AN AUTOMATIC METHOD FOR RED-EYE DETECTION AND CORRECTION IN DIGITAL IMAGES

by

Exam Roll: Curzon Hall - 0513

Registration No: HA - 1411

Exam Roll: Curzon Hall - 0557

Registration No: HA - 1189

4^{th} Year (Honors)

Session: 2008–2009

A project submitted in partial fulfilment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UNIVERSITY OF DHAKA

July 2013

Abstract

Red-eye is a common photographic artifact where the eyes of the subject appear red instead of their natural color. It is caused by the reflection of flash light used in low-light photography bouncing back from human retina into the camera. This project presents state of the art automatic red-eye detection and correction algorithm based on computer vision and pattern recognition algorithms. The proposed algorithm consists of two major steps: red-eye detection and red-eye correction where detection is the most challenging phase. Face regions are extracted from input image using skin-based segmentation. Iris positions are then tracked using edge detection and circular Hough transform algorithms. Finally the detected iris regions are searched for red pixel clusters to confirm the existence of red-eyes and corrected accordingly. The system is able to detect and correct red-eye from images with complex background and multiple faces. The algorithm is also invariant of different face positions, orientations and light intensity. The proposed system makes major contribution in red-eye detection and removal research and presents a new solution with acceptable accuracy rate. An image database consisting of 50 images of different resolutions with red-eye effect has been used to test our proposed algorithm. The average red-eye detection and correction accuracy rate achieved by the proposed method is above 90%.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Description

Red-eye effect is a very common photographic phenomenon especially occurs when using compact, consumer-oriented cameras. The pupil and sometimes the whole iris of subject's eye appears red instead of its natural color because of this artifact. This problem has become more prevalent in recent years with the advent of digital cameras and other electronic devices with camera (e.g. tablets, smarthphones, handheld gaming consoles and so on).

Figure 1.1: Red-Eye Problem

Red-eye problem mostly occurs because of the flash illumination used when capturing photos in dark. It is caused by light from the flash unit entering the pupil, multiply reflecting off the retina and finally exiting back through the pupil. Since the light that enters into the pupil is partially absorbed by capillaries of retina, the pupil appears red instead of its natural color in captured images. It's apparent from the mechanism that, the probability of appearing red-eye in image increases with the flash unit of camera being closer to the optical axis of the camera lens. Therefore, red-eye is usually observed in images captured by a small camera with an integral flash light unit.

Figure1.2 shows how the angle of reflection causes red-eye problem. The red-eye cone shines from the flashed eye back at the flash with an angle α. Its color is caused by the reflection of the flash off the blood vessels of the retina; the camera will record this red hue if the angle β between the flash source and the camera lens is not greater than α.

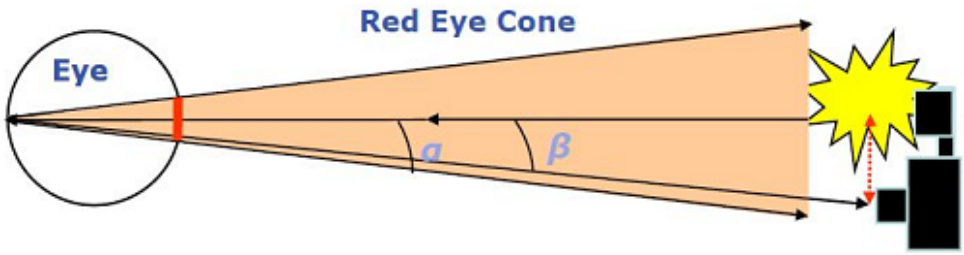


FIGURE 1.2: Angle of Reflection

Most of the recent cameras come with built-in red-eye reduction system and there are also a few softwares such as Adobe Photoshop, Apple iPhoto, Google Picasa etc. that provide red-eye removal feature. All these methods reduce the probability of red-eye phenomenon but most of them requires user intervention to localize red-eye region and don't provide 100% accuracy all the time. For these reasons, an automatic red-eye detection and removal method is very much desired that provides higher accuracy than the current systems.

1.2 Motivation towards the Work

The aim of the project is to develop a fully automatic computer vision based red-eye detection and correction system. Major objectives of this project are as follows:

- To extract face region from an image using skin-based segmentation
- To track iris positions in the extracted face area
- To remove false alams from the detected possible iris positions
- To search for red-eyes in the detected iris regions
- To correct red-eyes properly

1.3 Proposed System's Overview

The proposed automatic red-eye detection and correction system was developed in Eclipse IDE with g++ compiler on Mac OS X and Linux (Ubuntu) platform. The system proposed in this project has advantages over other red-eye detection and correction methods in terms of speed, simplicity and robustness. It doesn't use any learning feature so it doesn't need any preprocessing on any sample data which saves time and reduces overall computational complexity. The system is also fast because it is written completely in C++. It has two major components:

- Red-Eye Detection and
- Red-Eye Correction

The detection phase is most challenging and much longer than the correction phase. At first, the system reads the input image from memory and extracts the face region from the image using skin-based segmentation. Then edge detection algorithm is applied on the face image to retrieve edge information. After computing edge information, circular Hough transform algorithm is used to track iris

positions. The detected iris positions are searched for red-eye regions to ensure the presence of red-eyes in input image. Finally, a correction algorithm is applied if red-eye is found during the confirmation stage. The proposed system also handles memory management issues very carefully as the program needs to access and allocate a lot of memory space during processing.

1.4 Overview of the Book

This book is organized into eight chapters labeld as Introduction, Related Works, Search Space Reduction, Iris Tracking, Red-Eye Confirmation, Red-Eye Correction, Experiments, Results and Discussion, and Conclusion and Further Research. The chapters cover the following topics:

Chapter 1 introduces the red-eye problem in digital images and discusses the motivation of this work. A brief description of the working procedure of the proposed system is also presented in this chapter.

Chapter 2 presents a survey of current research literature on this topic. Brief and terse description of most prominent red-eye detection and correction techniques are discussed in this chapter.

Chapter 3 focuses on search space reduction which is a very common factor to speed up the whole system. It describes how the system dramatically reduces search space using skin-based segmentation.

Chapter 4 is the longest chapter of this book and presents the technique of iris tracking from the face image obtained from search space reduction process described in chapter 3. This chapter sheds some light on different edge detection algorithm at first and then it presents the circular Hough transform which is used to find circular shape in given image.

Chapter 5 discusses how a red-eye is confirmed from the detected irises and how other false alarms are discarded.

Chapter 6 describes the red-eye correction procedure.

Chapter 7 presents experimental results and detailed discussion on the achieved results.

Chapter 8 concludes this dissertation mentioning achievements and current limitation of the work. It also outlines some possible directions for future improvement of the proposed system.

Chapter 2

Related Works

2.1 Red-Eye Detection

Red-eye detection and correction has been studied by so many researchers for the past few years and the first step of all solutions proposed so far is red-eye detection. Detection is much more challenging and a difficult task because of the different image size, resolution, different pose, face orientation and lighting condition. However, different researchers have adopted different approaches to solve this problem. The researchers have categorized the solutions proposed so far into three major categories: feature-based object detection approaches, statistical learning based approaches and neural network based approaches. Many of these approaches use robust face detection which is itself a critical research problem and requires good amount of computation.

2.1.1 Feature Based Object Detection Approaches

The most recent and widely accepted red-eye solution was proposed by Hewlett-Packard Labs, consisting of a two module procedure (red-eye detection and correction), is described in the work of Luo *et al.* [8, 13]. The redeye detection part

is modeled as a feature based object detection problem, which uses *Adaboost* algorithm to simultaneously select relevant features and assign feature weight for the classifier. A systematic, orientation independent feature computation scheme for object detection is adopted, which can detect red eyes of arbitrary orientation.

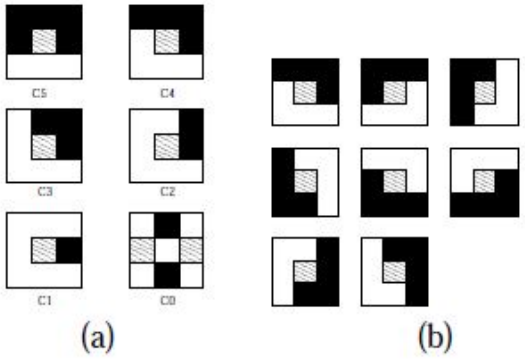


FIGURE 2.1: Circular Templates Used by HP Labs

2.1.2 Statistical Learning Based Approaches

Wan et al. [11] have developed a method based on Active Appearance Models (AAM). AAM is a statistical approach which can model a deformable object shape. Joining color information and a deformable model they locate red eyes as deformable objects. Finally a candidate region is considered as red-eye if the feature vector result tends to be positive.

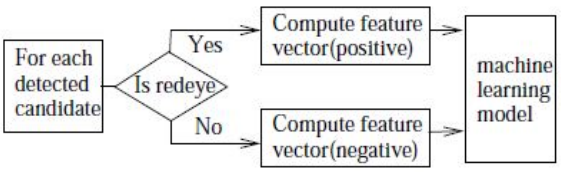


FIGURE 2.2: Machine Learning Model used by Wan

Volken *et al.* [12] base their work on image processing and heuristics. They first look for the red zones of the whole image, then estimate the probability of each of these zones being a red eye by evaluating their roundness, the amount of white around these zones, corresponding to the sclera and the amount of skin.

In the method developed by Zhang *et al.* [7] they find pixels of the whole image that may be marked as a red color or non-skin color pixel. To restrict red color areas, the algorithm proceeds to remove invalid regions by size and brightness restrictions and checks the surrounding non-skin pixels. Finally, an eye classifier is utilized to confirm each candidate region found in the previous steps. Only those regions confirmed by the classifier as human eyes will be passed to the auto-correction stage.

Several approaches use skin tone to limit a posteriori the false positive red eye candidates in a map of red patches. Luo *et al.* [13] after a preliminary screening where global candidate red eye areas are detected mainly on the base of redness and contrast, perform several local verification tests. In particular, each candidate red eye area that does not correspond to an isolated non-skin tone 'island' in a skin tone region is filtered from the final red eye map.

Slightly different from the schemes presented, Zhang *et al.* [7] adopted a heuristic algorithm looking for a group of candidate red regions. An eye classifier is then utilized to confirm whether each candidate region is a human eye or not.

2.1.3 Neural Network Based Approaches

Gasparini and Schettini *et al.* [21] have designed a modular procedure for automatic detection and correction of the redeye effect, specifically designed for uncalibrated images, such as images acquired by unknown systems under unknown lighting conditions. Firstly they apply an adaptive color cast removal algorithm to correct the color photo. This phase not only facilitates the subsequent steps of processing, but also improves the overall appearance of the output image. Then the method looks for redeye within the most likely face regions, obtained combining a color-based skin detector with a face detector based on a multi-resolution neural network.

