This is to certify that the
thesis entitled

DROWSY DRIVER DETECTION USING FACE TRACKING
ALGORITHMS

presented by

VENKATA RAVI KIRAN DAYANA

has been accepted towards fulfillment
of the requirements for the

Master of Science degree in Electrical and Computer Engineering

[illegible]
Major Professor's Signature

May 21, 2007
Date

DROWSY DRIVER DETECTION USING FACE TRACKING ALGORITHMS

By

Venkata Ravi Kiran Dayana

A THESIS

Submitted to
Michigan State University
In partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

2007

ABSTRACT

DROWSY DRIVER DETECTION USING FACE TRACKING ALGORITHMS

By

Venkata Ravi Kiran Dayana

Drowsy Driving is a major, though elusive, cause of vehicle crashes. The National Highway Traffic Safety Administration (NHTSA) conservatively estimates that 100,000 police-reported crashes are a direct result of driver fatigue each year in the United States. A detection system that can predict drowsiness and alert the driver by monitoring eye closure (PERCLOS) could reduce the number of fatigue-related crashes. Video sequence containing the driver's face can be analyzed to evaluate whether the eyes are open or closed. Ambient illumination may vary during the video sequence resulting in changes in perceived skin color. Therefore, a drowsy driver detection system demands robust, yet efficient algorithms to monitor a driver in real-time. In the present thesis, we evaluate different skin detection models to initialize the face of the driver followed by face tracking algorithms. Template-based tracking and Level-set based face tracking algorithms are implemented and optimized for real time operation. Results of the eye closure (PERCLOS) estimate are provided on a video database with multiple subjects.

Table of Contents

List of Figures

Chapter 1 Introduction

Drowsy driving is a class of problems related to operating an automotive vehicle while experiencing sleepiness or fatigue. Most drivers might have experienced drowsy driving at some time or other. According to the National Sleep Foundation's 2005 poll, 60% of adult drivers – about 168 million people – say they have driven a vehicle while feeling drowsy in the past year. Drowsiness slows reaction time, impairs judgment and increases the driver's risk of crashing. The National Highway Traffic Safety Administration (NHTSA) conservatively estimates that 100,000 police-reported crashes are a direct result of driver fatigue each year in the United States. This results in an estimated 1,550 deaths and 71,000 injuries. Often, the cause of a crash is reported as driver inattentiveness, which may be attributed to drowsiness or fatigue. Therefore, it is believed that drowsy driving is hugely underreported in the crash reports [4]. The above factors motivated vehicle manufacturers to focus on development of vehicle-based drowsy driver detection systems.

The basic idea behind a vehicle-based drowsy driver detection system is to monitor the driver unobtrusively for drowsiness. The detection system may sense both driver related features (physiological data) and vehicle related features (driving performance data), compute relevant measures to predict the onset of drowsiness. After the detection of drowsiness, the detection system can either alert the driver or take appropriate preventive action to avoid a crash. The system must have a very high detection rate and generate very few false alarms. The following sections would discuss in detail the need for on-board detection systems, comparison of relevant features and the practical issues in implementation.

1.1 Accidents related to Drowsiness

Driver fatigue or drowsiness is estimated to cause roughly 3.2 % of traffic accidents [20] by National Highway Traffic Safety Administration (NHTSA). Other researchers report that fatigue is responsible for 10 % of all the traffic accidents [1] and 25 % of all single-vehicle accidents [2]. A single-vehicle accident involves only one moving vehicle including collisions with animals, fixed objects and swerving off the road. In the United States, the cost of fatigue related crashes cost $12.5 billion per year in property loss and also claims 1,550 lives [20]. Drowsiness plays a much larger role in truck crashes. According to NHTSA 2003 report "Large Truck Crash Facts" [21], drowsy or fatigued drivers were responsible for 7.8 % of the single-vehicle accidents.

Statistics related to the overall drowsy driving, reveal a widespread problem. About 60 % of the adults in USA say that they have driven a vehicle while drowsy. Overall, 37 % of the driving population says they have nodded off or fallen asleep while driving at some time in their life [5]. Approximately, one third of the drivers' report that they last experienced this problem within past year alone. To summarize, drowsy driving is a common problem and is responsible for a significant number of vehicle crashes each year.

1.2 Drowsiness and Driver Vigilance

Studies in the previous section show anecdotal evidence for drowsiness causing a significant number of vehicle crashes. A thorough understanding of drowsiness can help us evaluate its effects on a driver's performance. Drowsiness, also referred to as sleepiness, is defined as the need to fall asleep. This process is a result of both circadian rhythm (24-hour cycle in the physiological processes of animals) and the need to sleep. Sleep can be irresistible and neurologically based sleepiness contributes to human error in a variety of settings and driving is no exception.

Drowsiness leads to crashes because it impairs elements of human performance that are critical to safe driving. Relevant impairments identified in the laboratory and in-vehicle studies include:

- ***Slower reaction time:*** Sleepiness reduces optimum reaction times as seen in Figure 1.1. Moderately sleepy people can have an increase in their reaction time that will hinder stopping in time to avoid a collision. Even small decrements in reaction time can have a profound effect on crash risk, particularly at high speeds.
- ***Reduced vigilance:*** Performance on attention-based tasks declines with sleepiness, including increased periods of non-responsiveness or delayed responding.
- ***Increased time for information processing:*** Processing and integrating information takes longer, the accuracy of short-term memory decreases, and performance declines.

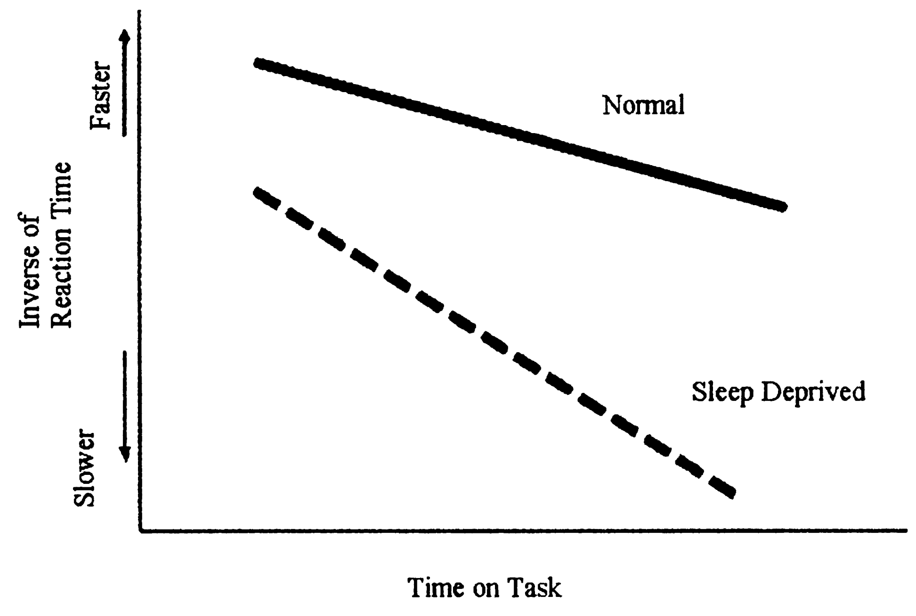


Figure 1.1 Effects of drowsiness on reaction time [6]

1.3 Drowsiness Features

Psycho-physiological and performance changes precede the onset of drowsiness. These changes form the basis for drowsiness features. Real-time monitoring of these features enable us to detect or predict the onset of drowsiness associated with the loss of driver alertness. These features can be broadly categorized as –

- Physical changes of drivers, eg., yawning, closed eyes.
- Physiological changes of drivers as measured by Electroencephalogram (EEG), Electrooculographic (EOG), Heart Rate Variability (HRV)

- Driving performance changes. For example lateral acceleration of a vehicle, steering variability.
- Subsidiary information, eg., time of the day, as drivers are more prone to drowsiness related crashes during the night [7].

All the above features contain discriminatory information to detect the drowsiness of a driver. Physiological changes have been frequently used to detect drowsiness. Brain waves measured with electroencephalograms (EEG) are especially good indicators of sleepiness and thus potential lapses in attention [8]. However, monitoring physiological changes requires the drivers to wear special sensors every time they set out to drive. Physical changes of drivers such as head orientation can be closely monitored using an on-board camera. Hence, most detection systems use physical changes as the primary feature along with other subsidiary features.

