixel value as a 24-bit (or less, depending on the desired color resolution) number, and to use this number as an index in a *Look-Up Table (LUT)*, the entries of which tell whether or not the analyzed pixel belongs to the object. Such a check would be extremely fast, yielding real-time performance. Further, a method for creating such an LUT using the dichromatic reflection model is suggested.

To create a Look-Up Table for fast color object segmentation the RGB color model was used. Consider a cube consisting of zeros and ones, where ones represent pixels that belong to the object and lie on the dichromatic surface, or, in this case, on the planar slices approximating the surface. Since color resolution of disaturated colors near the gray diagonal of the RGB cube and colors near the black and white corners of the cube is low, a safe volume determined by a cylinder with the axis being the gray diagonal is used. Pixels inside a cylinder volume are disregarded. The diameter of the cylinder was chosen to be 10. With these analytical descriptions in mind - the two planar slices, and the safe cylinder - ones are turned where the dichromatic surface runs, and zeros are left in the rest volume of the RGB cube.

With 8 bits per color (bpc) and three colors (*RGB*) one may obtain 16,777,216 various colors. Experiments on how perceptually different color images look at lower color resolutions were performed with the result that humans cannot perceive any difference up to 6 bpc. Also, the camera noise was measured by subtracting neighboring frames of the same image with the result that the lower 2 bits of the RGB byte values are in constant flux. When auto focus and auto exposure features were activated noise increased. To represent color resolution, 6 bpc was selected. With 6 bpc, 262,144 various colors may be obtained. To represent this number of colors, 262,144 entries are needed in the LUT. While 262,144 bytes may be used, this however would be a waste of space since only one bit of a byte entry will actually be used.

The LUT may be compressed into 262,144 bits (32,768 bytes), thus obtaining a very small LUT that may be used in applications where the amount of memory is a concern.

To reduce color resolution a simple algorithm suggested below may be utilized. See Figure 22.

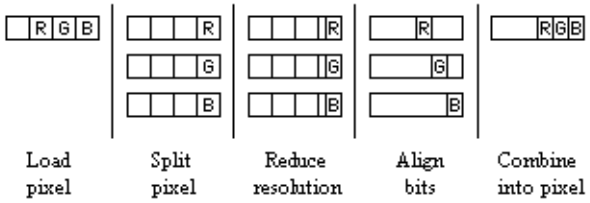

Figure 22. Reduction of color resolution.

In the initial pixel, three out of four bytes represent *RGB* values (the fourth upper byte is often zero and unused). The reason for using four bytes per pixel instead of three is that the packs of four bytes are processed by a CPU much faster than the packs of three. We split the four byte integer representing a pixel into three integers containing only *R*, *G*, and *B* values. Color reduction is performed as a simple register bit shift operation. The remaining *R*, *G*, and *B* values are then aligned and combined back into a single integer. The resulting integer may be used to access a particular entry in the LUT. The assembly code implementing the algorithm may be found in Appendix A.

To access the bit entry in the LUT that defines whether or not a pixel belongs to the object another simple algorithm is suggested. The pixel value, after reducing color resolution, is divided by 8 (by a bit shift to the right). The integer part of the division tells which byte of the LUT to check. The modula part of the division (obtained by a bit mask) tells which bit of the byte to check. The assembly code provided in Appendix A implements the LUT access algorithm. Result is either zero or non-zero. It may be used as a boolian variable there after.

As is seen from the provided code, the main advantages of using the LUT approach are speed and simplicity. All operations used, except memory access, are simple one CPU clock cycle operations. There are no divisions, square roots, trigonometric functions or other computationally expensive operations.

2.8 Results

The results of segmenting a color object from a sequence of live images using the suggested approach are very satisfactory. With an LUT occupying only 32 Kb of memory (6 bpc) and an RGB image of 320 x 240 pixels an extremely high speed segmentation was achieved. It takes 8 ms to segment the image using the suggested approach compared to 80 ms segmentation based on hue. Calculating hue for every pixel involves at least one integer division which significantly slows down the segmentation process. In NTSC a new frame comes every 33 ms (30 frames per second) and to stay real-time, without dropping frames, one should complete segmentation within a 33 ms limit. This has been achieved using the suggested approach.

The segmentation also accounts for a non-white illuminant, shadows and highlights. The achieved quality of segmentation is also higher compared to the hue-based segmentation which assumes object color to be of a uniform hue. A non-white illuminant and a specular surface cause a hue range to increase to accommodate areas of color space that represent the highlight. However, such inclusion also includes areas of color space that do not belong to the object, thus introducing additional noise and causing error in object recognition.

Figure 23 shows the red ball and the yellow plastic duck seen by the camera with automatic exposure at the indoor fluorescent illumination.

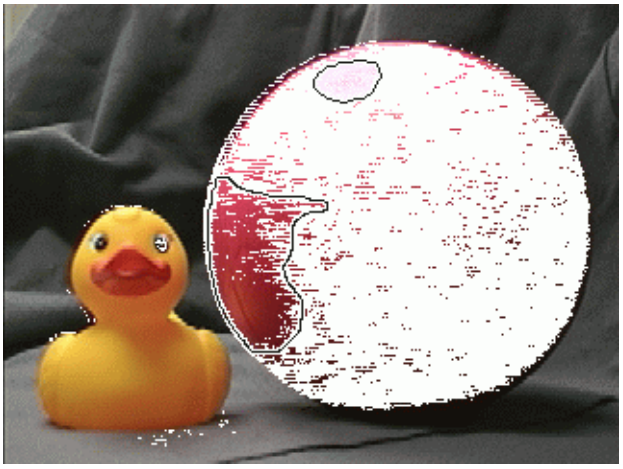

Figure 23. Resulting object segmentation.

White areas on the ball surface are recognized as belonging to the object, or as lying on the approximated dichromatic surface. There are two spots that are not recognized on the ball. One, on top, is a very bright area with pixels close to white. These pixels are either inside the safe cylinder volume and thus disregarded or are far from the dichromatic surface due to its rough approximation. The other spot is near the plastic duck. This spot is a result of inter-reflection. The yellow plastic of the duck reflects a yellowish light on the ball. This causes the pixels to drift off the dichromatic surface. When the duck is removed the spot is well recognized. In turn, the black eye of the duck has picked up some reddish inter-reflection from the ball. Some of the eye pixels are recognized as belonging to the ball.

The duck's beak has a carrot color close to red but is not recognized as the ball. This means that with selected color resolution and dichromatic surface thickness the colors of the two objects can still be well distinguished.

Satisfactory results were also obtained when the suggested approach was applied to model the color of human skin. Multiple objects of skin color were segmented out with acceptable quality in real time.

Figure 24 shows three superimposed color clusters of the red ball illuminated by a table lamp with a blue filter applied and seen by the camera with automatic exposure under varying illumination intensity and various viewing conditions.

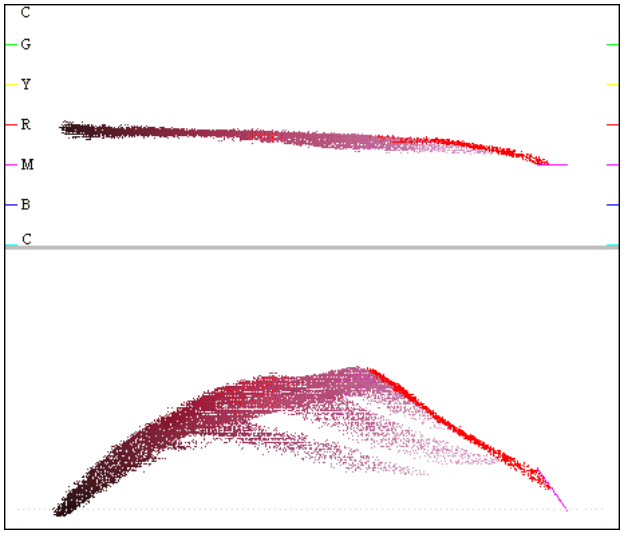

Figure 24. Fluctuations of the color cluster within the dichromatic surface.