Wang and Zhang [8] used neural network to detect faces and eyes. Gaubatz and Ulichney [10] reduce the search space looking for faces with the aid of a cascade of multi-scale classifiers [15, 16]. Hardeberg *et al.* perform a preliminary color segmentation based on thresholding in color spaces to locate the skin regions US patent, [24].

2.1.4 Other Approaches

There are some other approaches for solving the red-eye problem like: Willamowski and Csurka *et al.* [1], have developed a probabilistic approach based on stepwise refinement of a pixel-wise red eye probability map. The correction step applies a soft red eye correction based on the resulting probability map.

On the other hand, Marchesotti *et al.* in particular, have closely considered the problem of red eye correction, [25]. They have proposed three correction methods, evaluating what they have called their image degradation risk and their expected perceptual quality improvement. An adaptive system is applied to select the correction strategy dependent on these measures and on the red eye detection confidence.

Luo et al. also has focused on some color space based solution [13]. He defined two redeye color surfaces in the CIE Lab color space, a high contrast color surface and a low contrast color surface. The high contrast boundary curves are used to aggressively separate redeye color pixels from non redeye color pixels. The low contrast boundary curves are used in a less aggressive segmentation, in order to accommodate cases of redeye pixels somehow merged within non-redeye pixel areas (e.g. regions of pale or reddish skin). Both high and low contrast surfaces are used to detect candidate redeye pixels, and the successive segmentation and validation processes reduce the false alarms. They have also observed that the process of detecting red eyes using multiple redeye color models and/or a redness map, and merging the results, frequently improves the overall redeye detection accuracy.

2.2 Red-Eye Correction

Once the red eyes are detected, a color correction of the artifact is applied. It seems that this part of the problem is easier than the detection step. Actually several aspects must be taken into account to avoid degradation of the original images. The main guidelines in color correction are:

- Preserve the glint, the specular reflection of the flash in the eye, that gives the eyes a natural aspect;
- Correct both red eyes (if present) with the same strength and color;
- Avoid abrupt variation between the corrected and the uncorrected areas.

In most of the approaches, if a pixel has been detected as belonging to a redeye, it is replaced with a substantially monochrome pixel. Several strategies more or less complicated, can be adopted to obtain this natural correction.

In some other approaches, the researchers have attempted to decrease the level of the red colour by placing greater emphasis on the green and blue channels of the RGB model. The technique tries to reduce the energy of the red component by replacing the value with an average of the other two components. Since the pupil is dark and blackish in color, the idea was to perform several changes to the pixel values to ensure that there was no particularly dominant value in a channel. The technique is essentially darkening the pupil colour. The deviation of the red, green, and blue channels provides for a nice variation of colour over the pupil.

The simplest approach was performed by Wu [4], where the red color of the defected eyes is simply substituted by black. Corrections of this type could be very dangerous leading to a processed image which is even worse than the defective original.

Patti *et al.* [5] perform another simple neutral correction, assigning to the red area a gray value of 80% of the original luminance. With this scaling factor experimentally evaluated, the glint in the pupil is preserved.

Usually the correction is done by applying a weighted desaturation to smooth the perceived edges due to hard decision boundaries, Gaubatz *et al.* [14].

This can be done with the aid of a smoothing mask as suggested by Smolka *et al.* [15] to achieve a softer correction that should appear more natural.

An example of a possible monochrome correction, as reported in [29] is:

$$R_{new} = R_{old} \times (1 - Mask_{smooth}) + Mask_{smooth} \times R_{mch}$$
$$G_{new} = G_{old} \times (1 - Mask_{smooth}) + Mask_{smooth} \times G_{mch}$$
$$B_{new} = B_{old} \times (1 - Mask_{smooth}) + Mask_{smooth} \times B_{mch}$$

The coordinates of the monochrome pixel are R_{mch}, G_{mch} and B_{mch}, evaluated considering the intensity equal to the mean of (G, B) and the color correction is weighted with the smoothing mask ($Mask_{smooth}$). The correction mask can be considered as the map of the probability that a certain pixel belongs to a red-defect region or not. Pixels approaching to the eye boundaries receive a gradually decreasing probability, allowing for a smooth change between correct and uncorrected regions.

Luo *et al.* [13] propose a method to detect and correct redeye areas starting from different resolutions of the original image. In particular, the correction is adapted to the prescribed output resolution. For example, in a printer system application, the detected eye could be scaled up from a thumbnail size of about 384 pixels in width by 288 pixels in height, to a print image size of 1800 pixels in width by 1200 pixels in height.

They propose strategies to compensate errors that might occur as a result of the inaccuracy inherent in the quantization process involved in mapping areas from

small resolution to higher resolution. To refine the final area, if necessary, they propose a further pixel classification based on skin tone analysis.

Red-eye pixels are then corrected desaturating and darkening the original color. The darkening factors are computed based on luminance values of the input image pixels. These factors decrease with the darkness level of the pixels; that is, lower luminance values (i.e. darker pixels) are associated with lower darkness factors. Moreover, red-eye pixels near the center of the redeye correction region are assigned higher weights than redeye pixels near the boundaries.

2.3 Summary

In this chapter we have discussed several of the most prominent solutions to the red-eye problem that have already been proposed by researchers and many of these solutions are currently being used in various applications. Brief explanation of several red-eye problem solutions were presented with appropriate references. Red-eye detection and correction is an active research area and numerous research works are going on to achieve better performance in terms of speed and accuracy. A clear understanding of existing solutions helped us to identify critical considerations of the problem when we designed our system. We presented a comprehensive description of our proposed method in the next chapters.

Chapter 3

Proposed Red-Eye Problem Solution

3.1 Introduction

Our red-eye detection and correction system consists of five major phases:

- Facial Region Extraction
- Edge Detection
- Iris Tracking
- Red-Eye Confirmation
- Red-Eye Correction

The following illustration shows the steps sequentially:

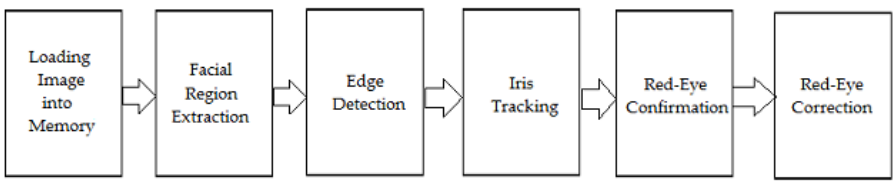


FIGURE 3.1: Red-Eye Detection System Architecture

In this chapter we present all these steps in detail along with some basic concepts of digital image processing.

3.2 Skin-Based Segmentation

In order to find red-eye pixels we will have to search through the whole image. We have already seen that the dimension of an image is the number of pixels it contains so it can be both time and memory consuming process to conduct searching for red-eye pixel through the whole image. To overcome this performance hindrance, we need to reduce our search space in a great extent since an eye occupies only a very small region in comparison with the whole image. Now we know for sure that an eye will always be in a face which is a skin region. Using this information we can narrow our search space in a great extent. We can only consider the skin regions as our search space and speed up the whole process.

To determine skin and non-skin region from an image we need to apply some skin-based segmentation process. Skin segmentation is the process of separating skin regions within an image from the background as well as from any other objects. The segmentation rules are the rules that will determine the formation of regions. These rules are mainly based on analyzing the pixel color properties of the region of interest which is human skin color in this case.

Skin-based segmentation is an open research problem and there has been much research in this sector [26-37]. Most of the skin segmentation techniques involve the classification of individual image pixels into skin and non-skin categories on the basis of pixel color which brings another challenge since pixel color differs under different lighting conditions. For this reason, we need to study different color models and find the reliable color model which is adaptable to people of different skin colors and in where we can differentiate between the luminance and chrominance components of a color pixel. We also need to know about pixel connectivity models for the skin segmentation procedure.

3.2.1 Pixel Neighborhood

An image in a digital image is spatially close to several other pixels. In a digital image represented on a square gird, a pixel has a common boundary with four pixels and shares a corner with four additional pixels. We say that two pixels are *4-neighbors* if they share a common boundary. Similarly, two pixels are *8-neighbors* if they share at least one corner.

For example, (in Figure 3.2) the pixel at location $[i, j]$ has four neighbors $[i+1, j]$, $[i-1, j]$, $[i, j+1]$ and $[i, j-1]$. The *8-neighbors* of the pixel include the *4-neighbors* plus $[i+1, j+1]$, $[i+1, j-1]$, $[i-1, j+1]$ and $[i-1, j-1]$. A pixel is said to be *4-connected* to its 4-neighbors and *8-connected* to its 8-neighbors.

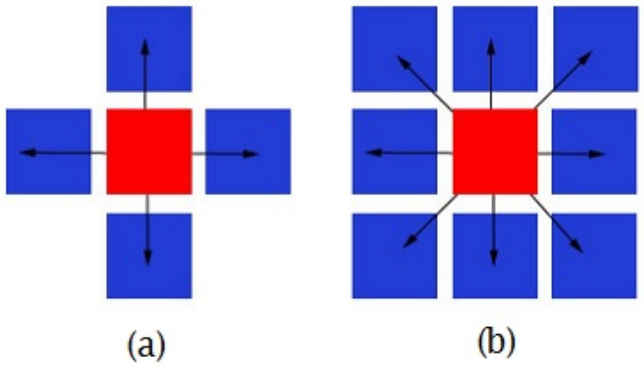


FIGURE 3.2: The 4 and 8-neighbors of a pixel

3.2.2 Color Models

A color space is a mathematical expression of the color collection. For the skin segmentation process we have studied three major color models: RGB, HSV and C_bC_r. Here is a brief discussion on these color models.

RGB Color Model

The RGB color model is the most widely recognized color model. It comprises of three components namely the Red, Green and Blue color channels. The RGB

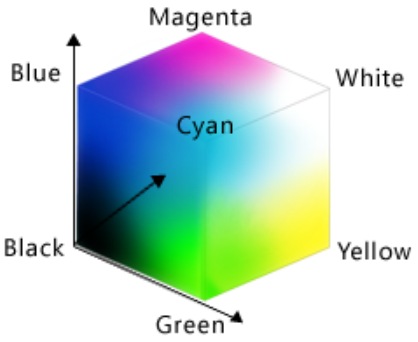


FIGURE 3.3: RGB Color Cube

model is used to a great extent in solving computer vision problems, but is better known for color representation in the displays of television sets and monitors. The value of a color by this model is best described as being a vector in three-space where red, green and blue represent the axis. Color is thus a result of the combination of the red, green and blue components.