1.3.1 Physical Features

Physical changes are the observable features a driver exhibits before or during the onset of drowsiness. These features capture changes in the eyes and face of a person due to drowsiness. Multiple eye-based features [9] like blink frequency, eye gaze direction and PERCLOS (PERcentage eyelid CLOSure) have discriminatory information in detecting drowsiness. Overall, PERCLOS feature was identified as the most reliable feature in measuring the driver alertness by the U.S. Federal Highway Administration [10]. PERCLOS feature measures the percentage of time in a minute when the eyelids are at least 80 % closed. Various other authors also refer PERCLOS as a standard measure for drowsiness detection [22].

PERCLOS features can be computed in real-time using image-processing techniques on the video data acquired from an on-board camera. The most common technique to compute PERCLOS feature is developed by researchers using an infrared camera [36, 37, 38]. Human eyes reflect infrared radiation centered on a wavelength of 850 nm [36, 38]. Two consecutive images are acquired simultaneously at 850 nm and 950 nm wavelengths and their difference highlights the bright pupil region. Figure 1.2 shows the acquired infrared images and their image difference [38]. A thresholding operation on the difference image is sufficient to decide whether the eyes are open or closed. The main drawback of this method is the need for two infrared illumination sources at 850 nm and 950 nm wavelengths. During the day, the light from sun contains infrared radiation at various wavelengths. This radiation prevents the acquisition of infrared images at precise wavelengths of 850nm and 950nm. Overall, infrared camera based PERCLOS feature performs well only during the night.

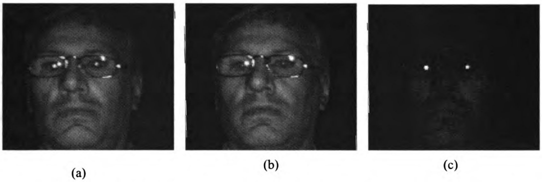


Figure 1.2 PERCLOS feature using infrared camera [38] (a) Infrared image captured using a 850 nm illumination source (b) Infrared image captured using a 950 nm illumination source. (c) Difference image with bright eye pupils.

An ordinary digital camera can capture color images during the day. This camera can be combined with the infrared camera to compute PERCLOS feature during both day and night. But, the image processing techniques to compute the PERCLOS feature on normal color images are not straightforward. Video data must be processed to detect the face and eyes of the driver in various conditions. The image processing algorithms must be efficient enough to track the eyes in real-time. Researchers have attempted to compute the PERCLOS feature using ordinary camera [40], without addressing the variations in illumination and skin color. The following sections describe an overall PERCLOS detection system using the color images and the problem of continuous tracking of face and eyes through multiple variations.

1.4 PERCLOS based detection

The schematic diagram of a drowsiness detection system is shown in Figure 1.3. An on-board camera captures the driver's face at a high resolution to identify whether his / her eyes are closed. The images are processed to compensate for different variables such as illumination conditions, skin-color and pixel noise. The face and eyes of the driver are continuously tracked. The tracked eye regions are classified at each frame into "open" and "closed" classes. Using training data generated using known controlled conditions, reliable PERCLOS features can be computed. PERCLOS features can be coupled with other subsidiary features to detect drowsiness in the driver. The system may then take remedial action or alert the driver of impending danger.

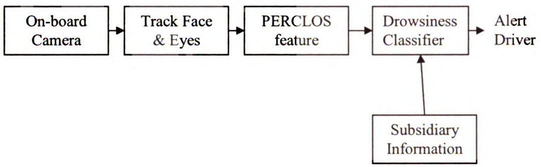


Figure 1.3 Schematic diagram of PERCLOS based detection module.

1.5 Tracking Face and Eyes

Tracking an object in a dynamic environment is a challenging problem [11]. In the PERCLOS based drowsy driver detection system, numerous variables can pose difficulties to the tracking algorithms. The driver's face can assume multiple positions as shown in Figure 1.5 (Images are presented in color). The color of the ambient illumination can cause color variations in the captured images as shown in Figure 1.4 (Images are presented in color). The above variations must be addressed by the face and eye tracking algorithms. The objective of this thesis can be described as –

"Track the face and eyes of a person using efficient image-processing algorithms and compute reliable PERCLOS measure"

Though the current objective of face tracking is to compute PERCLOS features, there are other applications for tracking such as face recognition and gesture recognition. [12].

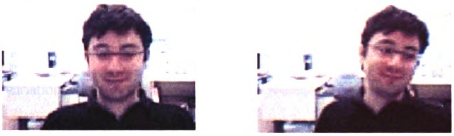


Figure 1.4 Images with two different positions of the face

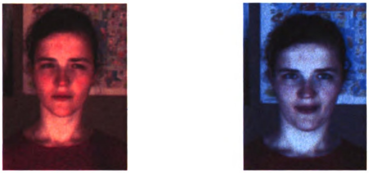


Figure 1.5 Images captured under different ambient illumination [9].

1.6 Outline of Thesis

The remainder of the thesis discusses various tracking algorithms and the features used in detecting the face regions. Chapter 2 discusses skin color detection methods for segmenting the face region. Chapter 3 presents a literature review of a variety of face tracking algorithms that were implemented and evaluated to monitor face and eyes of driver. Chapter 4 describes a pattern recognition classifier that differentiates between an open and closed eye. Results of combining all the above modules are documented in the Chapter 5. Conclusions and Futures work are presented in the Chapter 6.

Chapter 2 Face and Eye Detection

Skin color is an important feature for face detection and tracking. Color is robust to geometric variations and allows fast pixel-based processing. Also, human skin has a characteristic color and can be easily recognized by humans. Skin color based methods can be broadly classified as pixel-based and region-based methods. Pixel-based methods categorize each pixel as skin or non-skin individually. In contrast, region-based methods analyze a group of pixels and find the spatial relation between them. Both the methods require computation of skin color features based on a color space (RGB, HSI etc.). There is no clear consensus among researchers [13] about the superiority of one color space over another. The subsequent face detection can be accomplished using parametric techniques such as modeling skin color with a Gaussian distribution or non- parametric techniques such as color histogram or Bayes classifier.

2.1 Color Spaces

Color spaces are mathematical representation of colors by 3-tuples of numbers. The most common color space is tri-chromatic RGB (Red-Green-Blue). The other frequently used color spaces are the subtractive CMY (Cyan-Magenta-Yellow) and the HSI (Hue-Saturation-Intensity) color spaces.

2.1.1 RGB color space

RGB color space representation expresses any color as a combination of the primary color components (Red, Blue, and Green).

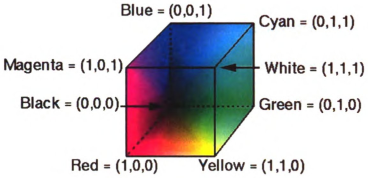


Figure 2.1 RGB color space representation [42]

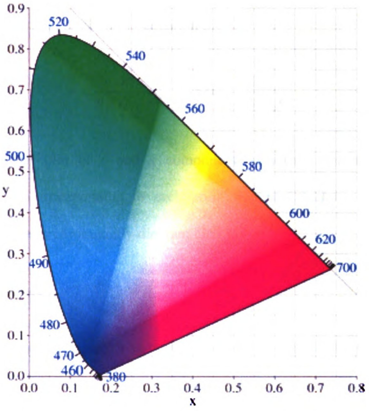


Figure 2.2 Chromaticity diagram with normalized *r* on the x-axis and normalized *g* on the y-axis. The outer curved boundary is the spectral (or monochromatic) locus with wavelengths shown in nanometers [35]

It is one of the most widely used color spaces. for storing and processing digital image data. Cathode Ray Tubes (CRT) and LCD (Liquid Crystal Display) screens utilize this representation for displaying images. The advantages of this representation are its additive properties and simplicity. However, luminance (grey-intensity) and chrominance information are not independent and there is high correlation between the three channels. Figure 2.1 shows the geometry of the RGB color model for specifying colors using Cartesian coordinate system

Face is a highly curved surface. Therefore, the observed intensity values (R, G, and B) exhibit strong variations. These variations may be minimized by normalizing each component with the overall intensity. This gives an intensity-normalized color vector with two components (*r*,*g*) as shown in equation 2.1

$$r = \frac{R}{R+G+B}; \qquad g = \frac{G}{R+G+B} \tag{2.1}$$

The two dimensional plot of *r* and *g* components creates a chromaticity diagram depicting the colors corresponding to each point. Figure 2.2 shows the location of different colors on the *rg* chromaticity diagram [35]

2.1.2 HSI color space

HSI color space expresses any color in terms of easily understandable properties – Hue, Saturation and Intensity. Hue defines the dominant color (such as Red, Yellow) of an area. Saturation is the amount colorfulness of an area in proportion to its brightness. The intensity value is related to the overall luminance of an area. Due to the separate encoding of chromaticity components -- Hue and Saturation from the intensity I, the HSI color space is intuitive to the user. The explicit discrimination between luminance and chrominance components makes HSI color space popular for color analysis.