These clusters lie in the dichromatic surface which in our case is approximated by two slices. The coefficients of P1-P2-P3 and P1-P3-P4 planes and their corresponding thicknesses are given below.

Points is the number of points in the slice

Em is the mean deviation of points from the slice, (Σ deviation) / Points

Ea is the average deviation of points from the slice, (Σ | deviation |) / Points

Th is thickness, Ea * 2

A, B, C, D coefficients of the plane equation of type

A * Red + B * Green + C * Blue + D = 0

Body reflection	Surface reflection
slice P1-P2-P3	slice P1-P3-P4
Points = 6018	Points = 2239
A = 0.145	A = 0.276
B = 0.772	B = 0.620
C = -0.619	C = -0.734
D = -7.503	D = -8.574
Em = 0.6	Em = -0.8
Ea = 3.2	Ea = 3.4
Th = 6.4	Th = 6.8

The angle between the slice normals is 13.3°. When the red Kodak filter was applied instead of the blue one to straighten the Hue curve, the angle changed to 4.3°, and thicknesses changed to 6.0 and 6.6 for the body and the surface slices respectively.

2.9 For Further Investigation

Although the results are very satisfactory, several things can be improved while a few problems need to be addressed.

2.9.1 Assumptions of the Dichromatic Reflection Model

The dichromatic reflection model contains several assumptions that are both restricting and simplifying. It assumes that the object surface exhibits a single spectrum of body reflection and a single spectrum of surface reflection. This requires that the object be uniformly colored and have both types of reflections present. It also restricts the illumination conditions of the scene allowing only light sources of a single spectra. Introducing an additional illuminant of a different spectrum will introduce an additional dichromatic surface in the RGB space. Interestingly, if several objects of different color are illuminated by a single light source, the intersecting line (curve) will represent the surface reflection vector. This may be used to estimate the color of the illuminant [47]. Adding ambient light would add a single term to every pixel of the color cluster, and cause a shift of the parallelogram defined by the body and surface reflection vectors away from the origin along the vector of ambient light [42]. This may result in a shift of the whole color cluster in the RGB space. Inter-reflections from surrounding objects also introduce distortion of the color cluster. Inter-reflection forces pixels to drift off the dichromatic surface. Increasing the thickness of the dichromatic surface may compensate for this drift, but the side effect of such an increase would be inclusion of areas of the color space that may belong to other objects.

However, for many applications, the dichromatic reflection model still provides a reasonable and useful description of the physics of light reflection.

2.9.2 Approximation of the Dichromatic Surface

In our approximation of the dichromatic surface quite a simple approach was used. Only two planar slices were fitted and they worked reasonably well. Nevertheless, the surface may be approximated more precisely. The choice of the reference points in our method is noise prone and the results are not repetitive. Considering all points in the slice will yield a higher accuracy of fitting. Another way to improve the approximation is to fit more planar slices. A combination of statistical data accumulation for particular regions of the surface, then application of the principal

components analysis at each region may yield a much better description of the dichromatic surface. Alternative way to a more precise description could be a linearization of pixel distribution in order to acquire an equation of the dichromatic plane. Then, by a reverse process, the plane may be transformed into a dichromatic surface. The later approach may be difficult to implement due to its complexity, i.e., an undefined order of transformations, and the great number of camera limitations.

2.9.3 Other Cues for Object Recognition

Color is an excellent clue for object recognition. However, using color alone may not be sufficient. Additional cues as intensity, texture, or geometry may improve accuracy of recognition.

2.10 Conclusion

In our work, an effort was made to develop an approach to efficient color object segmentation from a sequence of live images for use in real-time applications. A novel, compact, look-up table color representation of a dielectric object that modeled the behavior of a color cluster, yielded real-time performance and accounted for non-white illuminant, shadows, variable viewing conditions and camera operating parameters was proposed. Further development based on this approach may result in more efficient color representation of various materials and multi-colored objects.

Chapter 3

Using Density and Spatial Cues for Clustering Image Pixels in Real-Time

3.1 Problem

Often the intermediate result of identifying objects in the image is a binary image, with pixels reflecting presence or absence of the object. For instance, the binary image may be obtained as a result of image segmentation based on color, optical flow or correlation. Sometimes, instead of binary values, pixels may contain information on probability or uncertainty of the pixel belonging to the object. The problem is how to cluster these pixels efficiently in separate objects.

3.2 Introduction to Cluster Analysis

Clustering techniques seek to separate a set of data into groups or clusters and may be classified into types roughly as follows [7]:

a. *Hierarchical techniques* - in which the clusters themselves are classified into groups, the process being repeated at different levels to form a tree.

b. *Optimization-partitioning techniques* - in which the clusters are formed by optimization of a "clustering criterion". The clusters are mutually exclusive, thus forming a partition of the set of entities.

c. *Density or mode-seeking techniques* - in which clusters are formed by searching for regions containing a relatively dense concentration of entities.

d. *Clumping techniques* - in which the clusters or clumps can overlap.

e. *Others*

Many attempts have been made to propose a formal definition of the term cluster but none claims to provide a universal definition. One states that cluster is a group of contiguous elements of a statistical population. Others contain statements such as: a cluster is a set of entities which are alike, and entities from different clusters are not alike. Others suggest that entities within a cluster are more similar to each other than the entities from another cluster.

A description of what constitutes a cluster which probably agrees closely with our intuitive understanding of the term [7] is given by considering entities as points in a p-dimensional space, with each of the *p* variables being represented by one of the axis of this space. The variable values for each entity define a p-dimensional coordinate in this space. Clusters may now be described as continuous regions of this space containing a relatively high density of points, separated from other such regions by regions containing a relatively low density of points. This description matches the way we detect clusters visually in two or three dimensional space.

The main purpose of clustering data is to reduce the size and complexity of the data set and to identify classes of similar entities. Data reduction may be accomplished by replacing the coordinates of each point in a cluster with the coordinates of that cluster's reference point, or assigning a point to a particular cluster. Clustered data require considerably less storage space and can be manipulated more quickly than the original data. The value of a particular clustering method depends on the application and may, for example, reflect how closely the reference points represent the data, or, how closely clusters represent specific shapes or volumes. An important factor also is the speed of the clustering algorithm.

What determines good or representative clustering? For example, consider a single cluster of points along with its centroid or mean. If the data points are tightly clustered around the centroid, the centroid will be representative of all the points in that cluster. The standard measure of the spread of a group of points about its mean is the variance, often the sum of the squares of the distance between each point and the mean. If the data points are close to the mean, the variance will be small. A generalization of the variance, in which the centroid is replaced by a reference point that may or may not be a centroid, may be used to indicate the overall quality of a partitioning. Another example, is where the similarity of the object and cluster shape may be used to indicate the overall quality of a partitioning.

All types of clustering techniques have their specific problems:

Hierarchical techniques have a general disadvantage since they contain no provision for reallocation of entities which may have been poorly classified at an early stage in the analysis.

Optimization techniques, which seek to optimize some criterion, have a problem of finding a sub-optimal solution instead of a global solution. This is known as a local optima problem. Most optimization techniques also presume that the number of clusters is either known or given prior to clustering.

Density seeking methods, such as fitting mixtures of multivariate normal distributions, also suffer from the problem of sub-optimal solutions, since there may be more than one solution to the maximum likelihood equations.

Other techniques have their own problems, such as presuming that points form hyper-spherical clusters, what may not often be the case.

It should be noted that data noise is a problem for all clustering techniques. Noise may produce misleading clustering results by introducing additional clusters that are of no significance or by deforming or merging “good” clusters.

Therefore, the investigator who decides to use cluster analysis techniques and has a large number of methods available to him must examine the assumptions made by a particular method and satisfy himself that such assumptions are reasonable for his particular data. Because different techniques are likely to give different solutions, he should consider the validity of any found clusters and try various methods with the data to select the one giving the best results.