HSV Color Model

The HSV model comprises of three components. These are the Hue, the Saturation, and the Value. This model is a more intuitive representation of color information. The model is best presented as a color cone or cylinder.

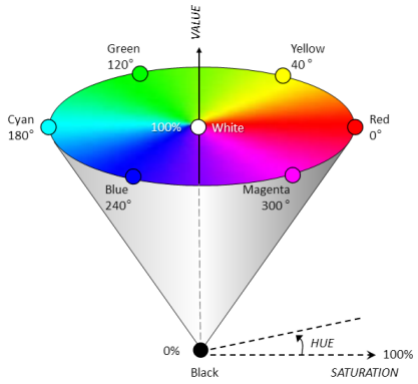


FIGURE 3.4: HSV Color Cone

The hue represents a person's perception of a color, for example green or orange. Hue changes as one move around the cone. Saturation is a measure of a color's

dilution by white light. Finally, the intensity is a measure of the brightness of the color. The great advantage of using this color model is the fact that it separates intensity from the color components unlike the RGB model that couples intensity with color information. The transformation between HSV and RGB color models is nonlinear.

YC_bC_r Color Model

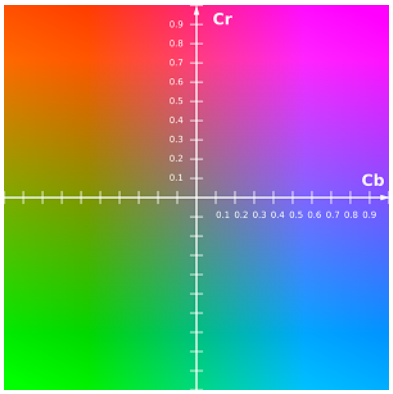


FIGURE 3.5: The C_bC_r Plane at a constant luma

In YC_bC_r space colors are specified in terms of luminance (the Y channel) and chrominance (C_b and C_r channels). Y is the luma component which specifies light intensity of a pixel and C_b and C_r are the blue-difference and red-difference chroma components of a pixel. Furthermore, YC_bC_r is not an absolute color space; rather, it is a way of encoding RGB information. The actual color displayed depends on the actual RGB primaries used to display the signal. Therefore a value expressed as YC_bC_r is predictable only if standard RGB primary chromaticities are used. The conversion between YC_bC_r and RGB color models is linear.

3.2.3 RGB to YC_bC_r Space

Most of the prominent skin segmentation methods [26], [29], [31] use HSV or YC_bC_r color space as both of them separate chrominance components from luminance

components. In our proposed system we decided to choose YC_bC_r color space because here we will have to convert every pixel of the image and we need to adopt a conversion system which requires less CPU computation power and since RGB to YC_bC_r conversion is linear, we decided to go with that. We can convert RGB to YC_bC_r color space using the following equation:

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0.2990 & 0.5870 & 0.1140 \\ -0.1687 & -0.3313 & 0.5000 \\ 0.5000 & -0.4187 & -0.0813 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{3.1}$$

3.2.4 Labeling Skin Regions

Mahmoud *et al.* [30] used the following C_bC_r space interval for skin segmentation:

$$Y \geq 80$$

$$85 \leq C_b \leq 135$$

$$135 \leq C_r \leq 180 \, where \, Y, C_b, C_r = [0, 255].$$

On another research, Chai and Phung *et al.* [29] developed an algorithm that exploits the spatial characteristics of human skin color. A skin color map is derived and used on the chrominance components of the input image to detect pixels that appear to be skin. The algorithm then employs a set of regularization processes to reinforce those regions of skin - color pixels that are more likely to belong to the facial regions. Working in the YC_bC_r space Chai and Phung [29] have found that the following range of C_b and C_r covers almost every tone in human skin cluster:

$$77 \leq C_b \leq 127 \, and \, 133 \leq C_r \leq 173$$

To be on the safeside we decided to take the upper and lower boundary of C_b and C_r components specified in the above approaches. The skin cluster we used:

$$Y \geq 80 \, and$$

$$77 \leq C_b \leq 135$$

$$133 \leq C_r \leq 180 \, where \, Y, C_b, C_r = [0, 255].$$

Our redesigned C_bC_r interval seems to be very robust and can detect skin regions from images with complex backgrounds. Figure 3.6 shows the output of our skin detection procedure:

FIGURE 3.6: Example of Skin Segmentation using a Test Image

Once we were able to identify whether a pixel is skin or non-skin we designed the following algorithm to get all the skin clusters from the input image:

1. Scan individual pixels and check if it is a skin pixel
2. If a skin pixel is found, run a flood-fill (DFS) using 8-neighbors model to get connected components of the skin pixels
3. When flood-fill finishes put the skin region in a queue
4. Do this for every pixel in the image

In this method we got all the skin regions visible in the image but since this method doesn't check whether the skin region is a face or not, many unnecessary regions are generated. However, we can discard many of these unwanted regions using some trivial checks which are discussed in the following section.

3.3 Discarding Possible Non-Face Regions

Like we said earlier, from our skin segmentation algorithm we may find lots of unnecessary skin regions (e.g. arm). Now in this step we need to perform a series of trivial tests to see if a region can qualify as a facial region:

- Checking height-width ratio. The height to width ratio of a human face should be close to 1. However, Chang and Robles *et al.* [50] proposed it should be greater than 0.8 and less than 1.6. We will check the height to width ratio of the detected skin regions and discard the regions that don't fall between these intervals.

- We should also have some checking about the area of detected region. According to Yang and Waibel [49], we can't retrieve facial features if the resolution is less than 16 x 16. We will discard such regions too.

- Lastly, after skin-segmentation, a face region should contain holes (non-skin pixel clusters inside a skin region) in it because the pixels on eye will be discarded during the face segmentation. For this reason, we should also discard solid regions without any holes as they cannot be a facial region.

3.4 Tracking Possible Eye Locations

After performing the above checks, we should now look for possible eye locations. Now from the last check of the above section we have come to know about holes. We should also keep track of pixels that are holes because after determining iris positions we will need to verify whether it's in a hole to be sure that it's an eye.

To keep track of the pixels that belong to a hole, we used a modified version of flood-fill (DFS) algorithm which searches for blank pixels inside a skin region and marks them as hole-pixels if found.

3.5 Iris Tracking

Many researchers have proposed and implemented various methods for segmentation and localizing iris from face images. Daugman *et al.* [54] has proposed one of the most practical and robust methodologies, constituting the basis of many functioning systems. Wildes *et al.* [18] proposed a gradient based binary edge map construction followed by circular Hough transform for iris segmentation [56].

In this chapter we describe our proposed iris tracking system which uses circular Hough transform algorithm. We also present noise reduction, effective edge detection methods and some other pre-processing techniques needed before applying Hough transform.

3.6 Circular Hough Transform

The Hough transform is a shape detection algorithm presented by Paul Hough in 1962 for the detection of features of a particular shape like lines in digitized images [57]. Later the fundamental algorithm was modified for detecting other regular curves like circle, ellipse and so on. It is a really robust algorithm and runs with less computational complexity. This algorithm is used to solve many problems in computer vision applications as most of the images contain feature boundaries which can be described as regular curves. The main advantage of the Hough transform technique is that it is tolerant to gaps in feature boundary descriptions and is relatively unaffected by image noise, unlike edge detectors [46].

The process of iris localization using Hough transform can be subdivided into the following procedures:

- Image smoothing for noise reduction.
- Applying and edge detection algorithm. We tried two different approaches in this step. Firstly, we used the naive Sobel filtering with 2^{nd} derivative edge improvement and later we tried a more robust and superior edge detector named Canny edge detector which gave better result than the first one.
- After getting edge intensity, an edge map is calculated.
- Then the edge map is accumulated into the (a, b)-space and from here we will get the coordinate of the center of iris circle.
- Finally we accumulate in r-space in order to find radius of the circle.

After performing all these steps we may find many unnecessary circles and for this reason the accumulator space is thresholded and only the highest peak values are taken into consideration.

3.6.1 Signal Convolution

Convolution is a mathematical operation where two signal functions f and g, produces a resultant function that is usually a modified version of one of the original functions. In our later steps we will be needing convolution frequently.

Though theoretically, signal convolution is performed on continuous signals we have used a discrete version as we are dealing with digitized images. Since we will be dealing with images with reasonable resolution, we decided to perform the convolution process in spatial coordinates rather than using Fourier transform (i.e. we directly computed $h = f * g$ instead of $h = \mathscr{F}^{-1}(\mathscr{F}f \cdot \mathscr{F}g)$. Although the multiplication of the Fourier images is much faster than the convolution in the

spatial coordinates, the Fourier transform, implemented as Fast Fourier Transform (FFT) with its complexity O(n log n), will be inefficient for convolution of the image and a small filter. The convolution method we implemented uses the following formula:

$$h[k,l] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i,j]g[k-i,l-j] \tag{3.2}$$

Actually in practice, i and j only run through the support of $f[i,j]g[k-i,l-j]$. We also should not forget that convolution distorts the image along the edges (since it assumes a periodic function, which is not the case here, and we padded the image with zeros in that case). Therefore, we should rather neglect the edges when working with the image after convolution.

3.6.2 Edge Detection

The purpose of edge detection in general is to significantly reduce the amount of data in an image, while preserving the fundamental structural properties to be used for further processing. Since we only need to detect circular shapes (i.e. iris) from the extracted face image, we removed all other unnecessary detail from our image using an edge detection method. However, before applying any edge detection algorithm we need to perform some pre-processing on the image. These pre-processing methods are described in the following sections:

3.6.3 Noise in Images

Real images are often degraded by random errors - this degradation is usually called noise. Noise can occur during image capture, transmission or any processing. Noise may be both dependent on, or independent of the image content.

Noise is usually described by its probabilistic characteristics. Idealilled noise, called white noise is often used. A special case of white noise is a Gaussian noise. A

random variable with a Gaussian (normal) distribution has its probability density function given by the Gaussian curve. In the 1D case the density function is:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{3.3}$$

Where μ is the mean and σ is the standard deviation of the random variable. Gaussian noise is a very good approximation to noise that occurs in many practical eases.