The HSI model in Figure 2.3 shows the HSI solid on the left. The HSI triangle representing a constant intensity with different colors is formed by taking a horizontal slice through the HSI solid. Hue is measured as an angle starting from the red corner. Saturation is given by the distance from the central axis.

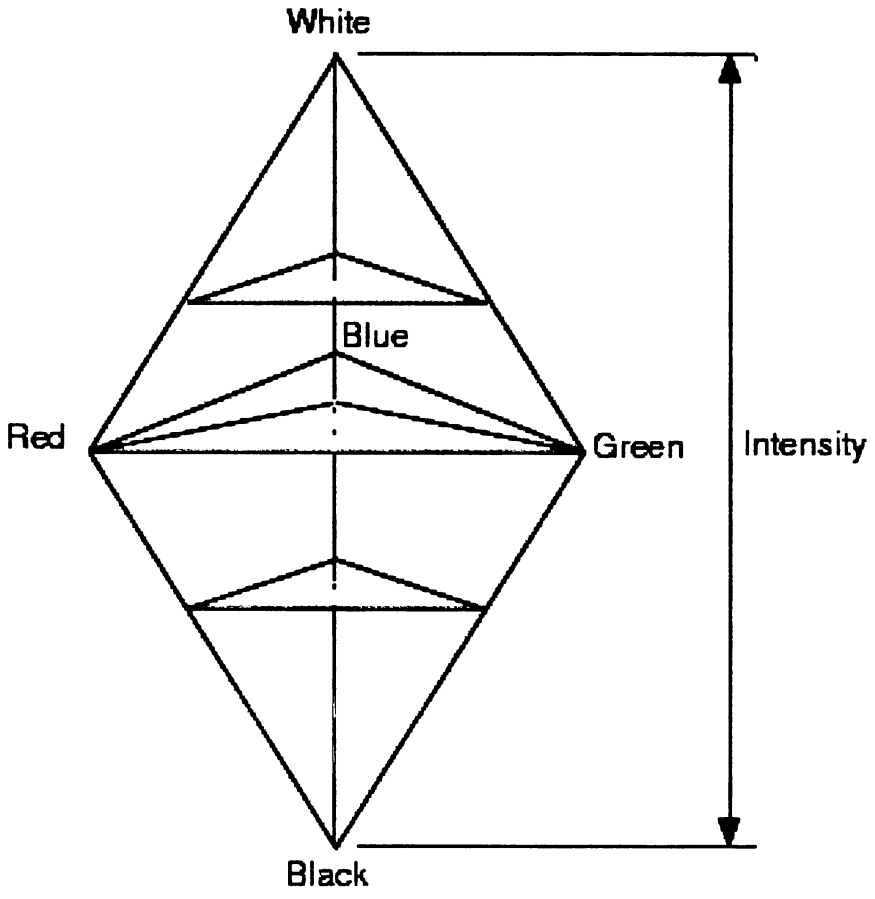


Figure 2.3 HSI color space representation. Colors on the surface of the solid are fully saturated, i.e. pure colors, and the grey scale spectrum is on the axis of the solid. [42]

Conversion between HSI and RGB can be performed using the below equations 2.2- 2.4

$$H = \arccos\frac{\frac{1}{2}((R-G)+(R-B))}{\sqrt{((R-G)^2+(R-B)(G-B))}} \tag{2.2}$$

$$S = 1-3\frac{\min(R,G,B)}{R+G+B} \tag{2.3}$$

$$I = \frac{1}{3}(R+G+B) \tag{2.4}$$

2.1.3 YCrCb color space

YCrCb is an encoded RGB representation for color television. Unlike RGB color model, there is clear distinction between luminance and chrominance components. The luminance (Y) component contains all the information required for black and white television, and captures our perception of the relative brightness of colors. Humans perceive green as much lighter than red, and red, lighter than blue. These relative intensities are encoded by their respective weights of 0.587, 0.299 and 0.114 in the RGB conversion equation 2.5. The two chromaticity components indicate the red (Cr) and blue (Cb) components.

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.144B \\ Cr &= R - Y \\ Cb &= B - Y \end{aligned} \tag{2.5}$$

2.2 Skin Modeling

Skin modeling is the development of a mathematical representation for the quantitative description and detection of skin regions in an image. The goal of skin detection is to build a decision rule that will discriminate between skin and non-skin pixels. This goal is usually accomplished by introducing a distance metric, which measures the distance between the pixel color and skin tone. Distance metric and the choice of color space are defined by the skin color model.

A pixel-based skin classifier can be built in one of the three ways –

- Explicitly defined skin model
- Non-parametric skin distribution model
- Parametric skin distribution model

2.2.1 Explicit skin model

This model segments a color space in to skin and non-skin clusters. The segmented skin cluster is defined explicitly using a number of straightforward rules. Essentially, an explicit skin model is a rule-base classifier with fixed thresholds. For example an explicit RGB skin model [23] can be characterized by the below rules.

A pixel (R,G,B) is classified as skin if all rules are satisfied:

- $R > 95$ and $G > 40$ and $B > 20$

- $max\{R,G,B\} - min\{R,G,B\} > 15$ and
- $|R - G| > 15$ AND $R > G$ AND $R > B$

The simplicity of the detection rules leads to the construction of a fast classifier. In order to evaluate the explicit model, we need a large dataset containing faces.

VidTIMIT dataset [24] comprises videos of 43 volunteers (19 female and 24 male), reciting small sentences. This dataset was recorded in 3 sessions, with an average delay of 7 days between Session 1 and Session 2, and 6 days between Session 2 and Session 3. The delay between sessions allows for changes in hair style and clothing. All the volunteers recited ten sentences with an average duration of 4.25 seconds per sentence. In addition, each person performed a *head rotation* sequence. The sequence consists of a person moving his / her head to the left, right, up and down. All the videos are recorded at a resolution of 384 x 512 pixels (rows x columns) and 25 frames per second.

For evaluating the explicit RGB skin model, images are extracted from the videos in VidTIMIT dataset. The performance of explicit RGB skin model can be seen in the results shown in Figure 2.4 and Figure 2.5. The skin regions from most of the images are properly segmented as shown in Figure 2.4. But, in a few images where the hair or clothing is similar to skin colors (eg. brown, yellow and orange) the segmentation results was erroneous as seen in Figure 2.5.

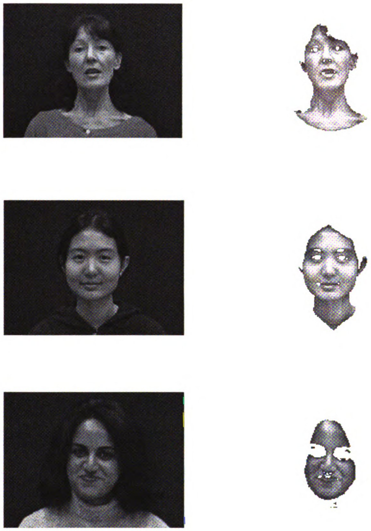
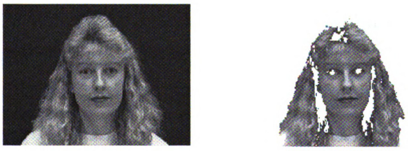


Figure 2.4 Three images from the VidTIMIT dataset on the left and their corresponding skin segmentation results on the right.