3.3 Previous Work in the Field of Clustering

The fundamentals of cluster analysis, as well as a good review of classical clustering techniques, are given by B. Everitt [7], R.O. Dabes. et. al. [6] and J. Hartigan [20]. L. Kaufman and P. Rousseeuw [26] provide detailed programming examples of six common clustering methods. An interesting, simple, efficient clustering algorithm called a "greedy algorithm" was proposed by T. Gonzales [17] and later improved by T. Feder and D. Greene [9]. Further development of this method may be found in the work by P. Agarwal and C. Procopiuc [1]. Another simple and efficient algorithm for clustering data, an extremely popular "k-means" algorithm, was conceptually described by S.P. Lloyd [30] and later improved by J. MacQueen [31]. Many papers [5, 8, 23, 40] suggest various improvements to the "k-means " algorithm, mostly based on the use of the random sub-sampling of the data sets at various stages of the algorithm. K. Fu and R. Gonzalez [13] suggested a split-and-merge algorithm to cluster areas using a decreasing resolution technique. A 2D version of this algorithm, also know as a quad-tree algorithm, is described by R. Gonzalez and R. Woods [16].

Out of a great number of clustering methods, the two most popular, "k-means" and "greedy", deserve special attention. We provide a brief description of each method below.

3.3.1 The "k-means" algorithm

Given S a set of N points and K the number of clusters, the algorithm

1. Chooses K reference points (e.g., at random) from S. Each reference point R_i defines a cluster C_i.

2. Then data points are partitioned into K clusters. Point p of S becomes a member of cluster C_i if p is closer in the underlying metric (e.g., the Euclidian distance) to R_i - the reference point of C_i than to any other reference point. Closer means - min d(R_i, p), where d(R_i, p) is a distance between point R_i of R and point p of S in the underlying metric.

3. The centroid for each cluster is calculated and the centroid becomes a reference point of its cluster.

4. During successive iterations, the centroids of each cluster are adjusted.

During the iterations, the algorithm goes through all data points and determines if for point p in cluster C_i, the centroid of C_i is the nearest reference point. If so, no adjustments are made and the algorithm proceeds to the next data point. However, if the centroid of cluster C_j becomes the reference point closest to the data point p, then p is reassigned to cluster C_j, the centroids of the loosing cluster C_i (minus point p) and the gaining cluster C_j (plus point p) are recomputed, and the reference points of clusters C_i and C_j are moved to their new centroids. After each iteration, every one of the K reference points is a centroid, or mean, hence the name "k-means". The iterations proceed until, for all data points, no re-allocation of points from cluster to cluster is possible. The C style description of the algorithm follows:

```
for j = 1 to K
  {
        R_j = {p}          /* p is a random point of (S | R) */
  }

for i = 1 to N
  {
        for j = 1 to K
          {
                D_j = d(R_j, p_i)
          }
        choose C_n where D_n = min {D}
        C_n = C_n U {p_i}
  }

for j = 1 to K
  {
        R_j = centroid of C_j
  }

do
  {
        Flag = FALSE
        for i = 1 to N
          {
                for j = 1 to K
                  {
                        D_j = d(R_j, p_i)
                  }
                choose C_n where D_n = min {D}
                if p_i ∉ C_n
                  {
                        Flag = TRUE
                        choose C_o where p_i ∈ C_o
                        C_o = C_o - {p_i}
                        C_n = C_n U {p_i}
                        R_o = centroid of C_o
                        R_n = centroid of C_n
                  }
          }
  } while (Flag)
```

Finally, the distribution of points will correspond to the centroidal Voronoi configuration, where each data point is closer to the reference point of its cluster than to any other reference point, and each reference point is the centroid of its cluster.

The algorithm is similar to the fitting routine, which begins with an initial guess for each fitted parameter and then optimizes their values. The algorithm runs in O(N*K) time.

The "k-means" algorithm does not guarantee the best possible partitioning, or finding the global minimum in terms of the error measure, but only provides a local minimum. However, the improvement of the partitioning and the convergence of the error measure to a local minimum is often quite fast, even when the initial reference points are badly chosen.

3.3.2 The "greedy" algorithm

Given S a set of N points and K the number of clusters, the algorithm chooses, in a greedy manner, a subset H of S consisting of K points that are farthest apart from each other and maintains for every point p of (S | H) the value dist(p) = min d(p, q), where q is a point of H and d(p, q) is the distance between p and q in the underlying metric (e.g. the Euclidian distance). Each point h of H determines a cluster, denoted C_i. A point p of (S | H) is assigned to cluster C_i if it is closer to h_i than to any other point in H. Note: (S | H) is a subset of S but not H, s and p are points of S, h is a point of H. The C style description of the algorithm follows:

$H = \{p_1\}$ /* p_1 is a random point of S */

$C_0 = S$

```
for i = 1 to N
  {
        dist(s_i) = d(p_i, p_1)
  }
```

```
for i = 1 to K-1
  {
        D = max {dist(p) where p is a point of (S | H)}

        choose h_i of (S | H) where dist(h_i) = D
        H = H U {h_i}

        for j = 1 to N
          {
                if d(s_j, h_i) <= dist(s_j)
                  {
                        dist(s_j) = d(s_j, h_i)
                        C_i = C_i U {s_j}
                  }
          }
  }
```

The running time of the algorithm is easily determined to be O(*N***K*). It has been later improved to O(*N**log*K*) by Feder and Greene [9].

It should be noted that all available clustering methods have their specific problems with regards to clustering image pixels. For instance, the split-and-merge algorithm is sensitive to image noise and both "greedy" and "k-means" assume that *k*, the number of clusters, is given prior to clustering, and that clusters have a hyper-spherical shape.

In the following section we suggest a clustering method that addresses these problems and provides real-time performance on clustering binary image pixels.

3.4 Suggested Method

Many types of data analysis, such as the interpretation of color images, involve data sets that often are excessively large so that their direct manipulation is impractical. To reduce the size of the data set some method of data compression or consolidation and region segmentation must first be applied without losing the essential character of the data and the context of the image.

Also, in some applications, attention has to be paid to image noise that may distract the clustering process. For this reason, noise filtering has to be implemented prior to or during the clustering process.

All consolidation methods sacrifice some detail and many segmentation methods misinterpret some regions. The most desirable methods have to be computationally efficient and yield results that are at least, for practical applications, representative of the original data and image context. With regards to the image noise, filtering should also be computationally efficient and avoid misinterpreting and altering the correct pixels.

We suggest the use of density and spatial cues for high-speed clustering in the presence of image noise. Thus our method falls into the general category of density techniques. Since all the algorithm steps outlined in the description below run in linear time proportional to the number of image pixels, the proposed algorithm also runs in linear time.

For illustration purposes, we used a simple setting shown in Figure 25: one big and two small red balls placed on the lab floor.

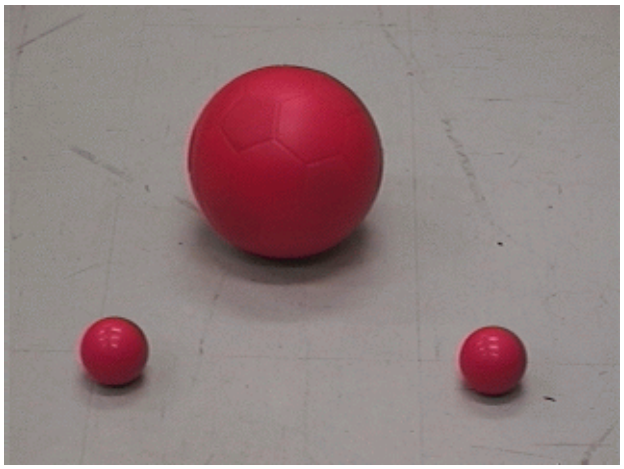

Figure 25. The original setting.

Color segmentation was applied to obtain a binary image shown in Figure 26. Segmentation highlighted the object pixels but also introduced severe noise which could be observed through the whole image.

Description of the suggested algorithm:

a) The binary image that for convenience may be assumed to be a set of pixels with values 0 and 1 is first split into adjacent *S* x *S* sample windows. For computational efficiency, *S* could be 2, 4, 8 or another number representing a power of 2. In our example, for a 320 x 240 binary image, *S* was chosen to be equal to 8, as seen in Figure 26. This choice provided real-time performance and enough detail of object shape.