3.6.4 Gaussian Smoothing

Noises can be mistaken for edges in the image and may even lead to many unnecessary processing which may hamper the performance of the system seriously. To prevent such unexpected scenario, noise must be reduced. Therefore the image must be smoothed by applying a Gaussian filter. Here we use a 2D Gaussian filter. Since approximating the values of the distribution function of normal distribution is rather complicated, I use the probability density function instead. So, since the centre of the filter is shifted to (0, 0), we have

$$smoothfilter[i,j] = \frac{1}{(\sqrt{2\pi}\sigma)^2} \cdot e^{-\frac{i^2+j^2}{2\sigma^2}} \tag{3.4}$$

Instead of (mathematically correct)

$$smoothfilter[i,j] = \int_{i-0.5}^{i+0.5} \int_{j-0.5}^{j+0.5} \frac{1}{(\sqrt{2\pi}\sigma)^2} \cdot e^{-\frac{i^2+j^2}{2\sigma^2}} dy\, dx \tag{3.5}$$

However, we didn't use the above function to generate the Gaussian kernel because with a standard deviation of $\sigma = 1.4$ we will get a 5x5 Gaussian filter like following:

$$\frac{1}{159} \cdot \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \tag{3.6}$$

For this reason we used this pre-computed matrix instead of generating it manually. The effect of smoothing with this Gaussian filter is shown in Figure 3.7.

(a) Original Image (b) Smoothed Image

FIGURE 3.7: Noise Reduction using Gaussian Smoothing

3.6.5 Sobel Filter

Sobel filter is used for edge detection purposes. Technically, it is a discrete differentiation operator that computes an approximation of the gradient of the image intensity function. The operator uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives - one for horizontal changes, and one for vertical. The filter masks are shown below:

$$S_v = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad S_h = \begin{bmatrix} -1 & 2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \tag{3.7}$$

These filters are convolved with the image and then merged using Euclidean:

$$|\nabla f| = sobel_{merged}[i,j] = \sqrt{(sobel_{vert}[i,j])^2 + (sobel_{horiz}[i,j])^2} = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \tag{3.8}$$

sobelmerged then contains the edge map where each pixel contains the intensity of the edge.

In Figure 3.8 we show the edge images after applying vertical, horizontal sobel filters. A merged edge image of both of these is also shown.

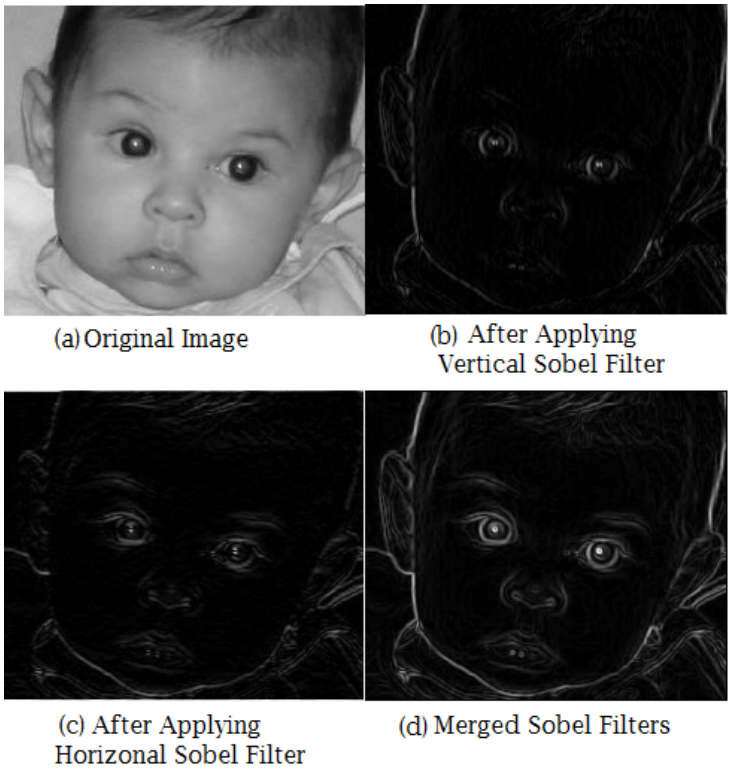


FIGURE 3.8: Example of Vertical, Horizontal and Merged Edge Images after Applying Sobel Filter

3.6.5.1 Edge Improvement Using 2^{nd}-Derivative

The Sobel filters themselves are quite often mislead by noise which they falsely recognize as edges. This can be to a large extent solved by smoothing the image but then the edges tend to be thicker. A good solution is to support the Sobel

filter by a 2nd-derivative filter. Whereas Sobel filters are based on looking-up the maxima of the first derivative (middle graph of Figure 3.9, the 2nd-derivative filter is looks-up zero crossings of the 2nd-derivative (right image of Figure 3.9).

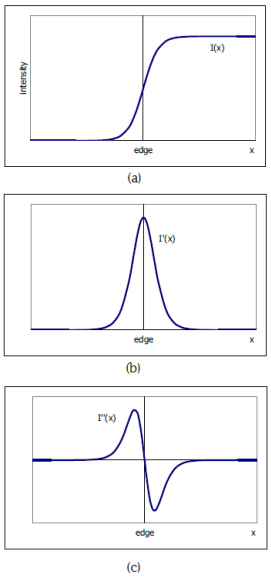


FIGURE 3.9: Behaviour of the intensity function (up), its derivative (middle), and second derivative (down) around an edge

We can seee that the first derivative reaches its maximum, the second derivative crosses zero at the edge. The 2^{nd}-derivative filter is implemented in the following way. To reduce the occurrence of high frequencies (noise), we smooth the image, and then apply the Laplace operator on the smooth image, i.e.

$$\nabla^2 (G * f) = \frac{\partial^2 (G * f)}{\partial x^2} + \frac{\partial^2 (G * f)}{\partial y^2} \tag{3.9}$$

where f is our image, and G is the Gaussian smoothing filter. From the linearity of convolution, it follows that

$$\nabla^2 (G * f) = (\nabla^2 G) * f. \tag{3.10}$$

$\nabla^2 G$ (Laplacian of Gaussian - LoG) can be pre-computed, and has the "Mexican hat" form (two "Mexican hat" filters of different sizes are used to calculate the derivative). After applying the "Mexican hat" filter, the zero-crossings have to be

found. This is done by running a 2x2 window over the image - if both positive and negative values appear in the window, a zero crossing occurs. A point is then considered as an edge point when the Sobel filter signalizes an edge and the "Mexican hat" filter signalizes a zero-crossing.

In Figure 3.10 we show an sobel edge image improved using 2^{nd} derivative.

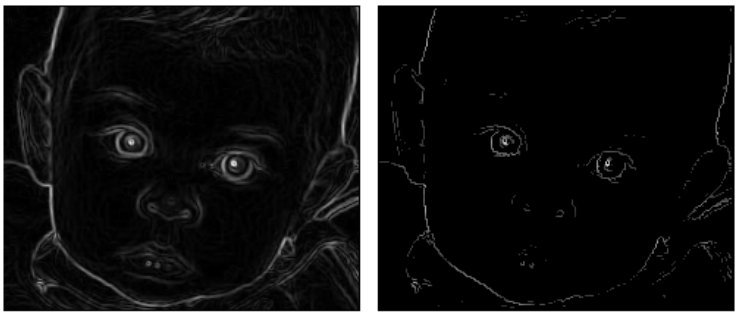


FIGURE 3.10: Improved Edge Image after Applying 2^nd Derivative

3.6.6 Canny Edge Detector

The Canny Edge Detector is one of the most commonly used edge detection method frequently used in image processing and computer vision applications. It is the best edge detection tool available so far and detects edge in a very robust manner. It was developed by John F. Canny [7] in 1986. The algorithm can be subdivided into the following five steps:

1. **Smoothing:** Gaussian smoothing to remove noise.

2. **Finding Gradients:** The edges are marked where the gradients of the image has large magnitudes. Gradients are computed using Sobel filter described earlier.

3. **Non-maximum Suppression:** Only local maxima should be marked as edges.

4. **Double Thresholding:** Potential edges are determined using a more robust thresholding mechanism.

5. **Edge Tracking by Hysteresis:** In this step, final edges are determined by suppressing all edges that are not connected to a very certain (strong) edge.

Since we have already discussed about the procedures of steps 1 and 2 we will start from step 3 in here.

3.6.6.1 Non-maximum Suppression

The edges we obtain after applying Sobel filter are either very thick or very narrow depending on the intensity across the edge and how much the image was blurred at first. Now our main target is to find edges that are only one pixel wide. In this step, we will keep only those pixels on an edge with the highest gradient magnitude by preserving all local maxima in the gradient image, and deleting everything else.

Now, for each edge pixel (x, y), three neighboring pixels are examined based on the following rules:

- If $\theta'(x, y) = 0°$, then the pixels $(x+1, y), (x, y)$, and $(x-1, y)$ are examined.
- If $\theta'(x, y) = 90°$, then the pixels $(x, y+1), (x, y)$, and $(x, y-1)$ are examined.
- If $\theta'(x, y) = 45°$, then the pixels $(x+1, y+1), (x, y)$, and $(x-1, y-1)$ are examined.
- If $\theta'(x, y) = 135°$, then the pixels $(x+1, y-1), (x, y)$, and $(x-1, y+1)$ are examined.

If pixel (x, y) has the highest gradient magnitude of the three pixels examined, it is kept as an edge. If one of the other two pixels has a higher gradient magnitude, then pixel (x, y) is not on the "center" of the edge and should not be classified as an edge pixel.

3.6.6.2 Double Thresholding

The edge-pixels remaining after the non-maximum suppression step are (still) marked with their strength pixel-by-pixel. Many of these will probably be true edges in the image, but some may be caused by noise or color variations for instance due to rough surfaces. The simplest way to discern between these would be to use a threshold, so that only edges stronger that a certain value would be preserved. The Canny edge detection algorithm uses double thresholding. Edge pixels stronger than the high threshold are marked as strong; edge pixels weaker than the low threshold are suppressed and edge pixels between the two thresholds are marked as weak.

3.6.6.3 Edge Tracking by Hysteresis

Strong edges are interpreted as "certain edges", and can immediately be included in the final edge image. Weak edges are included if and only if they are connected to strong edges. The logic is of course that noise and other small variations are unlikely to result in a strong edge (with proper adjustment of the threshold levels). Thus strong edges will (almost) only be due to true edges in the original image. The weak edges can either be due to true edges or noise/color variations. The latter type will probably be distributed independently of edges on the entire image, and thus only a small amount will be located adjacent to strong edges. Weak edges due to true edges are much more likely to be connected directly to strong edges. Edge tracking can be implemented by BLOB-analysis (**B**inary **L**arge **OB**ject). The edge pixels are divided into connected BLOB's using *8-connected* neighborhood. BLOB's containing at least one strong edge pixels are then preserved, while other BLOB's are suppressed.