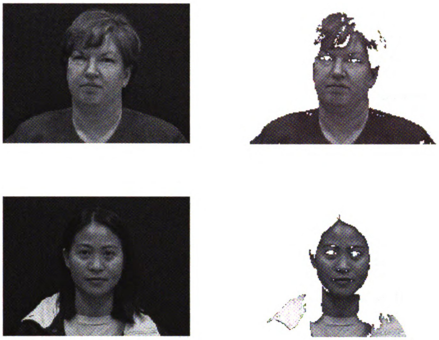


Figure 2.5 Improper skin segmentation results on three images from VidTIMIT dataset.

The quality of skin segmentation using the explicit skin model is evaluated manually. A decision is made as to whether the segmented skin regions contain more than 80% of the skin pixels and less than 20 % of the non-skin pixels. Using this decision metric, the explicit RGB skin model segmented skin from 63 % of the faces in VidTIMIT dataset. Unlike other models, there is no training involved and the decision rules are created empirically. The explicit skin model captures a rough estimate of the skin cluster in a particular color space. It is a useful model when skin segmentation has to be done without any training examples or prior information.

2.2.2 Non-Parametric models

The main idea underlying non-parametric skin modeling is to estimate the skin color distribution from the training data. The color space is partitioned into discrete regions and assigned a skin probability value for each region. The most common way to estimate the discrete probability distribution is by generating the histograms of skin and non-skin regions [25, 26].

Color space is quantized into a number of bins, each corresponding to a particular range of color components. For example, if the choice of color space is HSI, then a two dimensional histogram of Hue and Saturation is generated. Each bin of the 2D histogram stores the number of times that particular color component occurred as skin in the training images. After accumulating the histogram bins from all the training images, the histogram counts are normalized converting histogram values in to discrete probability distribution. Therefore, the probability estimate of a color belonging to skin can be generated using equation 2.6

$$P(c \mid skin) = \frac{skin[c]}{Norm} \tag{2.6}$$

where *skin*[c] gives the value of the histogram bin, corresponding to the color vector *c* and *Norm* is the normalization coefficient. These normalized bins represent the likelihood of a particular color *c* corresponding to skin. The estimate of probability density is continuous and has more information than the discrete results of explicit model. Figure 2.6 shows a face image and its corresponding skin probability estimate.

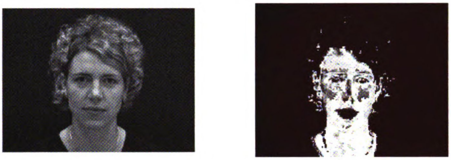


Figure 2.6 Skin probability estimation using 2D histogram. Right image shows the probability of skin at each pixel. Note the darker regions of eyes and face with less probability.

The two dimensional histogram of skin and non-skin pixel colors, makes it possible to apply the Bayes rule. Bayes rule to compute the probability of observing skin color pixel *c* is –

$$P(skin \mid c) = \frac{P(c \mid skin)P(skin)}{P(c \mid skin)P(skin) + P(c \mid non-skin)P(non-skin)} \tag{2.7}$$

Both the conditional probabilities *P (c| skin)* and *P (c| non-skin)* can be estimated directly using the skin and non-skin histograms. The prior probabilities can be estimated from the overall skin and non-skin pixels expected in the image. Bayes rule is then applied to the test data provided the scene illumination remains the same as in the training images.

After the estimation of probability distribution, the skin segmentation is straightforward using the maximum aposteriori probability (MAP) rule.

$$pixelclass = \underset{\lambda \in \{skin, non-skin\}}{\arg\max} \; (P(\lambda \mid c)) \tag{2.8}$$

A pixel in a test image is classified as skin, if the probability *P (skin | c)* crosses a threshold θ (generally 0.5). The advantages of this model over an explicit model come from the quantization of the color space with histogram bins. Quantization increases the generalization ability of the model. In addition, the absence of luminance information in the model makes it robust to illumination changes. The important parameters in this model are the histogram bin size and the threshold θ. The value of threshold θ is set at 0.5 using the MAP rule, and the histogram bin size is determined using a 10-fold cross validation method. [44]

Color component *c* can be represented in any of the chrominance pairs such as normalized *r* and *g*, Hue and Saturation, red and blue chromaticity. The effectiveness of a skin detection algorithm depends on the chrominance space in which the skin color is modeled. Recent studies have compared the performance of different color spaces for human skin detection [13]. These studies support the Hue-Saturation chrominance space as exhibiting the smallest overlap between and skin and non-skin distributions. The skin probability distribution generated using 12 images from VidTIMIT data set is shown in Figure 2.7 . The skin regions are manually selected from these training images.

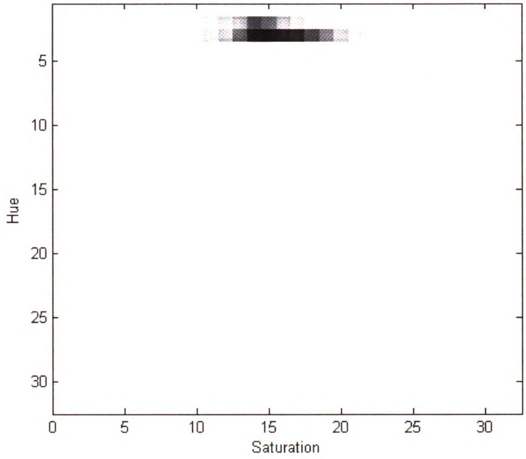
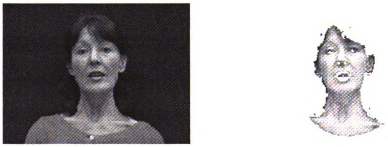


Figure 2.7 2D histogram of skin with 32 x 32 bins on Hue-Saturation chrominance space. Darker regions indicate higher frequency counts.

Using the above histogram, the test images from VidTIMIT data are segmented. The performance of the non-parametric probability distribution method is shown in Figure 2.8 and Figure 2.9 . Overall, 81 % of the images in VidTIMIT are correctly segmented.



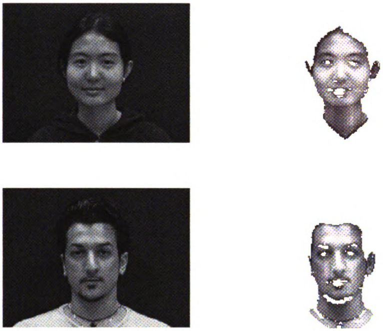


Figure 2.8 Skin segmentation results on three images from VidTIMIT dataset using the non-parametric probability distribution model

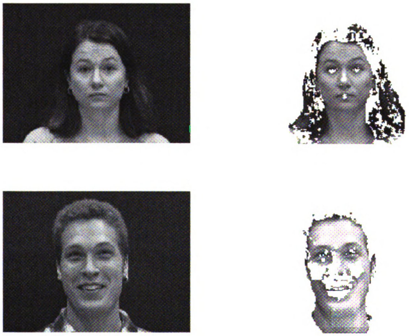


Figure 2.9 Incorrect skin segmentation results on two images from VidTIMIT dataset using the non-parametric probability distribution model

The skin regions from most of the images are accurately segmented as shown in Figure 2.8 . But, few images showed inaccurate segmentation of hair and face regions as seen in Figure 2.9 . False segmentation of hair is due the size of histogram bin. Small histogram bin size requires large training set to estimate the skin probability. Whereas, a large histogram bin size falsely segments neighboring colors as skin. Missing skin segmentation of the face is mostly due to gray tones. Hue estimate in HSI model is inaccurate for gray tones leading to an overlap of skin and non-skin regions. In general, some false calls must be expected, due to the non-seperability of skin from non-skin regions in color space.