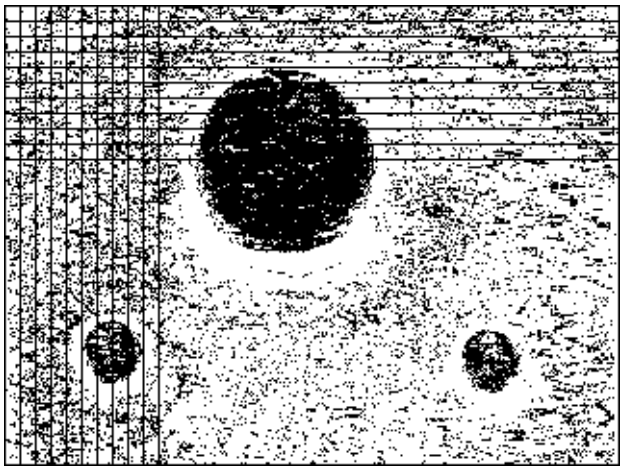

Figure 26. The binary image.

b) A density map is then created that reflects the density of object pixels in the original binary image. Every pixel in the density map corresponds to a particular sample window in the binary image. The value *V* of a pixel in the density map reflects the number *N* of object pixels in the sample window. The higher the number *N*, the bigger is the density pixel value *V* ($V=N/S^2$). In our example, the density pixel value is represented by the gray level, with black corresponding to the high density regions and white corresponding to the low density regions. The density map in its

original size of 40 x 30 pixels is shown in Figure 27 on the bottom left side. It should be noted that the density map size is S^2 times smaller than the original binary image. Such data compression significantly speeds up the clustering process, since only the density map pixels are used for further consideration. The density map that is enlarged to the size of the original binary image is shown in Figure 27 on the right side.

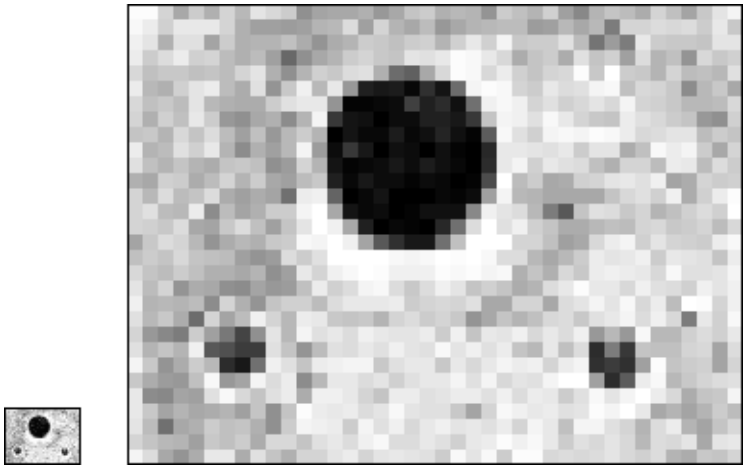

Figure 27. The density map.

c) By setting a threshold for the density we can eliminate low density pixels from further consideration. This is the first step in separating noise from objects, since image noise is usually sparse. Object pixels, however, are assumed to be dense. In our example the density threshold was set to 128 (50% of the gray scale). The density map with pixel values greater than 128 is shown in Figure 28.

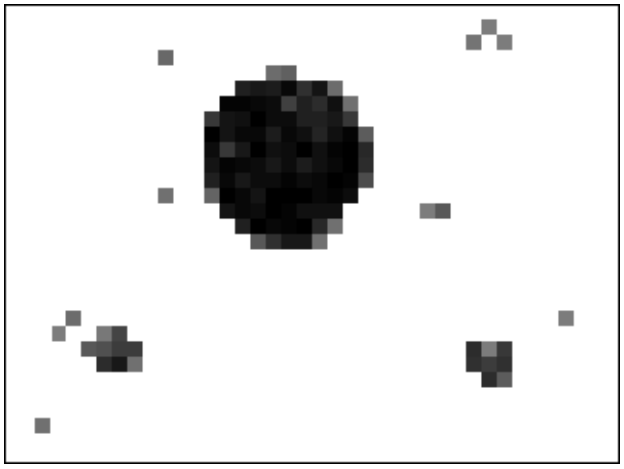

Figure 28. The density thresholding.

d) The next step in the algorithm involves clustering the remaining density map pixels by using an 8-neighbor approach that combines adjacent pixels into single separated clusters. The connected components algorithm [2] performs this task in linear time O(*N*), where *N* is the number of image pixels, in our case the number of density map pixels. Ten obtained clusters are shown in Figure 29.

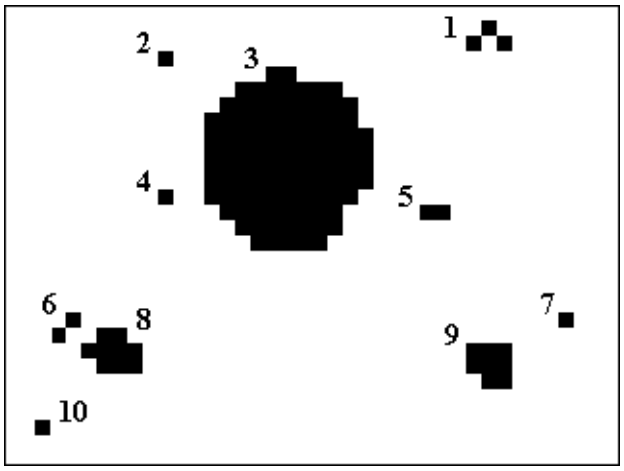

Figure 29. The 8-neighbor clustering.

e) The obtained clusters are then analyzed for their area. By setting the second threshold for the area we can eliminate spatially small clusters. This is the second step in separating noise from objects since image noise comes mostly in spatially small areas. Objects, however, are assumed to come in relatively spatially large areas. The value of the threshold is application specific and depends on various factors such as image scale and object pixel density. In our example, the threshold was set to 4 pixels (density map pixels) and the result of thresholding is shown in Figure 30.

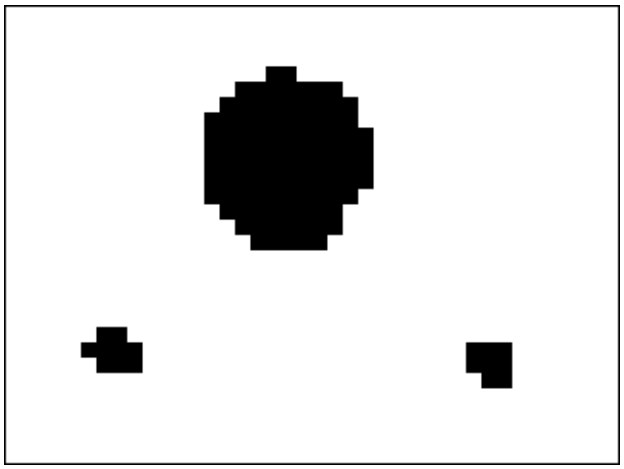

Figure 30. The area thresholding.

After these steps, compression to a density map, two thresholds and 8-neighbor clustering, the remaining clusters may be used in higher level analysis, e.g. for object recognition and object tracking.

Three objects found in the original image are shown in Figure 31. The bounding white boxes indicate the limits of the detected clusters. One may notice that severe noise present in the original image was eliminated during the clustering process and that the clusters are representative of the objects shapes.

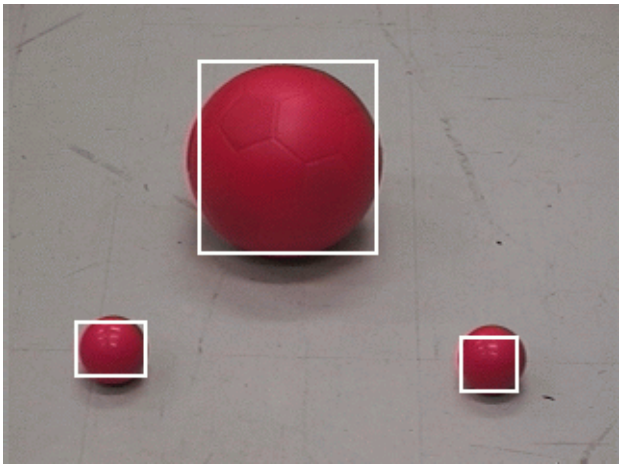

Figure 31. The clustering results.