The effects on test image after performing Canny edge detection is shown below:

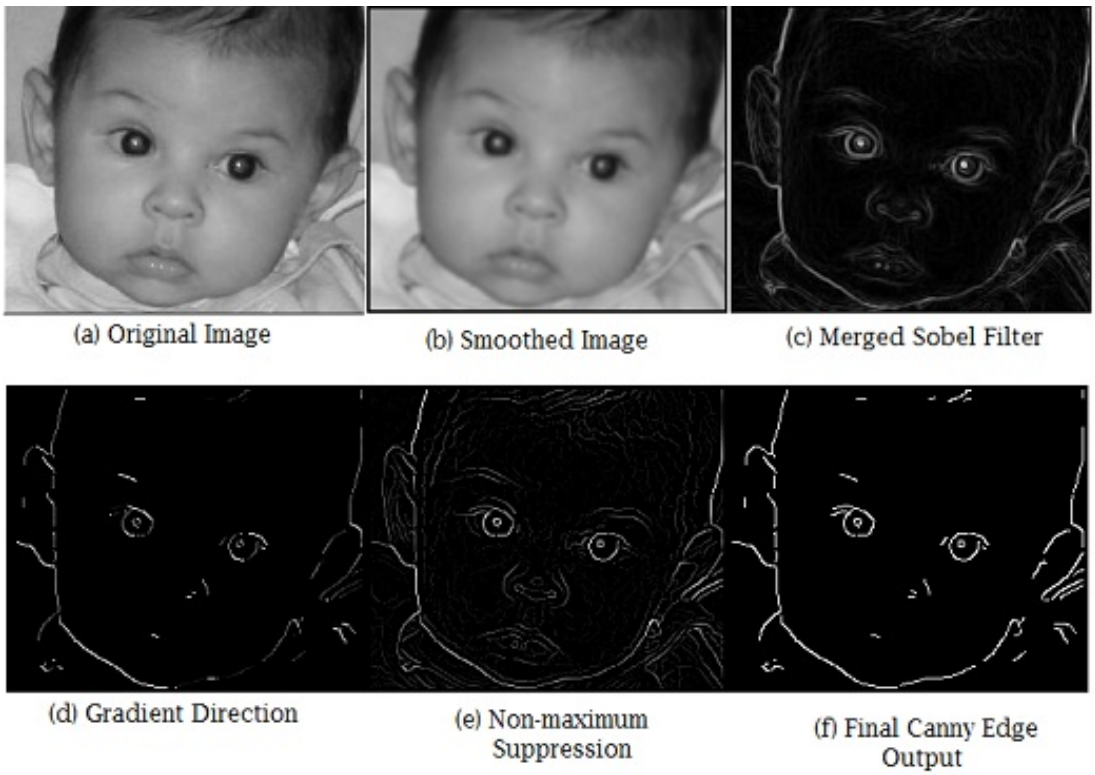


FIGURE 3.11: Steps in Canny Edge Detection

3.6.7 Performance Analysis of Sobel Vs Canny

We have tried edge detection using two different approaches: Sobel filter with 2nd derivative edge improvement and Canny edge detection. The final output images after applying both of these algorithms are given below:

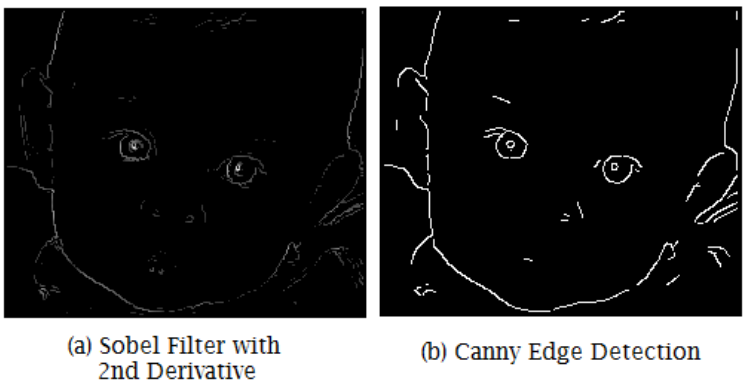


FIGURE 3.12: Edge Quality Analysis between Sobel with 2^{nd} Derivative and Canny Edge Detection Method

The output images show that Sobel operator with 2nd derivative edge improvement actually improves the edge condition but still it produces many unwanted edges (i.e. noise) and hence using this edge detection method in our proposed system can lead to some unwanted output. On the other hand, canny edge detection generates fine one pixel wide edge and also removes unnecessary edges or noises which seems better for our application. For this reason, we choose Canny edge detection for extracting edges from the face image.

3.6.8 Calculating Direction map

In order to calculate direction map, we make use of the already calculated convolutions of the image with the horizontal and vertical Sobel filter. For every detected edge point $[i, j]$ we calculate the angle θ as

$$\theta[i,j] \;=\; \tan^{-1}\left(\frac{\frac{\partial f}{\partial y}[i,j]}{\frac{\partial f}{\partial x}[i,j]}\right) \;=\; \tan^{-1}\left(\frac{sobel_{vert}[i,j]}{sobel_{horiz}[i,j]}\right) \tag{3.11}$$

The angle θ is normalized within the interval $(-\pi, \pi)$. However, since the edges with angles θ and $\theta + \pi$ have got the same direction, we can shift all $\theta[i, j]$'s so that they lie in $(-\frac{\pi}{2}, \frac{\pi}{2})$.

3.6.9 Accumulation into (a, b)-space

The key idea of the circular Hough transform is that we need to transform each of the points in (x, y)-plane to a parameter space. Now transforming a circle is actually simpler to represent in parameter space, compared to any other curve, since the parameters of the circle can be directly transfer to the parameter space. The equation of a circle is:

$$r^2 \;=\; (x \;-\; a)^2 \;+\; (y \;-\; b)^2 \tag{3.12}$$

As it can be seen that a circle has got three parameters: $a, bandr$. Where (a, b) is the center of the circle in the X and Y direction respectively and r is the radius. Such a circle can be represented using the following parametric equation:

$$\begin{aligned} x &= a + r\cos(\theta) \\ y &= b + r\sin(\theta) \\ where\, r &\,\epsilon\, (minr, maxr) \end{aligned} \tag{3.13}$$

Where (minr, maxr) is the range of circle radii that are taken into account. We defined minr to be 5 and maxr to 50 and this predefined range works for every case that we have encountered during simulation.

To obtain center of a circle, at each edge point we draw a circle with center in the point with the desired radius. This circle is drawn in the parameter space, such that our x axis is the a-value and the y axis is the b value. At the coordinates, which belong to the perimeter of the drawn circle, we increment the value in our accumulator matrix, which essentially has the same size as the parameter space. The following equation shows this process:

$$A(i \pm a,\ j \pm b) \leftarrow A(i \pm a,\ j \pm b) + E(i, j) \tag{3.14}$$

Where A is the (a, b)-space array or commonly known as accumulator array and $E(i, j)$ is the strength of the edge pixel, obtained by merging the two Sobel filter values of that corresponding point. Figure 3.13 shows the process clearly:
In this way, we sweep over every edge point in the input image drawing circles with the desired radii and incrementing the values in our accumulator. When every edge point and every desired radius is used, we can turn our attention to the accumulator. The accumulator will now contain numbers corresponding to the number of circles passing through the individual coordinates. However, these spots can be quite diffiused and dimmed, particularly if the circles are rather

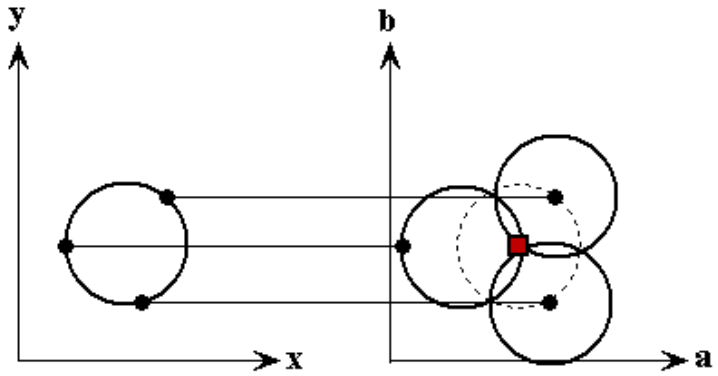


FIGURE 3.13: Accumulation into (a, b)-space

distorted into ellipses. Therefore it is necessary to concentrate these hot spots. I use convolution with a 17x17 "Mexican hat" filter which tends to concentrate the highest brightness in the centre of gravity of the hot spot. Then, the (a, b)-space is thresholded so that isolated spots are found. In these isolated spots, I find local maxima using a recursive algorithm which tends all local maxima with a minimum given distance. There can be more than one maximum in one spot if the spot is large enough. These maxima are then considered being centers of the sought-after circles.

Strictly speaking, the intensity of the line that is drawn into the (a, b)-space should decrease with factor $\frac{1}{r}$, i.e.

$$A(i \pm a,\ j \pm b) \leftarrow A(i \pm a,\ j \pm b) + \frac{E(i,j)}{r} \tag{3.15}$$

Because without this factor, large circles, consisting of 'more points', will accumulate higher value in the centre of the circle than small circles, where less points will contribute. On the other hand, due to inaccuracies in estimating the angle of the line, lines coming from greater distance will show larger dispersion in the area of the centre, which will balance the higher number of contributing points.

3.6.10 Accumulation into r-space

After the localization of the circles' centers, the circles' radii have to be found. This is done by accumulating in a one-dimensional space which coordinate is the radius of concentric circles with the given centre.

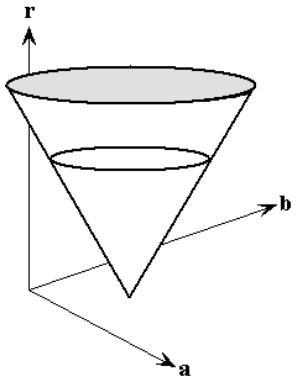


FIGURE 3.14: Accumulation into r-space

For every (discrete) radius in the desirable range (minr, maxr), the sum of edge strengths $E(i, j)$ for the points P along the circle with the given radius is calculated:

$$R(r) = \sum_{P \epsilon circle(r)} E(P) \tag{3.16}$$

In this way we can get radii of the circles that we have determined center coordinates in the previous section.

3.7 Thresholding Accumulator Space

We can get all the circles using the above method; however, this will generate a lot of circles (sometimes even more than hundred based on image quality and noise) and we only need the circles that represent iris. For this reason, we need to threshold this (a, b)-space and discard all those circles that cannot be iris at all.