2.2.3 Parametric models

The probability distribution of skin color can be estimated in a more compact form using parameters like mean and variance of the skin color. An elliptical Gaussian probability distribution of the skin color vector can be defined as in equation 2.9.

$$p(c \mid skin) = \frac{1}{2\pi \left|\Sigma_S\right|^{1/2}} . e^{-\frac{1}{2}(c-\mu_S)^T \Sigma_S^{-1} (c-\mu_S)} \tag{2.9}$$

where c is a color vector and μ_S and Σ_S are the mean vector and co-variance matrix respectively of the distribution. These model parameters can be estimated by equation 2.10

$$\mu_S = \frac{1}{n}\sum_{j=1}^{n} c_j$$
$$\Sigma_S = \frac{1}{n-1}\sum_{j=1}^{n} (c_j - \mu_S)(c_j - \mu_S)^T \qquad (2.10)$$

where n is the number of skin color training pixels.

The probability measure gives an estimate of how close a given color is to the training skin samples or the mean skin color vector μ_S. The skin segmentation decision can be made if the probability of a color vector exceeds a preset threshold.

Parametric estimation of skin probability can generalize and interpolate the training data better than what is possible in non-parametric estimation. Also, the skin model is more compact with just μ_S and Σ_S describing the entire model compared to the 32x32 matrix of non-parametric model. But, the choice of color space (RGB, HSI) plays a crucial role in Gaussian estimation. The elliptic Gaussian assumption may not hold in different color spaces. If the actual skin color probability distribution has multiple peaks, then a mixture of Gaussians must be used [27]. In general, for skin color estimation either an explicit RGB skin model or non-parametric estimation models provide adequate results.

2.2.4 Non-adaptive Vs Adaptive Models

Skin models can be either non-adaptive or adaptive. A non-adaptive model uses skin samples from training data to generate a static set of rules. If the test skin tone is representative of the training data, then a static model exhibits good performance. But, in a dynamic environment like a moving car, the scene illumination affects the skin chromaticity [19]. The above drawback can be corrected using an adaptive skin model. In an adaptive skin model, skin tone from the current illumination is used to train the model. Any skin region from the current image can be used for training the skin model. For example in a video sequence containing a face, the skin model may be initialized using the skin tone of the regions around the eyes [18].

2.3 Illumination Effects

The skin color captured by a camera depends on the skin pigmentation, brightness and the color of illumination. The same skin area can appear as two different colors under different illuminations. This is a problem commonly referred as color constancy. Human color perception system ensures the color of objects remain relatively constant under varying illumination conditions. But, the computer vision algorithms capture two distinct hues in different illumination conditions as seen in Figure 2.10. The illumination effects on the skin color have to be compensated for, in order to achieve reliable skin color segmentation.

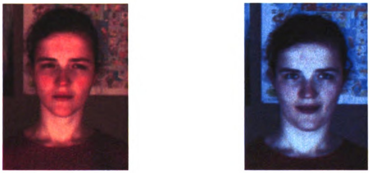


Figure 2.10 Images captured from the same subject under different illumination temperatures. Left image has a correlated color temperature (CCT) of 2600 K and the right image has a CCT of 6200 K [9]

2.3.1 Skin models under varying illumination

The skin models designed in Section 2.2 fail to segment the skin regions under varying illumination temperatures. The explicit model segments all the skin regions under normal illumination conditions in Figure 2.11 (Images are presented in color). But, the skin regions are not segmented when the illumination is changed to a temperature of 6200 K as seen in Figure 2.12 (Images are presented in color).

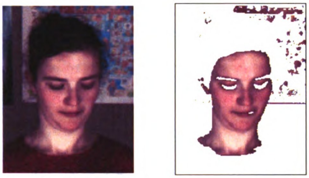


Figure 2.11 Skin segmentation results of explicit model on an image captured at nomal illumination of 3800 K. [15]

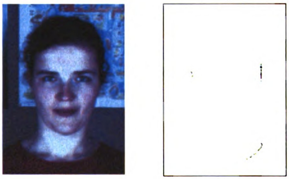


Figure 2.12 Skin segmentation results of explicit model on an image captured at an illumination temperature of 6200 K. [15]

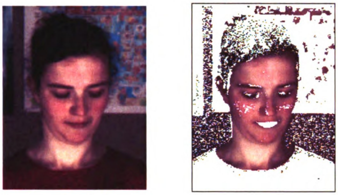


Figure 2.13 Skin segmentation results of non-parametric model on an image captured at an illumination temperature of 3800 K [15]

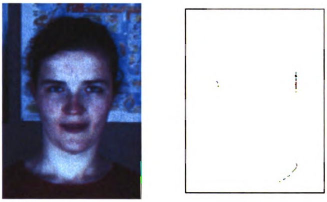


Figure 2.14 Skin segmentation results of non-parametric model on an image captured at an illumination temperature of 6200 K [15]

The segmentation results with the non-parametric distribution model are also similar. All the skin regions are segmented along with some background under normal illumination as seen in Figure 2.13 (Images are presented in color). But, the skin regions are not segmented correctly when the illumination conditions are changed as seen in Figure 2.14 (Images are presented in color).

2.3.2 Compensation Methods

Illumination effects can be partially compensated by any of the following adaptive methods—

- Estimate the illumination vector from the image and remove its effect on the skin chromaticity
- Train the skin model adaptively using only the skin regions of current illumination.

Estimating illuminant color to segment skin is a promising method. Recently, many researchers have reported some success using this method. A dichromatic reflection model is used to understand the reflected light due to skin pigment and the illuminant color. The chromaticity captured by a color camera is due the superposition of body reflections and surface reflections. The body reflections show chromaticity due to skin pigmentation and the surface reflections show the illumination chromaticity. Using the dichromatic model, illuminant color can be estimated from the human skin color as described by Storring et al. [16].

A dichromatic reflection model describes the reflected light from dielectric objects like skin as an additive mixture of light *Ls* reflected from the material's surface and light *Lb* reflected from the material's body. The observed color C_L at a pixel can be written as a linear combination of interface color C_i and body reflection color C_b of the material.

$$C_L = m_i C_i + m_b C_b \tag{2.11}$$

On a uniformly colored surface like skin, the colors C_i and C_i of interface and body do not change. But, the linear combination of both the colors results in a cluster in the color space as shown in Figure 2.15 [39].

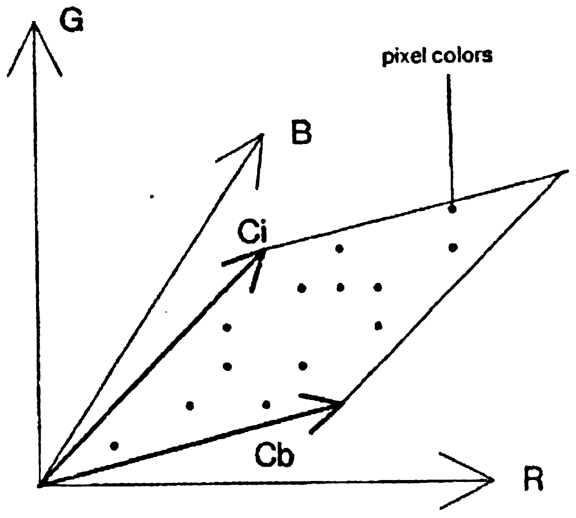


Figure 2.15 Pixels of uniform surface lie on a parallelogram in RGB color space [39]

Using the pixels belonging to face region, both C_i and C_b can be estimated from the color cluster. The illuminant color is simply interface reflection color C_i. Though this method gives an accurate illuminant color, it is computationally intensive.

Another commonly used method is to adaptively train the skin model with the current illumination information. The mean vector of the Gaussian skin model can be modified adaptively to track the face in varying illuminations [17]. The same technique can also be applied to train the non-parametric skin model.

2.4 Adaptive Skin Segmentation

Adaptive skin segmentation combines various modules that are discussed in the above sections. The schematic diagram of an adaptive skin segmentation algorithm is shown in Figure 2.16. The previous few frames (3-4 seconds) are passed through a motion detection filter. The output of this filter masks out regions that are slowly varying. Eye blink is commonly detected by the motion detection filter. Assuming that the background does not change and the face stays predominantly in the center of the frame, potential skin regions are located. These regions are used to update the probability estimate of non-parametric model. The updated skin model segments the skin regions from the latest video frame. The largest connected region is determined using a connected component algorithm. The output is the boundary of the largest connected region outlining the face of the person.