A certain trade off between the size of a sample window and both thresholds has to be made. This trade off is application specific and depends on the image scale, nature of noise and other conditions. A problem of finding optimal values for these parameters is a challenging task and could be interpreted as a problem finding an optimal volume in a 3-dimensional space (size of a sample window, density threshold and area threshold) that provides best quality object segmentation. This problem is left for further investigation.

3.5 Results

We implemented the suggested clustering algorithm in our color object tracking system. All 320 x 240 binary images coming every 33 ms (NTSC - 30 frames per second) after color based segmentation were clustered in real time without dropping frames. Clustering each image using 8 x 8 sample windows takes 3-6 ms on an Intel Pentium II 233 MHz PC running under Linux. With no compression to a density map (1 x 1 sample windows), clustering takes 14-75 ms with multiple frame drops and low quality object representation. The results of clustering were successfully used further for object recognition and tracking objects with a pan/tilt camera.

Chapter 4

Tracking Objects with a Pan/Tilt Camera

4.1 Problem

To be able to track objects with a pan/tilt camera, objects must first be detected and recognized in the neighboring frames of the image sequence. Then, based on the objects' previous movement, their future movement can be predicted. This prediction determines the behavior of the camera. The problem is how to make camera movement smooth but at the same time reactive to the quick change in the object velocity and trajectory.

4.2 Grabbing Images

In the implemented tracking system we used a Sony EVI-D30 color CCD camera with an NTSC output via S-Video port to a Matrox Meteor frame grabber. The NTSC signal provided 30 image frames per second. Each frame consisted of two alternating fields - even and odd - each coming every 33 ms. The frame grabber was configured to grab even fields only and to convert them into RGB images. Odd fields were ignored to avoid pixel threading inherent to the interlace of the frame fields when objects move too fast. The grabbing process utilized DMA to work in parallel with the main control software running on a PC. The framegrabber continuously put RGB images into a designated shared memory buffer, so that when the control software required an image, it was already in memory. This arrangement provided an extremely high-speed image acquisition.

4.3 Identifying Objects

Every even filed of an NTSC sequence is treated as a separate image. After the field pixels have been clustered into separate objects, a problem arises regarding tracking these objects from frame to frame. Objects must first be identified, then based on their past behavior, future motion can be predicted.

For every object in the frame the following parameters were recorded:

a. area
b. center of mass
c. linear velocity (direction and value)
d. angular velocity (direction and value)
e. history (the number of times the object was consequently recognized from frame to frame)

To identify the same objects in neighboring frames the following assumptions were made:

a. the object area stays within ± 20% of its previous value
b. the object center of mass is located in the predicted vicinity

We call objects in the frame *t*-1, *parents*, and objects in frame *t*, *children*. To identify objects the following rule was applied: each parent claims a child based on the assumptions mentioned above. If conflict arises between parents in claiming a child, the argument is resolved in favor of the parent with the longest history. The unclaimed children reset their history counter and angular and linear velocities to zero. These objects are treated as having just appeared on the scene without previous history. The claimed children inherit their parents' history counter and increment it by one. They also calculate linear and angular velocities in accordance with their current position, and position and predicted movement of their respective parents.

4.4 Object Movement Prediction

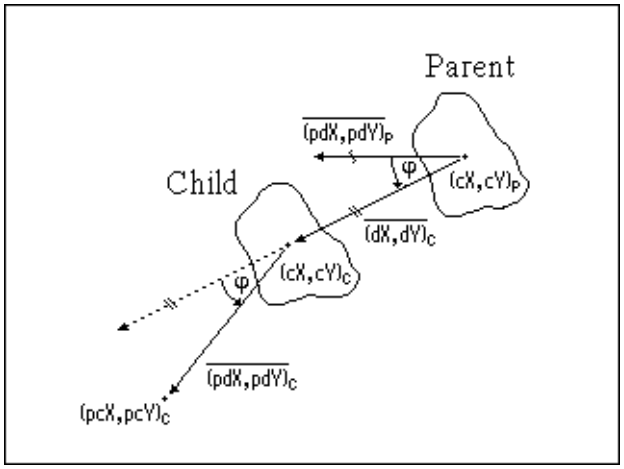

Figure 32. Object movement prediction.

The direction and value of the linear velocity for a child are calculated based on the centers of mass (*cX, cY*) of a child and a parent, as seen in Figure 32. The two dimensional linear velocity vector (*dX, dY*) is calculated as follows:

$$\begin{bmatrix} dX_c \\ dY_c \end{bmatrix} = \begin{bmatrix} cX_c \\ cY_c \end{bmatrix} - \begin{bmatrix} cX_p \\ cY_p \end{bmatrix}$$

Note: subscripts *c* and *p* stand for "child" and "parent"

Direction and value of the angular velocity vector for a child are calculated based on the linear velocity vector of a child and a predicted movement vector of a parent, as seen in Figure 32. In fact, for the angular velocity we calculate the *sin* and *cos* of φ - the *angle of rotation* (value), which can be positive or negative (direction). The calculations are based on the dot product of two vectors:

$a \cdot b = |a||b|\cos(\varphi)$

$\cos(\varphi) = a \cdot b / (|a||b|)$

$$\cos(\varphi) = (dX_c * pdX_p + dY_c * pdY_p) / \sqrt{((dX_c^2 + dY_c^2) * (pdX_p^2 + pdY_p^2))}$$

$$\sin(\varphi) = \sqrt{(1 - \cos^2(\varphi))}$$

The vector of predicted movement (*pdX*, *pdY*) for a child is calculated using a matrix of rotation and a linear velocity vector as follows:

$$\begin{bmatrix} pdX_c \\ pdY_c \end{bmatrix} = \begin{bmatrix} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{bmatrix} * \begin{bmatrix} dX_c \\ dY_c \end{bmatrix}$$

The predicted center of mass (*pcX*, *pcY*) for a child in frame *t*+1 is calculated using the current center of mass and the vector of predicted movement as follows:

$$\begin{bmatrix} pcX_c \\ pcY_c \end{bmatrix} = \begin{bmatrix} cX_c \\ cY_c \end{bmatrix} + \begin{bmatrix} pdX_c \\ pdY_c \end{bmatrix}$$

One can see from the above arrangement for parents and children that tracking is not a problem if objects are uniquely identified through the frame sequence and if the objects don't collide creating occlusions (two parents and one child), or separate (one parent and two children). These exceptions are handled in a simple manner using the history counter. However, a more elaborate artificial intelligence approach could be applied to solve this problem [39, 48].

4.5 Tracking Modes

There are two tracking modes implemented in the system.

In the first mode, the system tries to track multiple objects at the same time. The *agglomerative center of mass* is calculated for the predicted centers of mass of all objects. The camera tries to keep the agglomerative center of mass in the center of the viewed scene.

In the second mode, a particular object may be selected for tracking. The camera tries to keep the predicted center of mass of the selected object in the center of the viewed scene. Other objects in the image are detected but disregarded.

4.6 Camera Movement

After an agglomerative center of mass for multiple objects or a predicted center of mass of the selected object is calculated, the *compensation vector* (*comX, comY*) measured in pixels is calculated as the difference between the center of the image (*icX, icY*) and a particular center of mass, as shown in Figure 33. This vector reflects the amount of pan and tilt to be made by the camera to track the object(s).