To do this, we need to visualize the data in accumulator space and try to extract anything useful from it. For starters, we can easily assume that iris is the most

circular shape in a human face. Secondly, the accumulator space stores the number of circles passed through that point, therefore a higher accumulator value means a higher probability of that point being the center of a circle. Now merging these two facts, we can conclude that the highest values of accumulator space correspond to the iris positions.
For a frontal face image we should only consider the two highest peaks of accumulator space but to be on the safe side we take four peak values so that we do not miss any iris position by any chance. Later, we will discard unnecessary circles using some stronger features.

3.8 Eye Confirmation

In the last chapter we detected four possible iris as well as eye locations. Now we present a confirmation process by which we will be able to discard the circles that cannot be iris.

Eye confirmation is done using a very simple feature that we calculated earlier in section 3.2 (Skin-Based Segmentation). In section 3.2 we kept track of the non-skin pixel clusters (holes) inside the face region and we can easily conclude that if the detected iris location falls into a hole then it can be an eye and we will conduct another step of verification in the next section. For this step of verification we will check whether the detected iris which is a circle using basic coordinate geometry.

We know the coordinate of lower-left and upper-right corner of the bounding rectangle of a hole. Now if the center of an iris region is (x, y) with radius r then we can easily check whether it falls inside the hole region or not. We discard any circle that doesn't satisfy this condition.

3.9 Red-Eye Confirmation

To confirm whether an eye region contains red-eye or not, we simply scan every pixel of the bounding rectangle of iris circle. The Figure shows the iris circle with its bounding rectangle with coordinates:

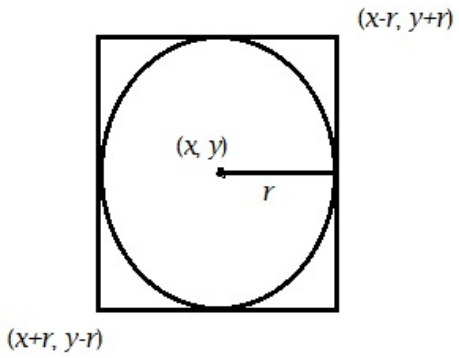


FIGURE 3.15: Iris Circle with Bounding Rectangle

3.9.1 Color-Based Segmentation in HSI Space

We used HSI color model to check if it's a red-eye pixel. We used Hue based red-pixel segmentation. RGB to HSI conversion is done using the following equations:

$$\begin{aligned} H &= \cos^{-1}\left\{\frac{\frac{1}{2}[(R-G)+(R-B)]}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right\} \\ S &= 1-\left\{\frac{min(R,G,B)}{\left(\frac{R+G+B}{3}\right)}\right\} \\ I &= \frac{1}{3}(R+G+B) \end{aligned} \tag{3.17}$$

HSI model simplifies red pixel selection process as we can directly access to the Hue component of the pixel. Here the transition from purple to red occupies a continuous arc around the hue component. Since red starts at 0° we decided to consider all the pixels that lies on the continuous arc with ±5° deviation. Figure 3.16 shows the hue range we used for red pixel segmentation:

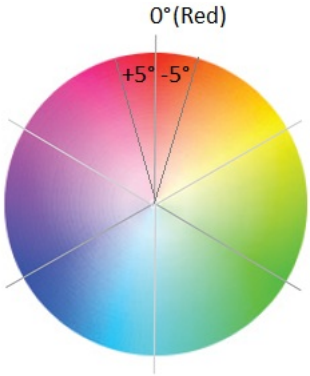


FIGURE 3.16: Hue Range Used for Red Pixel Segmentation

We used the following HSI range for red pixel selection:

$$H = (0 \pm 5)^\circ$$
$$S > 50$$
$$I > 25$$

The pixel selection algorithm also allows for pixels that are bright white. The reason for this is that the white pixels might otherwise divide potential red-eye regions into two regions, which is an unacceptable possibility.

Now if iris circle contains a cluster of red-eye pixels it will be considered as a red-eye otherwise we conclude there was no red-eye problem in the input image.

3.10 Restoration of Iris Color

We designed a simple yet realistic correction algorithm for restoring the natural color of the red-eye pixels. In the red-eye detection and correction method developed by Patti et al. [5] he claims, during red-eye effect, only the red component of the pixel gets damaged and the rest of the two components (i.e. green and blue) remain just fine. We used this information to design our correction algorithm.

Since green and blue are unchanged we keep them unchanged and assigned the average of these two as the red component. The following equations show the procedure:

$$\begin{aligned} R_{correct} &= \frac{G_{redeye} + B_{redeye}}{2} \\ G_{correct} &= G_{redeye} \\ B_{correct} &= B_{redeye} \end{aligned} \tag{3.18}$$

This procedure gives satisfactory output and preserves the glint inside the pupil.

3.11 Summary

In this chapter we have introduced our proposed solution for the red-eye problem in digital images. At first, we introduced the concept of digital images, its different parameters (e.g. resolution, connectivity), different color models and conversion between the color models. Later we presented our approach to reduce the search space using a skin-based segmentation along with some trivial checks to discard some false alarms. After that we discussed how iris is localized from the extracted face image in details. Finally, we developed logical some consequences that helped us to decide whether the detected iris circles correspond to actual eyes or not. We also presented the red pixel segmentation and red-eye method proposed by our red-eye detection and correction method. Our proposed method produced realistic outputs despite of being simple.

Chapter 4

Experimental Results and Performance Evaluation

4.1 Introduction

We have presented our automatic red-eye detection and correction system in the last chapter. In this chapter we demonstrate red-eye detection and correction results obtained using our system and measure performance of our proposed system. We have conducted extensive testing using a diverse red-eye image dataset consisting of 50 images with different resolution. However, here in this report we are presenting performance analysis of the major phases of our proposed system using only 10 selected images. Our experimental procedure is categorized into the following three categories:

1. Iris tracking performance analysis providing the results of iris localization process and corresponding performance evaluation.

2. Red-eye detection performance analysis presenting the results of red-eye detection procedure from the input images.

3. Red-eye correction performance analysis presenting the correctness and naturality of our proposed correction procedure.

4.2 Experimental Environment

We implemented and tested our proposed system on the following system environment:

- **Platform:** Mac OS X, Linux (Ubuntu)
- **Processor:** Intel Core i5 @ 2.5 GHz
- **Memory:** 4 GB RAM
- **Development Tools:**
    1. C++ in Eclipse IDE
    2. OpenCV

4.3 Test Image Preparation

We have collected test images from our personal photos and various other openly accessible image databases, some of them are mentioned below:

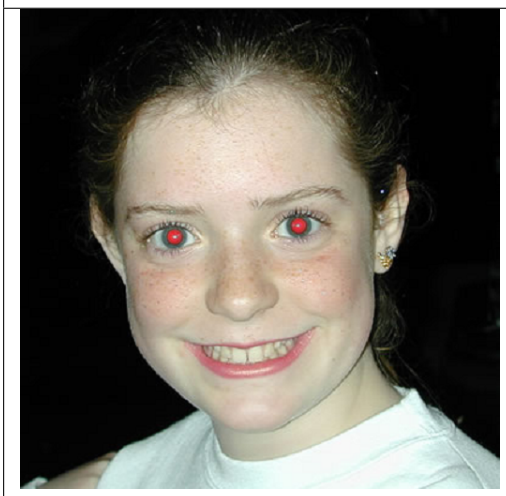
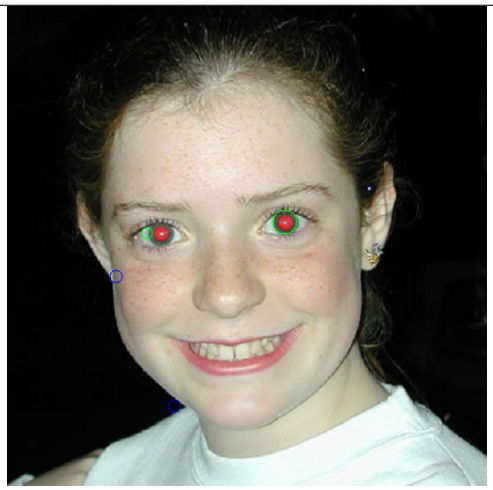
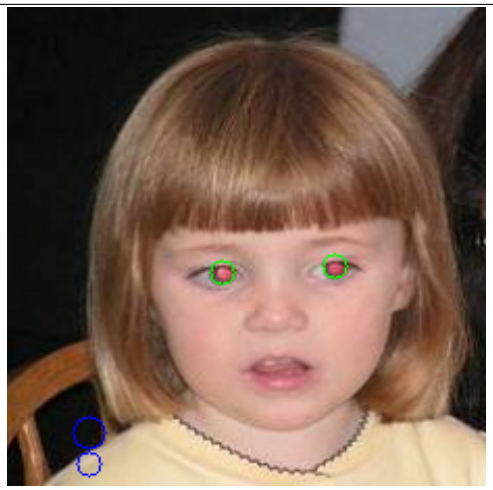
- The Berkeley Segmentation Dataset and Benchmark, UC Berkeley [58]
- Vision and Autonomous Systems Center's (VASC) Image Database, Carnegie Mellon University [59]
- Computer Vision Test Images, Carnegie Mellon University [60]

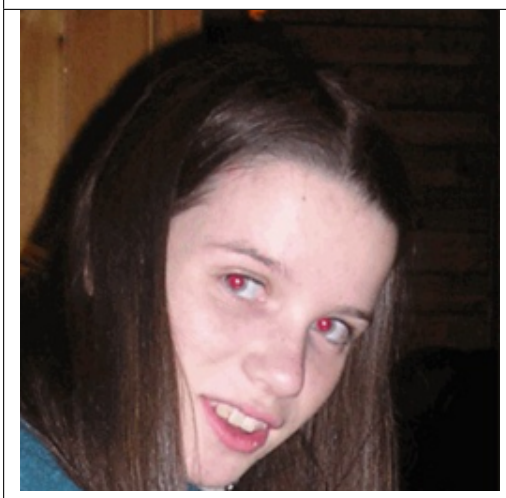
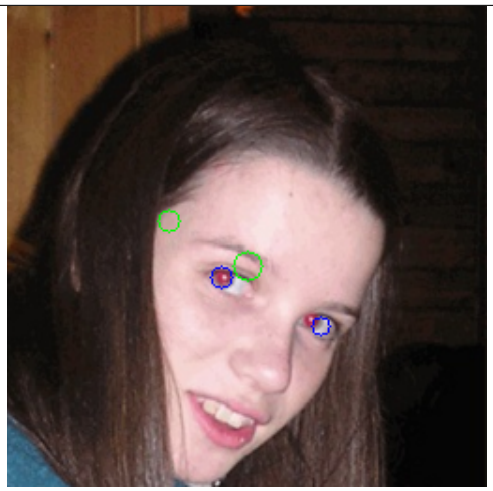
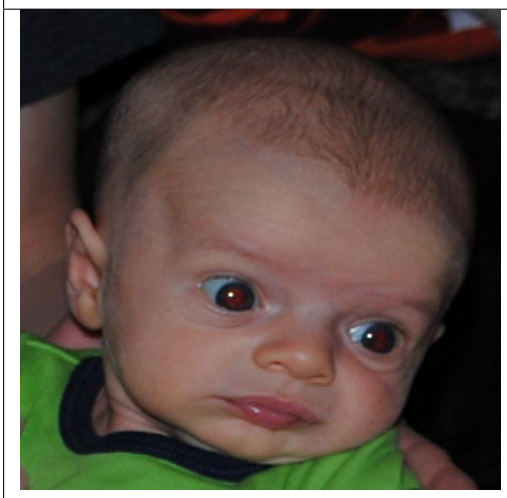
Our test images consisted of color *24-bit* bitmap images with red-eye artifact. Most of the images were of resolution 512x512 and 256x256 though our implementation is size invariant and can handle images of arbitrary resolution.