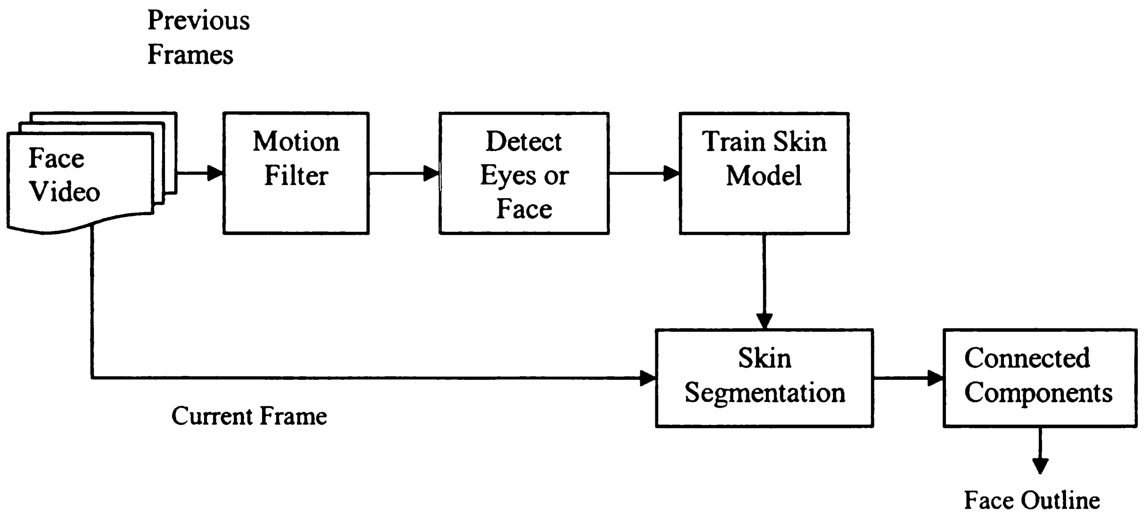


Figure 2.16 Schematic diagram of adaptive skin segmentation

2.4.1 Motion Detection Filter

There are different methods for identifying regions of motion and segment moving objects in image sequences: image differencing, adaptive background and optical flow. The simplest approach is *image differencing*. In this method, corresponding pixel locations in two images are compared and locations where significant changes occur are marked as regions of motion. The algorithm to detect changes between two images is as described below:

Input $I_t[r,c]$ and $I_{t-\Delta}[r,c]$: two monochrome input images taken Δ seconds apart. Input θ is an intensity threshold.

$I_{out}[r,c]$ is a binary output image; B is a set of bounding boxes.

1. For all pixels [r,c] in the input images,

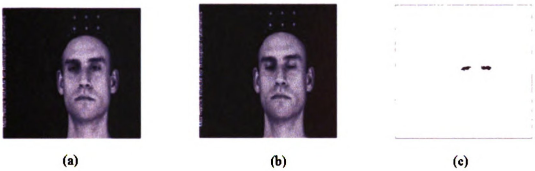
   set $I_{out}[r,c] = 1$, if $(\left|I_t[r,c] - I_{t-\Delta}[r,c]\right| > \theta$

   set $I_{out}[r,c] = 0$ otherwise.

2. Perform a closing of the I_{out} using a small disk to fuse neighboring regions.
3. Perform connected components extraction on I_{out}
4. Remove small regions assuming they are noise.
5. Compute the bounding boxes of all remaining regions of changed pixels.
6. Return $I_{out}[r,c]$ and the bounding boxes B of regions of changed pixels.

The above described motion filter is applied to the data from a video clip with the subject closing his eyelids. This video is used with permission from the "Face Video Database of Max Plank Institute for Biological Cybernetics", Tuebingen, Germany. Two frames are shown in Figures 2.17(a) and (b) with the eyes open and closed. Image differencing is suited for detecting blinking; because humans have mean-blink duration of 51.9 ms [28]. The rapid change in eyelids creates a motion in successive frames. The output of the motion filter is shown in Figure 2.17(c). The output here contains only the eyelid regions, as these are the only significant changes between these two frames.

Based on the location of the eye regions, four skin regions around the eyes indicated in white boxes are extracted for skin modeling as seen in Figure 2.17(d). Figure 2.15 (e) shows the extracted skin regions. The adaptive skin model is trained on these extracted skin samples and used for detecting the face region on the future video frames.



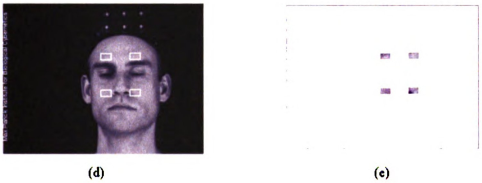


Figure 2.17 Motion Filtering and skin region extraction. (a) and (b) Two frames in the video with the eyes open and closed. (c) Output of the motion filter. (d) Extracted skin regions in white boxes. (e) Extracted skin regions for skin modeling.

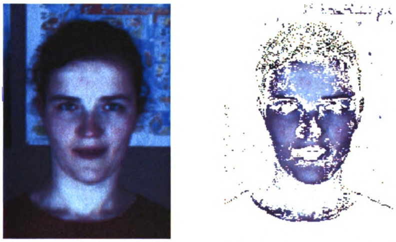


Figure 2.18 Adaptive skin segmentation results of non-parametric model on an image captured at an illumination temperature of 6200 K.

Adaptive skin segmentation in varying lighting conditions using a non-parametric histogram model is as shown in Figure 2.18 (Images are presented in color).

2.4.2 Face Detection

Face detection can be described as a problem of trying to outline the face region from a given image. In the present scenario, the video captures predominantly the face of the driver. Therefore, it is sufficient to identify the largest connected skin region and label it as the face.

Morphological operation "*closing*" is applied on the segmented binary image. This operation fuses neighboring skin regions. After the fusing operation, small regions are removed by applying connected components algorithm elaborated in the Appendix A. Face detection results are shown in Figures 2.19 (a-h).

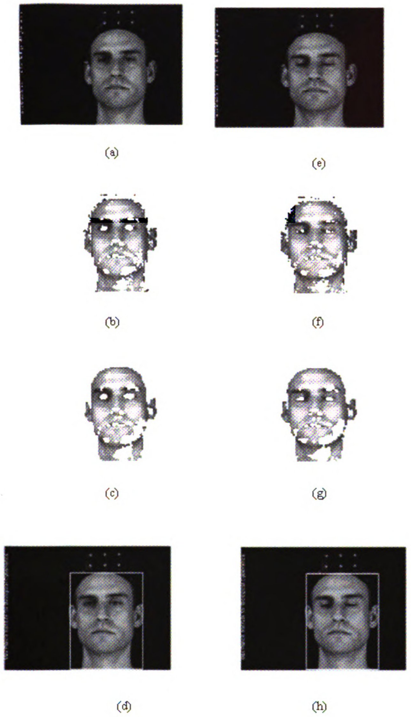


Figure 2.19 Skin segmentation and face detection outputs. (a) and (e) Two frames in a video sequence with the eyes open and closed. (b) and (f) Segmented output using the skin model. (c) and (g) Output after applying morphological operation and connected components algorithm. (d) and (h) Outline of the detected face

Figures 2.19 (d) and (h) are the final outputs of the face detection. They contain the boundary of the face region. The next step is to detect the eyes inside the boundary of the face.

2.4.3 Eye Detection

The problem of detecting the eyes from a video sequence can be described as eye detection. Eye detection is necessary, as computing the PERCLOS features calls for continuous tracking of the eyes of the driver. Eyes may be detected using a combination of the motion detection filter output and the face detection output.

Motion detection filter provides a very good segmentation of the eye regions as seen in Figure 2.17 (c), when the subject blinks. As the average human blink rate is approximately 12 blinks per minute [14], the motion detection filter can initialize the eye regions fairly often. The output of the face detection can be used to limit motion processing to the face region. The schematic diagram of the eye detection method is shown in Figure 2.20.