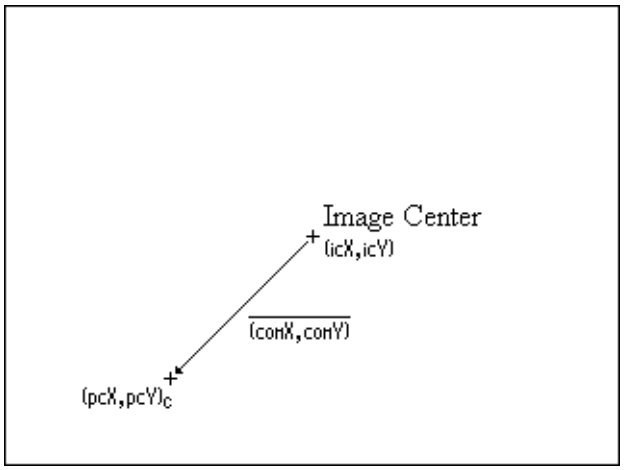

Figure 33. The compensation vector.

$$\begin{bmatrix} comX \\ comY \end{bmatrix} = \begin{bmatrix} pcX_c \\ pcY_c \end{bmatrix} - \begin{bmatrix} icX \\ icY \end{bmatrix}$$

The Sony EVI-D30 camera used for tracking has the following characteristics [44, 45]:

Pan range	200° in 1,760 increments (8.8 inc/deg)
Tilt range	50° in 600 increments (12.0 inc/deg)
Pan speed range	from 0 to 65 deg/sec (573 inc/sec) in 24 discrete steps
Tilt speed range	from 0 to 30 deg/sec (363 inc/sec) in 20 discrete steps
Zoom range	from f5.4 to f64.8 mm in 12 discrete steps

Note: experimentally measured pan/tilt speeds for particular discrete steps may be found in Appendix B.

The amount of camera pan and tilt depends on a compensation vector and a zoom factor of the camera. For a wide lens a small compensation vector requires a small amount of pan/tilt. However, for a telephoto lens the same compensation vector may require a large amount of pan/tilt. From experiments we found the coefficients and the formula that link together the zoom factor, the compensation vector and the corresponding amount of pan/tilt.

To control the camera, two modes called *static* and *dynamic* may be applied. In the static mode, a certain amount of pan/tilt movement at particular speeds could be requested from the camera. In the dynamic mode, pan and tilt speeds could be set to move the camera in a specific direction continuously. In this mode the camera is in a state of continuous movement with pan and tilt speeds changing from zero to their respective limits.

In our tracking system we used the dynamic mode to track object(s). This mode provided smooth but at the same time reactive movement of the camera. The following formula was used to determine pan and tilt speeds:

$Speed_{Pan}$ # = *comX* * 135 / *Zoom_Factor*

$Speed_{Tilt}$ # = *comY* * 184 / *Zoom_Factor*

Vector (*comX, comY*) is calculated as described above for every single frame. The *Zoom_Factor* may be obtained directly from the camera by sending an inquiry via an RS232 port. This request should be made every time the camera zoom is changed manually or by the controls software. The ratio of 184/135 reflects the difference in increments per degree for pan and tilt. It is approximately a ratio of 3/4. The exact numbers 135 and 184 were found experimentally to accommodate for the zoom factor of the camera.

If, after calculating $Speed_{Pan}$ and $Speed_{Tilt}$ numbers, the speeds exceed camera limits (24 for pan and 20 for tilt), they are linearly scaled to fit their limits, preserving the direction of movement. Although the plot in Appendix B demonstrates a non linear change of the actual speed vs speed number, the dependence in approximation may be assumed linear, since the change is insignificant. A small look-up table may be utilized to guarantee a perfect linearity, if required.

The speeds are updated every frame, i.e., 30 times a second, in the controls software, but only 4-5 times a second in the camera. This happens due to the delayed response of the camera to the control commands. Also, we did not take into consideration the camera speed while calculating predicted object movement since this movement is calculated within very small intervals of time, and camera movement within these intervals is negligible compared to the object movement.

One may suggest utilizing a filter, for example a Kalman filter [3], to make the movement of the camera smooth. We did not use the filter for two main reasons. First, the camera movement is very smooth without introducing a filter and the filter would only add extra computational cost. Second, if the filter is introduced, the camera response to a quick change of object speed and trajectory would be delayed and such behavior as the ball bouncing from the floor or from the walls would not be adequately modeled.

4.7 Lost and Found Strategy

Since the camera pan and tilt speeds are limited to a particular range, there may be situations when the camera movement is too slow to follow a fast moving object. To solve this problem, when the camera is in the tracking mode the following *lost and found strategy* is utilized: when the selected object is lost, the camera tries to follow for 5 seconds in the direction and with the speed it moved before losing the object. If the follow-search yields no results the camera returns to the position where the object was lost, waiting for it to emerge.

4.8 Results

The implemented tracking system consisting of a Sony EVI-D30 CCD color camera providing an NTSC signal via S-Video connection, a Matrox Meteor NTSC frame grabber, and a PC with an Intel Pentium II - 233 MHz processor running Linux provided tracking of single and multiple objects in real time. It takes only 2-3 ms to identify objects, predict their movement, and send a command to the camera to follow the object(s). The camera movement was smooth but at the same time reactive to the quick changes in the object dynamics.

Chapter 5

Experimental Results

As a result of our research we developed a tracking system capable of tracking multiple objects from an NTSC frame sequence in real time. The layout of the tracking system is shown in Figure 34.

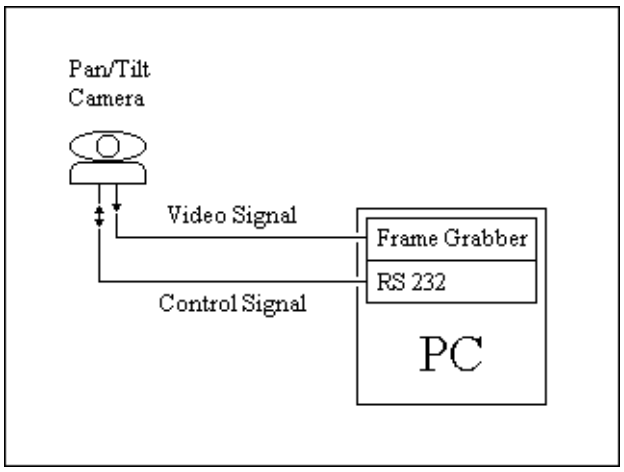

Figure 34. The layout of the tracking system.

Without using any sophisticated special purpose equipment we achieved highly acceptable performance. It takes only 25-26 ms to grab a frame, display it on the screen, process the frame image, identify objects and move the camera. The results of tracking are showed in Figure 35 where the red ball is rotated fast enough, sometimes with acceleration, in front of a still camera to produce a blurred image of the object.

It has to be admitted that at such speeds hardly any other method than one based on color would be infallible. Our tracking method proved to be very reliable in these instances. It is observed that

the object is identified with good accuracy (white pixels and the bounding box) and that the prediction of the object movement (vector from the object center) is quite precise. It is also observed from this experiment that due to the fact that we compensate for angular velocity the predicted object position does not get off the circular trajectory.