4.4 Performance in Iris Tracking

TABLE 4.1: Iris Tracking Performance Analysis

Input Image	Output Image	Comments
		Iris localization succesful. Successfully detected both irises.
		Iris localization successful from an image with complex background. Succcessfully detected both irises.
		Successfully detected both irises.

Input Image	Output Image	Comments
		One iris detected.
		Successfully detected both irises.
		Successfully detected both irises.

Input Image	Output Image	Comments
		One iris detected.
		Iris localization succesful. Successfully detected both irises.
		Iris localization succesful. Successfully detected both irises.

Input Image	Output Image	Comments
		Partial iris localization. One iris detected.

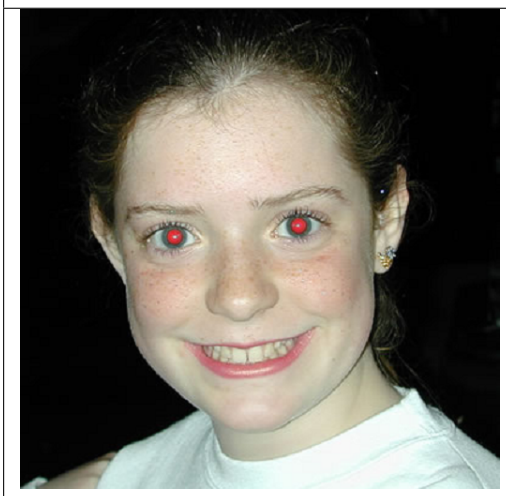
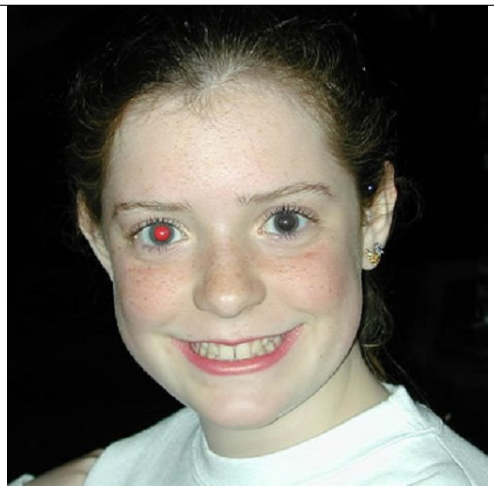
The average iris localization accuracy obtained from our experiment is 89%.

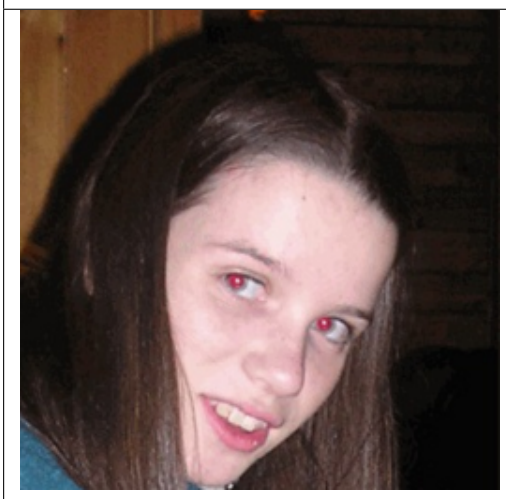
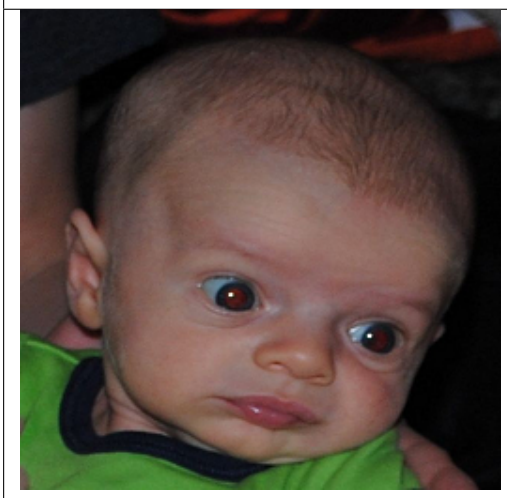
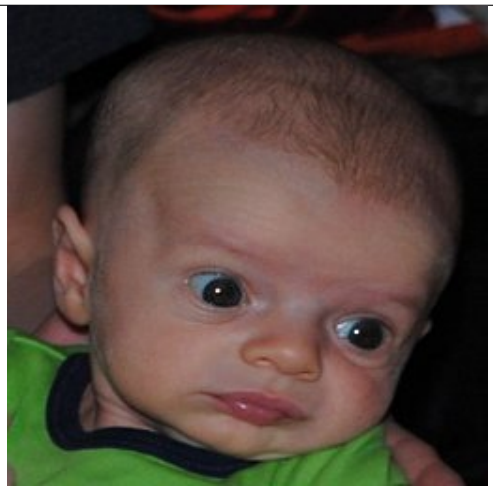
4.5 Performance in Red-Eye Correction

We tested the performance of our red-eye correction method using images with red-eye effect. Some of the selected results are presented here:

TABLE 4.2: Red-Eye Correction Performance Analysis

Input Image	Output Image	Comments
		Both red-eyes corrected successfully.
		Both red-eyes corrected successfully.
		Both red-eyes corrected successfully.

Input Image	Output Image	Comments
		One red-eye corrected.
		Both red-eyes corrected successfully.
		Both red-eyes corrected successfully.

Input Image	Output Image	Comments
		Both red-eyes corrected successfully.
		Both red-eyes corrected successfully.
		Both red-eyes corrected successfully.

Input Image	Output Image	Comments
		One red-eye corrected.

The average red-eye correction accuracy obtained from our experiment is 92%.

4.6 Summary

In this chapter we elaborated the experimental results of our proposed automatic red-eye detection and correction system. We presented outputs of the major phases of our proposed system along with accuracy ratio and performance analysis. Since our method uses only pattern matching and geometrical features, when run-time performance is considered our method clearly outperforms most of the existing solutions of red-eye problem. We also tested our implementation with some high resolution image (i.e. 4000x3000 Digital SLR camera images) and our system works without any problem and generates output within reasonable time span. Considering all the mentioned achievements, our proposed method clearly stands as an excellent performing red-eye detection and correction tool capable of being employed in any real-time or embedded system with low hardware configuration.

Chapter 5

Conclusion and Further Research

5.1 Introduction

This project aims at solving an existing problem: the red-eye effect, in the field of digital image processing and successfully implements an automatic red-eye detection and correction system. The proposed system can be used as a post-capturing operation in compact digital cameras or as a dedicated application for automatic red-eye detection and correction. In this project we explored the red-eye problem and current solutions thoroughly starting with comprehensive literature review. Being motivated by several pattern recognition and segmentation method in digital image processing, we eventually proposed a new approach and was able to validate its working procedure and analyze its performance by implementing the whole system from scratch.

5.2 Contributions

The major contribution of this project work is: it solves an open research problem with excellent performance both in terms of great red-eye detection and correction and minimal runtime complexity. We were not only able to perceive the red-eye problem from a relatively new perspective but were also able to establish it by

implementation. Here we list the unique features and achievements of our proposed method that represents our contribution:

- The system achieved 92% success rate in red-eye detection and correction.
- The system implements a robust iris localization method which can also be used as a pre-processing tool in iris recognition systems.
- Another important contribution of our proposed method is that the system doesn't use any machine learning features and for this reason it doesn't require any additional pre-processing hence it is lightweight in terms of runtime complexity and able to perform faster than any other current solution.
- The system performs several other image pre-processing techniques such as skin segmentation, shape detection and transformation between different color models which makes the system lighting condition invariant. Experimental results show that this helped achieving a better performance level and thus can be considered as a contribution.

5.3 Limitations and Future Improvements

One of the major limitations of the current system is that, its performance degrades when the skin segmentation cannot extract the full face image because of any facial conditions (e.g. spectacles, side pose). New research can be done so that the system becomes invariant of these conditions.

There are scopes for some other improvements such as the current system only works for humans but red-eye problem can also appear in the images of animals. Improving the eye detection phase of the current system may work in that case. The ultimate target of will be to design a system which will be able to solve red-eye problem for any photo in any constraints while preserving less runtime complexity that have already been achieved.

5.4 Concluding Remarks

Red-eye effect is a classical problem and one of the most extensively researched tasks in the field of digital image processing. Everyday billions of photographs with red-eye effect are being captured by millions of persons around the world so a fast solution of this problem is very much expected in this field. We feel that introducing a totally new concept in such a well studied problem is a notable achievement from our side. Being inspired by the past works on this problem, we logically dissected and analyzed the problem thoroughly and came up with a new solution and were successful in implementing and demonstrating the efficiency of our proposed system.

Bibliography

[1] J. Willamowski, G. Csurka, "Red Eye Detection and Correction", US7567707, July 2009.

[2] S. Ioffe, C. Troy, "Method and Apparatus for Red-Eye Detection", US7343028, March 2008.

[3] Nanu, F., Corcoran, P., Capata, A., Drimbarean, A., Steinberg, E., "Method of Detecting Redeye in a Digital Image", US0112599, (2008).

[4] S. Wu, "Method of Processing Red Eye in Digital Images", US7295686, November 2007.

[5] Patti, A., Kostantinides, K., Tretter, D., Lin, Q., "Apparatus and A Method for Reducing Red-Eye in A Digital Image", US6016354, (2000).