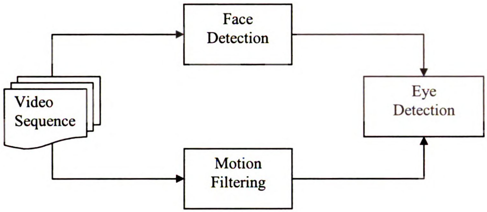


Figure 2.20 Schematic Diagram of Eye Detection

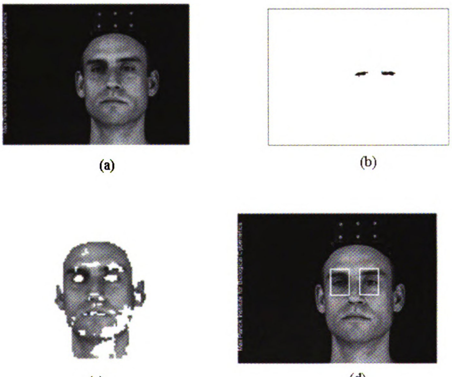


Figure 2.21 Results of eye detection. (a) Image from a video sequence (b) Motion filter output (c) Face detection output (d) Eye detection output

Figure 2.21 shows the results of eye detection algorithm. Output of motion detection filter is shown in Figure 2.21 (b). Face detection and eye detection outputs are shown in Figure 2.21 (c) and (d) respectively. Excessive motion of the face due to turning or bending may cause the algorithm to fail. Motion detection filter might detect multiple regions on the face as potential eye regions. This problem can be partially solved using information such as the location and size of the eyes.

After successful detection of face and eyes, the next step is to track them both continuously. Tracking the face and eyes of a person is elaborated next in Chapter 3.

Chapter 3 Face and Eye Tracking

"Tracking an object" is the ability to follow a specific region or object in a video. Figure 3.1 shows an example of tracking an eye. Face tracking algorithms track the movements of the face or eyes in a video sequence. These algorithms can serve as a pre-processing stage for other facial image analysis tasks such as PERCLOS estimation, face recognition, gesture recognition and gaze tracking. Ideally, a face detection algorithm can be repeatedly applied on each frame of a video to track the face. But, a face detection algorithm is computationally intensive and cannot keep track of the face in real-time. On the contrary, face tracking algorithms use temporal correlation to track a face in real-time.

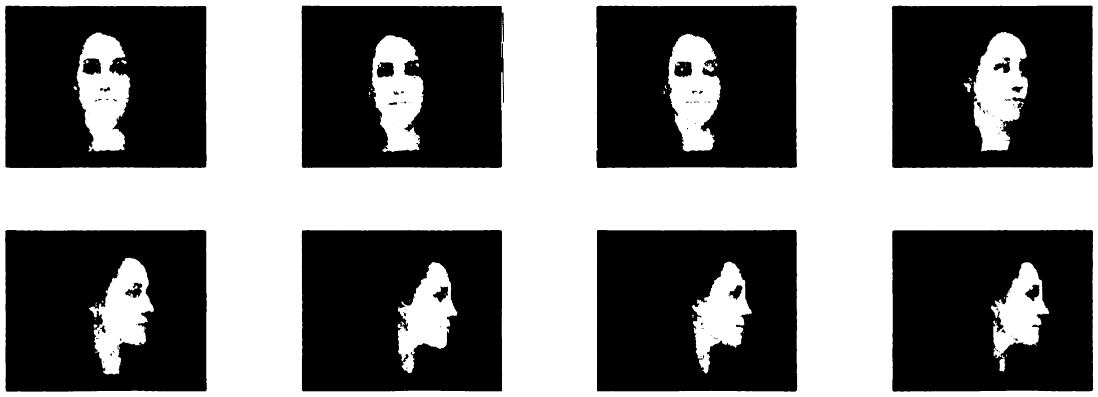


Figure 3.1 Tracking an eye region in a video sequence.

Face tracking algorithms must track a face through multiple sources of variations such as noise, scene illumination, occlusion effects etc. If the algorithm loses track of the face, there must be an initialization process that allows the algorithm to restart the tracking. Therefore, block diagram of a continuous tracking system contains both the detection and tracking modules as shown in Figure 3.2.

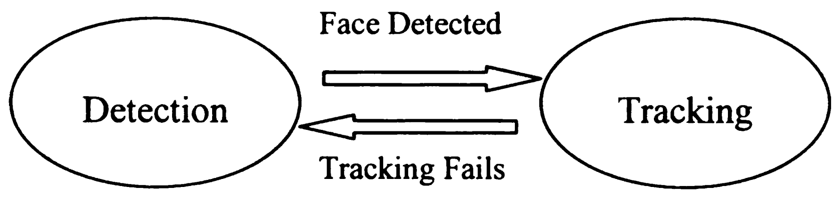


Figure 3.2 Block diagram of a real-time tracking system

Tracking can be performed using either templates or motion trajectories. Commonly used methods for tracking an object are –

- Template Image based tracking
- Level-Set based tracking

3.1 Template Image based Tracking

A template image which is representative of the object of interest is first generated by the detection algorithm. This template is matched with future video frames to estimate the location of the object. For example, template image of an eye region can be matched with the subsequent video frames as shown in Figure 3.3.

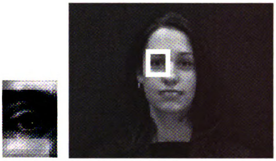


Figure 3.3 Template Image based tracking. (a) Eye template (b) Eye region matched in a video frame

Template matching method is based on the following assumptions

- the object in the template is either stationary or moves slowly compared to the video acquisition rate
- the objects in the scene are rigid and cannot change their three-dimensional orientation
- the object in the template cannot repeat in the scene. Repeated objects will results in multiple template matches

Depending on the problem, additional assumptions can be introduced. All the above assumptions are satisfied to some extent by the face image sequences. The movement of face or eyes is common, but it is a slow process compared to the video frame rate of 30 frames per second. Face is bisymmetric in nature, causing the eye region to repeat. To overcome this problem, search area must be restricted to one half of the face region.

The block diagram for template-matching algorithm is shown in Figure 3.4. The detection stage makes use of skin detection model to isolate either the eyes or face as discussed in

Chapter 2. A template of the required region (eyes, face) is then extracted from the detection results as shown in Figure 3.5. Template-matching algorithm finds the location in the image data, which best matches with the template image. If the matching location is successfully found, then template-matching algorithm is repeated for successive images. Otherwise, the tracking algorithm restarts with a new template image.

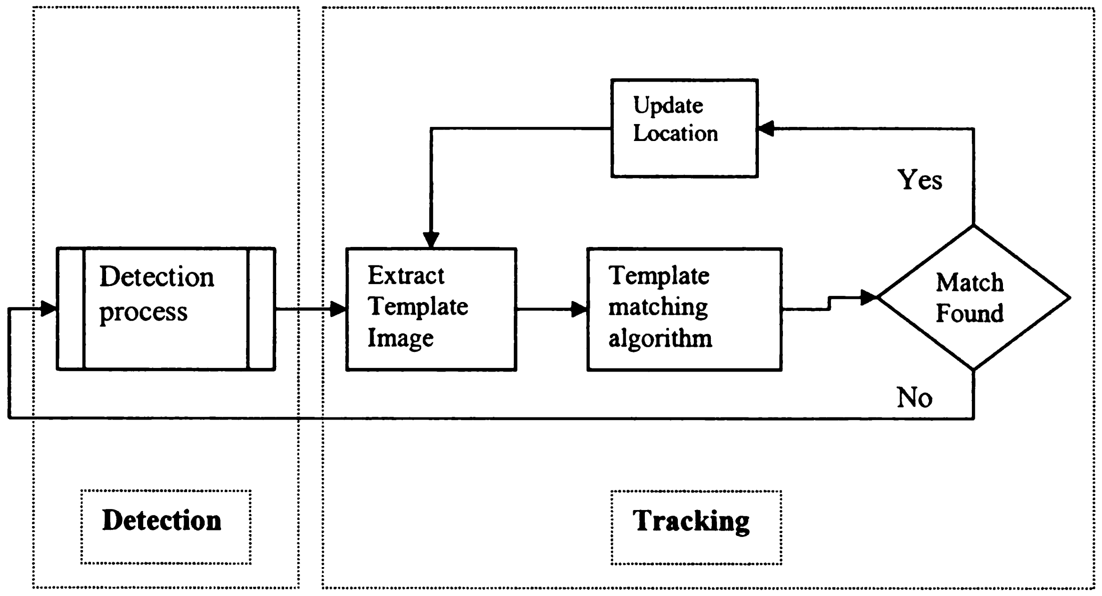


Figure 3.4 Block diagram of template-matching algorithm

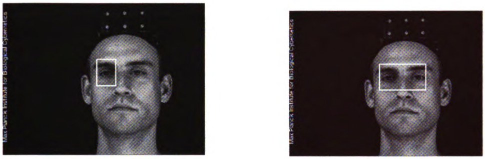

Max Planck Institute for Biological Cybernetics
Max Planck Institute for Biological Cybernetics

Figure 3.5 Examples of template eye images enclosed in the white rectangles

In general, any template-matching algorithm can be characterized by the following stages --

- Choosing a template region in the detection process
- Template Features for generating a template image
- Similarity or Distance measure for comparing a template with the image in template-matching algorithm.