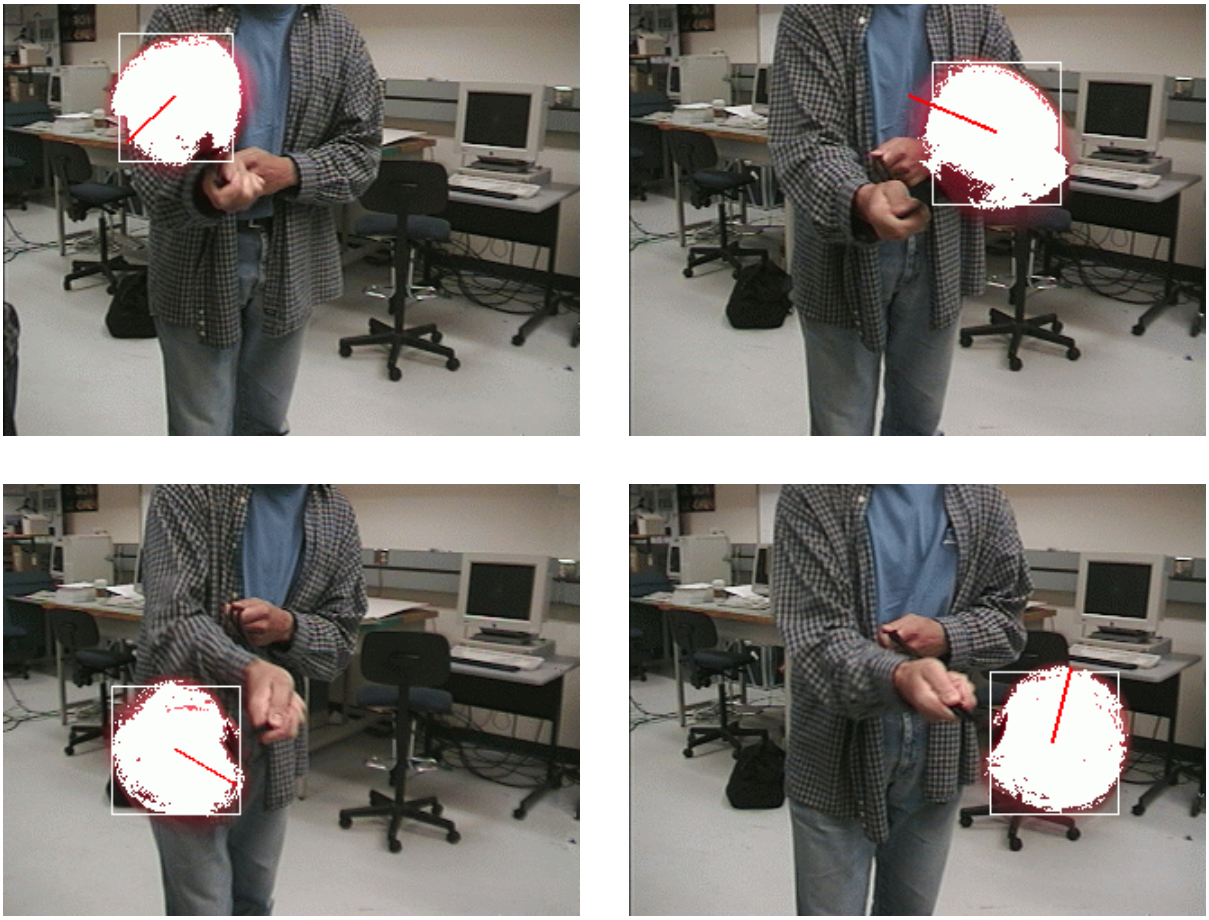

Figure 35. Tracking the red ball.

Chapter 6

Conclusion

In our research we addressed the question of building an active tracking system capable of tracking multiple objects from an NTSC frame sequence in real time. We achieved highly acceptable real-time performance in tracking color dielectric objects by utilizing the dichromatic reflection model to represent object color at various illumination and viewing conditions, and by using density and spatial cues to cluster object pixels into separate objects. We identified objects from the neighboring frames and predicted their future movement based on their previous history. In the implemented tracking system we achieved both a smooth movement of the camera and an instant reaction to the change of object speed and trajectory.

Although the results obtained are very satisfactory, there is always potential for improvement and the challenge for further investigation. We mentioned the specifics of these in the respective chapters. Here we would only like to add that restricted by real-time constraints, we used a simple but efficient method for detecting and recognizing objects. However, there is potential left for a more elaborate artificial intelligence approach, for example, a spatial reasoning technique [32] used to segment objects in the image. In case time limits are exceeded due to the additional computational cost, the smart buffering techniques [29] may be utilized to provide the parallel synchronous execution of the sensing, processing and reaction tasks.

To finalize this work we would like to point out that the tracking system as well as its separate functional components proved to be of practical interest to researchers and engineers in the fields of video conferencing, remote sensing and robot soccer. The positive feedback from these communities was the greatest reward for the time and effort spent.

Bibliography

[1] P.K. Agarwal, C.M. Procopiuc, Exact and Approximation Algorithms for Clustering, Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms, 1998

[2] A.V. Aho, J.D. Ullman, Foundations of Computer Science, Computer Science Press, N.Y., pp.481-482, 1992

[3] B. Anderson, J. Moore, Optimal Filtering, Prentice-Hall, NJ, 1979

[4] M. Black, A. Jepson, EigenTracking: Robust Matching and Tracking of Articulated Objects Using a View-Based Representation, Proc. ECCV 96, Vol I, pp 329-342

[5] P.S. Bradley, Usama M. Fayyad, Refining Initial Points for K-Means Clustering, Proceedings of the Fifteenth International Conference on Machine Learning, Morgan Kaufmann, 1998

[6] R.O. Dabes, A.K. Jain, Algorithms for Clustering Data, Prentice Hall, 1988

[7] B. Everitt, Cluster Analysis, Heinemann Educational Books, 1974

[8] V. Faber, Clustering and the Continuous k-Means Algorithm, Los Alamos Science Magazine, Number 22, 1994

[9] T. Feder, D. Greene, Optimal algorithms for approximate clustering, Proc. 20th Annual ACM Sympos. Theory Comput., 1988

[10] P. Fieguth, D. Terzopoulos, Color Based Tracking of Heads and other Mobile Objects at Video Frame Rate, Proc. CVPR 97, pp. 21-27

[11] J.D. Foley, A. Van Dam, S.K. Feiner, J.F. Hughes, R.L. Philips, Introduction to Computer Graphics, Addison-Wesley, 1997

[12] E. Forgy, Cluster analysis of multivariate data: efficiency vs interpretability of classifications, Biometrics, 1965

[13] K. Fu, R. Gonzalez, C. Lee, Robotics, Control, Sensing, Vision and Intelligence, McGraw-Hill, NY, 1987

[14] B. Funt, V. Cardei, K. Barnard, Learning Color Constancy, IS&T 4th Color Imaging Conference, 1996

[15] B. Funt, K. Barnard, L. Martin, Is Machine Color Constancy Good Enough ?, Proc. ECCV 98, Vol. II, pp. 445-459

[16] R. Gonzalez, R. Woods, Digital Image Processing, Addison Wesley, 1992

[17] T. Gonzalez, Clustering to minimize the maximum inter-cluster distance, Theoretical Computer Science #38, 1985

[18] G. Hager, K. Toyama, XVision: Combining Image Warping and Geometric Constraints for Fast Visual Tracking, Proc. ECCV 96, Vol. II, pp 507-517

[19] G. Hager, P. Belhumeur, Real Time Tracking of Image Regions with Changes in Geometry and Illumination, Proc. CVPR 1996, pp. 403-410

[20] J. Hartigan, Clustering Algorithms, John Wiley & Sons, 1975

[21] D. Hearn, M.P. Baker, Computer Graphics, Prentice Hall, 1994

[22] R. Herpers, G. Verghese, M. Jenkin, E. Milios, A. Jepson, J.K. Tsotsos, SAVI: An Actively Controlled Teleconferencing System, Proc. Second IEEE Workshop on Perception for Mobile Agents, 1999

[23] M. Inaba, H. Imai, M. Nakade, T. Sekiguchi, Application of an Effective Geometric Clustering Method to the Color Quantization problem, 13th ACM symposium on Computational Geometry, 1997

[24] Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference, Intel, 1997

[25] M. Isard, A. Blake, Condensation - conditional density propagation for visual tracking, Int. J. of Computer Vision, 1998

[26] L. Kaufman, P.J. Rousseeuw, Finding Groups in Data: An Introduction to Cluster Analysis, John Wiley & Sons, 1990

[27] G.J Klinker, A physical approach to Color Image Understanding, PhD dissertation, Carnegie Mellon University, Pittsburgh, Pa, 1988

[28] G.J. Klinker, S.A. Shafer, T. Kanade, The Measurements of Highlights in Color Images, Int. J. Comp. Vision 2(1), pp. 7-32, 1988

[29] J. Little, J. Kam, A Smart Buffer for Tacking Using Motion Data, Proc. CAMP, 1993

[30] S.P. Lloyd, Least squares quantization in PCM. IEEE Transactions on Information Theory IT-28, 1982