[6] J. Schildkraut, R. Gray, "Computer Program Product for Redeye Detection", US6252976, June 2001.

[7] John F. Canny, "A Computational Approach to Edge Detection", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 8, No. 6, pp. 679698, November 1986.

[8] L. Zhang, Y. Sun, M. Li, H. Zhang, "Automated Red-Eye Detection and Correction in Digital Photographs", Proc. IEEE Int. Conf. Image Processing (ICIP), pp. 2363-2366, (2004).

[9] S. Ioffe, "Red Eye Detection with Machine Learning", IEEE Int. Conf. Image Processing (ICIP), pp. 871-874, (2003).

[10] M. Gaubatz, R. Ulichney, "Automatic Red-Eye Detection and Correction", Proc. IEEE Int. Conf. Image Processing (ICIP), pp. 804-807, (2002).

[11] J. Wan, X. Ren, G. Hu, "Automatic Red-Eyes Detection Based on AAM", Proc. IEEE Int Conf Systems, Man Cybernetics, pp. 6337-6341, (2004).

[12] F. Volken, J. Terrier, P. Vandewalle, "Automatic Red-Eye Removal Based on Sclera and Skin Tone Detection", Proc. IS&T 3rd Eur. Conf. Color Graphics, Imaging Vision (CGIV), pp. 359-364, (2006).

[13] H. Luo, J. Yen, D. Tretter, "An Efficient Automatic Redeye Detection and Correction Algorithm", IEEE International Conference on Pattern Recognition (ICPR), Proceedings of the 17th International Conference, Vol. 4, pp. 883-886, August 2004.

[14] Matthew Gaubatz and Robert Ulichney, "Automatic Red-Eye Detection and Correction", Proc. International Conference on Image Processing (ICIP), (2002).

[15] B. Smolka, K. Czubin, J.Y. Hardeberg, K. N. Plataniotis, M. Szczepanski, K. Wojciechowski, "Towards Automatic Redeye Effect Removal", Pattern Recognition Letters, Vol. 24, No. 11, pp. 1767-1785, (2003).

[16] Ilia V. Safonov, "Automatic Red Eye Detection", Proc. of 17th International Conference on Computer Graphics and Vision (GRAPHICON), pp. 112-119, (2007).

[17] Yi Wang, Fuhuei Lin, "A Novel Automatic Red Eye Detection and Removal Method", Communication and Information Technologies (ISCIT07) Sydney, NSW, pp. 759-762, October 2007.

[18] R. Wildes, "Iris Recognition: An Emerging Biometric Technology", Proc. IEEE, vol. 85, pp. 1348-1363, 1997.

[55] S. Battiato, G. M. Farinella and M. Guarnera, G. Messina, D. Ravi, "Red-Eyes Removal through Cluster Based Linear Discriminant Analysis", Proceedings of the IEEE International Conference on Image Processing (ICIP), pp. 2185-2188, September 2010.

[20] D. Sidibe, P. Montesinos, S. Janaqi, Parc Scientifique, G. Besse, France, "A Simple and Efficient Eye Detection Method in Color Images", International Conference Image and Vision Computing, New Zealand, (2006).

[21] F. Gasparini, R. Schettini, "A Review of Redeye Detection and Removal in Digital Images Through Patents", Recent Patents on Electrical Engineering, Vol. 2, No. 1, pp. 45-53, (2009).

[22] M. Rizon and T. Kawaguchi, "Automatic Eye Detection using Intensity and Edge Information",Proceedings of TENCON 2000, pp. 24-27, September 2000.

[23] Raimondo Schettini, Francesca Gasparini and Fadi Chazli, "Modular Procedure for Automatic Red Eye Correction in Digital Photos", Proc. SPIE Color Imaging IX: Processing, Hardcopy, and Applications, pp. 139-147, vol. 5293, (2004).

[24] Jon Y. Hardeberg, "Red Eye Removal using Digital Color Image Processing", Proc. IS&T Image Processing, Image Quality, Image Capture Systems (PICS), pp. 283-287, (2001).

[25] Marchesotti L, Csurka G, Bresssan M., "Safe Red-eye Correction Plug-in Using Adaptive Methods", IEEE Proc Int Conf Image Analysis Processing Workshops (ICIAPW), pp. 192-195, (2007).

[26] D. Chai, and A., Bouzerdoum, "A Bayesian Approach to Skin Color Classification in YC_bC_r Color Space", In Proc. Of IEEE Region Ten Conference, vol. 2, pp. 421- 4124, (1999).

[27] F. Tomaz, T. Candeias, and H. Shahbazkia, "Fast and Accurate Skin Segmentation in Color images". CRV04, (2004).

[28] Neelamma K. Patil, Ravi M. Yadahalli, Jagadeesh Pujari, "Comparison between HSV and YC_bC_r Color Model Color-Texture based Classification of the Food Grains", International Journal of Computer Applications, Volume 34, No.4, November 2011.

[29] S. L. Phung, A. Bouzerdoum, and D. Chai, "A Novel Skin Color Model in YC_bC_r Color Space and Its Application to Human Face Detection". In IEEE International Conference on Image Processing (ICIP-2002), vol. 1, pp. 289292, (2002).

[30] Tarek M. Mahmoud,"A New Fast Skin Color Detection Technique", World Academy of Science, Engineering and Technology, 43, pp. 501-505, (2008).

[31] Ahmed Elgammal, Crystal Muang and Dunxu Hu, "Skin Detection - A Short Tutorial", Encyclopedia of Biometrics by Springer-Verlag Berlin Heidelberg, (2009).

[32] L. Sigal., S. Sclaroff and V. Athitsos, "Skin Color-Based Video Segmentation under Time-Varying Illumination", IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 26, pp. 862-877, July 2004.

[33] D. Chai and K. N. Ngan,"Face Segmentagion Using Skin-Color Map in Videophone Applications", IEEE Trans. on Circuits and System for video Technology, Vol.9, No.4, pp. 551-564, (1999).

[34] M.J. Jones and J.M. Rehg,"Statistical Color Models with Application to Skin Detection", Proc. IEEE Conf. on Computer Vision and Pattern Recognition, Vol. 1, pp. 274-280, (1999).

[35] Chiunhsiun Lin, "Face Detection in Complicated Backgrounds and Different Illumination Conditions by Using YC_bC_r Color Space and Neural Network", Pattern Recognition Letters, pp. 2190-2200, (2007).

[36] Baozhu Wang, Xiuying Chang, Cuixiang Liu, "A Robust Method for Skin Detection and Segmentation of Human Face", Second International Conference on Intelligent Networks and Intelligent Systems, (2009).

[37] Raja Y., McKenna SJ, Gong S., "Tracking and Segmenting People in Varying Lighting Conditions Using Colour", In Proceedings of Face and Gesture, Nara, Japan, pp. 228-233, (1998).

[38] Tang JS, Kawato S., Ohya J., "Face Detection from A Complex Background", Research Report, ATR Media Integration & Communications Research Laboratories, Kyoto, Japan, (2000).

[39] K. Sobottka and I. Pitas, "A Novel Method for Automatic Face Segmentation, Facial Feature Extraction and Tracking", Signal Processing: Image Comm., vol. 12, no. 3, pp. 263-281, (1998).

[40] B. Menser and M. Wien, "Segmentation and Tracking of Facial Regions in Color Image Sequences", SPIE Visual Comm. and Image Processing 2000, vol. 4067, pp. 731-740, June 2000.

[41] Kimme, C., D. Ballard and J. Sklansky, "Finding Circles by an Array of Accumulators", In the Communications of ACM, pp. 120-122, (1975).

[42] S. Pedersen, "Circular Hough Transform", Aalborg University, Vision, Graphics, and Interactive Systems, November 2007.

[43] D. H. Ballard, "Generalizing The Hough Transform to Detect Arbitrary Shapes", Pattern Recog. 13, pp. 111-122, (1981).

[44] H. Rhody, Chester F., "Hough Circle Transform", Carlson Center for Imaging Science Rochester Institute of Technology, October 2005.

[45] D. Ioannou, W. Huda, and A. F. Laine, "Circle Recognition Through a 2D Hough Transform and Radius Histogramming", Image Vision Comput. 17, pp. 15-26, (1999).

[46] H. K. Yuen, J. Princen, J. Illingworth, and J. Kittler, "Comparative Study of Hough Transform Methods for Circle Finding", Image Vision Comput. 8, pp. 71-77, (1990).

[47] Noureddine Cherabit, Fatma Zohra Chelali, Amar Djeradi, "Circular Hough Transform for Iris localization", Scientific & Academic Publishing, (2010).

[48] J. Cai & A. Goshtasby & C. Yu, "Detecting Human Faces in Color Images", Wright State University, U. of Illinois.

[49] Jie Yang and Alex Waibel, "A Real-Time Face Tracker", CMU CS Technical Report.

[50] Henry Chang and Ulises Robles, "Face Detection", Internet Publication. Available at: *http://www-cs-students.stanford.edu/ robles/ee368/main.html*

[51] J. D. Foley, A. V. Dam, S. K. Feiner, and J. F. Hughes, "Computer Graphics: Principles and Practice", New York, Addison Wesley, (1990).

[52] R. Jain, R. Kasturi, B. G. Schunck, "Machine Vision", Co-published by MIT Press and McGraw-Hill Inc., (1995).

[53] M. Sonka, V. Hlavac, R. Boyle, "Image Processing, Analysis and Machine Vision", 3rd Edition, (2008).

[54] J. Daugman, "How iris recognition works", IEEE Transactionson Circuits and Systems for Video Technology, vol. 14, pp. 21-30, January 2004.

[55] S. Battiato, G. M. Farinella, M. Guarnera, G. Messinaz, D. Ravi, "Boosting Gray Codes for Red Eyes Removal", Proceesing of International Conference on Pattern Recognition (ICPR), pp. 4214-4217, August 2010.

[56] Rajesh. Bodade and Dr. S. Talbar, "Dynamic Iris Localisation: A Novel Approach suitable for Fake Iris Detection", IEEE Conferences, 9781-4244-3941-6/09.

[57] Wojciech Wojcikiewicz, "Hough Transform, Line Detection in Robot Soccer", Coursework for Image Processing, 14th March 2008.

[58] The Berkeley Segmentation Dataset and Benchmark, University of California, Berkely. Available at: *http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/*

[59] Vision and Autonomous Systems Center's (VASC) Image Database, Carnegie Mellon University. Available at: *http://vasc.ri.cmu.edu/idb/*

[60] Computer Vision Test Images, Carnegie Mellon University. Available at: *http://www.cs.cmu.edu/ cil/v-images.html*