[31] J. MacQueen, Some methods for classification and analysis of multivariate observations, Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability. Volume I, Statistics. University of California Press, 1967

[32] T. Matsuyama, T. Wada, Cooperative Spatial Reasoning for Image Understanding, International Journal of Pattern Recognition and Artificial Intelligence, Vol. 11, # 1, pp. 205-227, 1997

[33] C.L Novak and S.A. Shafer, Color Vision. In Encyclopedia of Artificial Intelligence, pp. 192-202, 1992

[34] C.L Novak, S.A. Shafer, R.G Willson, Obtaining Accurate Color Images for Machine Vision Research, Proc. SPIE 1250, pp. 54-68, 1990

[35] N. Oliver, A. Pentland, F. Berard, LAFTER: Lips and Face Real Time Tracker, Proc. CVPR 1997, pp. 123-129

[36] W. Ostwald, Colour Science, Winsor & Winsor, London 1931

[37] Y. Raja, S. McKenna, S. Gong, Segmentation and Tracking Using Colour Mixture Models, Proc. ACCV 98, Vol. I, pp 607-614

[38] Y. Raja, S. McKenna, S. Gong, Object Tracking Using Adaptive Colour Mixture Models, Proc. ACCV 98, Vol. I, pp 615-622

[39] D. Reid, An Algorithm for Tracking Multiple Targets, IEEE Trans. Auto. Control No. 6, pp 843-854, 1979

[40] S. Schroter, Lookup-Table Calibration for Efficient Colour Image Segmentation, Forschungsbericht 666/1998, Fachbereich Informatik, Universitat Dortmund, 1998

[41] S.A. Shafer, Describing light mixtures through linear algebra, J. Opt. Soc. Amer. 72(2), pp. 299-300, 1982

[42] S.A. Shafer, Using Color to Separate Reflection Components, Color Res. App. 10(4), pp. 210-218, 1985

[43] A.R. Smith, Color Gamut Transform Pairs, Proc. SIGGRAPH 78, pp. 12-19, 1978

[44] Sony EVI-D30/EVI-D31 Operating Instructions, Sony Corporation, 1996

[45] Sony EVI-D30/EVI-D31 VISCA/RS232C Command List Version 1.0, Sony Corporation, 1996

[46] J.M. Tenenbaum, T.D. Garvey, S. Weyl, H.C. Wolf, An interactive Facility for Scene Analysis Research, Technical Report TN 87, SRI International, Menlo Park, California, 1974

[47] S.Tominaga, B.A. Wandell, Standard surface reflection model and illuminant estimation, J. Opt. Soc. Am. A 6, pp. 576-584, 1989

[48] V. Tucakov, Interpreting Severe Occlusions in Region-Based Stereo Vision, M.Sc. Thesis, Dept. of Computer Science, UBC, 1998

[49] C. Vieren, F. Cabestaing, J. Postaire, Catching Moving Objects with snakes for Motion Tracking, Pattern Recognition Letters No. 7, pp.679-685, 1995

[50] C. Wren, A. Azerbayejani, T. Darrell, A. Pentland, Pfinder: Real-Time Tracking of the Human Body, SPIE Vol. 2615, 1995

Appendix A

Assembly Code

Please note that the assembly code for the Intel PC processor [24] is written in the GNU notation: with the *Source* on the ***Left***, and the *Destination* on the ***Right***.

A.1 The color resolution reduction algorithm

```
   // zero EBX
   // zero EDX

"xor       %%ebx, %%ebx \n\t"
"xor       %%edx, %%edx \n\t"

   // load Pixel into EAX
   // B in EBX
   // G in EDX
   // R in EAX

"mov    %(Pixel), %%eax \n\t"
"mov        %%al, %%bl  \n\t"
"mov        %%ah, %%dl  \n\t"
"shr         $16, %%eax \n\t"

   // CL = 8
   // CL = 8 - (current bpc)

"mov          $8, %%cl  \n\t"
"sub      %(bpc), %%cl  \n\t"
```

```
   // reduce resolution in R
   // reduce resolution in G
   // reduce resolution in B

"shr        %%cl, %%al  \n\t"
"shr        %%cl, %%dl  \n\t"
"shr        %%cl, %%bl  \n\t"

   // CL = current bpc
   // align G
   // CL = CL * 2
   // align R

"mov      %(bpc), %%cl  \n\t"
"shl        %%cl, %%edx \n\t"
"shl          $1, %%cl  \n\t"
"shl        %%cl, %%eax \n\t"

   // combine G & R
   // combine B & G & R

"or         %%edx, %%eax \n\t"
"or         %%ebx, %%eax \n\t"
```

A.2 The LUT access algorithm

```
   // copy Pixel into EBX
   // load 3 into CL
   // EAX = Pixel  /  8
   // EBX = Pixel mod 8

"mov    %(Pixel), %%eax \n\t"
"mov       %%eax, %%ebx \n\t"
"mov          $3, %%cl  \n\t"
"shr         %%cl, %%eax \n\t"
"and          $7, %%ebx \n\t"
```

```
   // EAX = &LUT + EAX

"add     %(&LUT), %%eax \n\t"

   // DL = LUT byte
   // AL = 00000001
   // CL = Pixel mod 8
   // AL = AL <- CL, create a mask
   // apply a mask to the LUT byte
   // store a LUT bit in the Result

"mov     (%%eax), %%dl  \n\t"
"mov          $1, %%al  \n\t"
"mov        %%bl, %%cl  \n\t"
"shl        %%cl, %%al  \n\t"
"and        %%dl, %%al  \n\t"
"mov   %%al, %(Result)  \n\t"
```

Appendix B

Experimentally measured pan/tilt speeds of the Sony EVI-D30 camera

The plot of *Speed #* vs *Speed [deg/sec]* is shown in Figure 36.

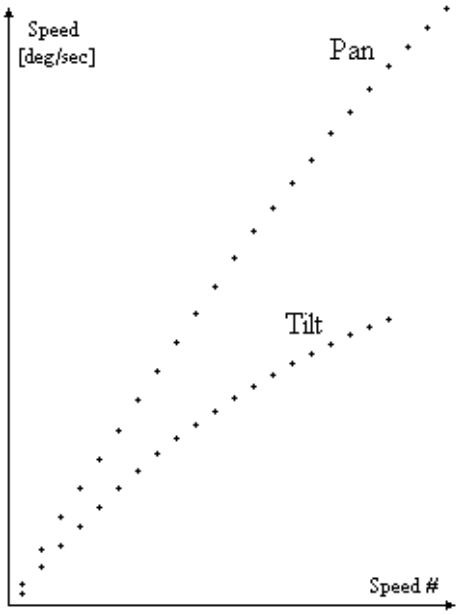

Figure 36. Sony EVI-D30 pan/tilt speeds.

PAN (1,600 increments)				TILT (500 increments)			
Speed #	Time sec	Speed inc/sec	Speed deg/sec	Speed #	Time sec	Speed inc/sec	Speed deg/sec
1	53.8	29	3	1	17.0	29	2
2	27.0	59	7	2	8.6	57	5
3	18.2	88	10	3	5.9	85	7
4	13.7	116	13	4	4.5	110	9
5	11.1	144	16	5	3.7	134	11
6	9.3	171	19	6	3.2	157	13
7	8.1	198	22	7	2.8	179	15
8	7.1	224	25	8	2.5	199	17
9	6.4	250	28	9	2.3	219	18
10	5.8	275	31	10	2.1	237	20
11	5.3	299	34	11	2.0	251	21
12	4.9	323	37	12	1.9	266	22
13	4.6	348	40	13	1.8	281	23
14	4.3	371	42	14	1.7	298	25
15	4.1	392	45	15	1.6	307	26
16	3.9	415	47	16	1.6	321	27
17	3.7	436	50	17	1.5	332	28
18	3.5	459	52	18	1.5	344	29
19	3.3	480	55	19	1.4	355	30
20	3.2	498	57	20	1.4	363	30
21	3.1	519	59				
22	3.0	536	61				
23	2.9	555	63				
24	2.8	573	65				