The first stage of choosing a template region was discussed in detail in the previous chapter. The second and third stages are discussed in the subsequent sections of this chapter.

3.1.1 Template Features

A template image can be composed of any feature that is characteristic of the object being tracked. The choice of template feature is critical in a tracking algorithm. The feature must be insensitive to noise and other variations like illumination. The simplest feature is the intensity of a pixel. Figure 3.6 shows an eye template based on the intensity feature. Other local features such as edge elements or chromaticity components may be used depending on the application.

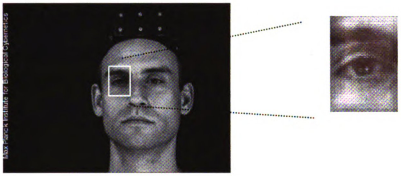


Figure 3.6 Eye template based on the pixel intensity

The template features commonly used for tracking eye are –

- Intensity feature
- Skin probability feature
- Combination of Intensity and Skin probability features

1. The *intensity feature* creates a sub-image from the original image. Template-matching with a sub-image is the well known *Image Registration* problem. If the template area has distinct edges as in the eye region, the intensity feature performs well. The intensity feature however is sensitive to variations due to pixel noise and illuminations effects.

2. An estimate of the *skin probability* can be used as an alternate template feature as expressed in equation 3.1. Two-dimensional histogram of color components provides a distinct skin probability profile around the eyes. These skin color variations in the eye

area are unaffected by small intensity changes and pixel noise. Figure 3.7 shows a template generated using the skin probability feature. The extracted template shows the skin probability in the eye area.

$$T(x,y) = P(skin \mid x,y) \tag{3.1}$$

where $T(x,y)$ is the template feature at location (x,y) and $P(skin| \ x,y)$ is the probability of skin given the (x,y) location.

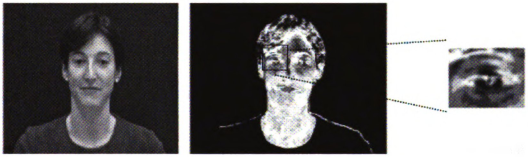


Figure 3.7 Eye template created using the skin probability feature

3. A new feature can be derived by combining both the intensity and skin probability features. The skin probability estimate could be multiplied with the pixel intensity as depicted in the equation 3.2. The resulting feature puts emphasis on the intensity variations without the disadvantages of illumination effects. An eye template created using the new feature is shown in Figure 3.8.

$$T(x,y) = I(x,y).P(skin \mid x,y) \tag{3.2}$$

where *T(x,y)* is the template feature at location (*x,y*); *I*(*x,y,*) is the intensity of a pixel; P*(skin| x,y)* is the probability of skin given a (*x,y*) location;

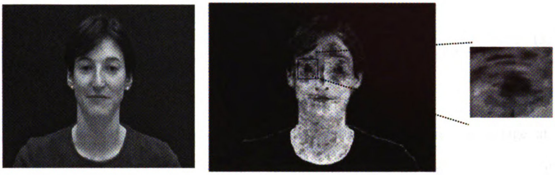


Figure 3.8 Eye template created using the combined feature

3.1.2 Template-Matching using Cross-Correlation

Cross-correlation is a standard method of estimating the degree to which two signals or images are correlated. In order to identify a 2D pattern in an image, the template is correlated across the image. The locations with high correlation are the regions where the template matches with the image.

Template-matching algorithm –

Let T(*x,y*) represent a two-dimensional template that must be found in a larger image I(*x,y*).

Position the template (or mask) sequentially throughout the image and compute linear cross-correlation at each position as shown in Figure 3.9. The correlation coefficient at a position (*m*,,*n*) is given by equation 3.3

$$\rho(m,n)=\frac{\sum\limits_{x,y}(I(x,y)-\overline{I(x,y)})(T(x-m,y-n)-\overline{T})}{(\sum\limits_{x,y}(I(x,y)-\overline{I(x,y)})^2.\sum\limits_{x,y}(T(x-m,y-n)-\overline{T})^2)^{1/2}} \tag{3.3}$$

The numerator estimates the correlation between the template and the image at a specified position (*m*,*n*). The correlation is normalized with the image energy in the denominator. The ratio is the cross-correlation coefficient ρ at a location (*m*,*n*), where $-1<\rho<+1$. Equation 3.3 creates a correlation coefficient matrix ρ over the entire image *I*.

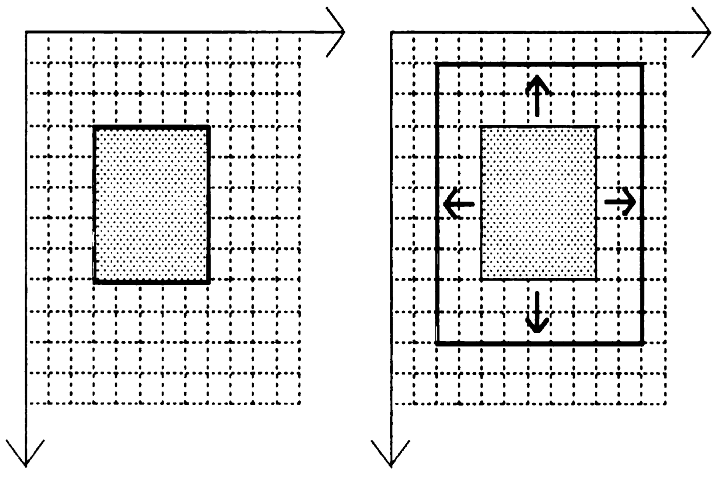


Figure 3.9 Cross-correlation of the dotted template on a larger image. The arrows indicate shifting of the template window

Find the position of maximum correlation coefficient in the ρ matrix. This coefficient must satisfy a set of conditions in order to be a template match.

- The peak vale of the correlation coefficient matrix should reside on an ε-neighborhood of the center of the matrix, where ε is a 5 x 3 window.
- The maximum correlation coefficient must be greater than a threshold θ, where $0 < \theta < 1$
- There cannot be any other local maxima in the correlation matrix within $p * \sigma$ of the absolute maximum coefficient, where σ is the standard deviation of the correlation coefficient matrix and p is a fraction ranging from 0 to 1. This condition ensures that the global maximum is dominant compared to the local maxima.

The above conditions ensure that the template location shifts slowly and there is only one dominant maximum coefficient in the correlation matrix ρ. If the above conditions are satisfied, then template-matching is a success. Tracking window will be updated to the new location of the template match.

An example of cross-correlation based template matching is shown in Figure 3.10. Figure 3.10 (a) is an open eye template T that must be tracked in the face image I. Figure 3.10 (b) is the face image representing the tracking search area. Correlation matrix ρ is plotted as an intensity image in Figure 3.10 (c). The two areas with bright intensities are the eye regions. Figure 3.10 (d) shows a surface plot of the correlation matrix. Notice that the maximum correlation coefficient is located at the center of left eye. In a face tracking algorithm, the search area is restricted to only one half of the face. Surface plot corresponding to the left half of correlation matrix is shown in Figure 3.10 (e). The left correlation matrix satisfies all the conditions for a template match.

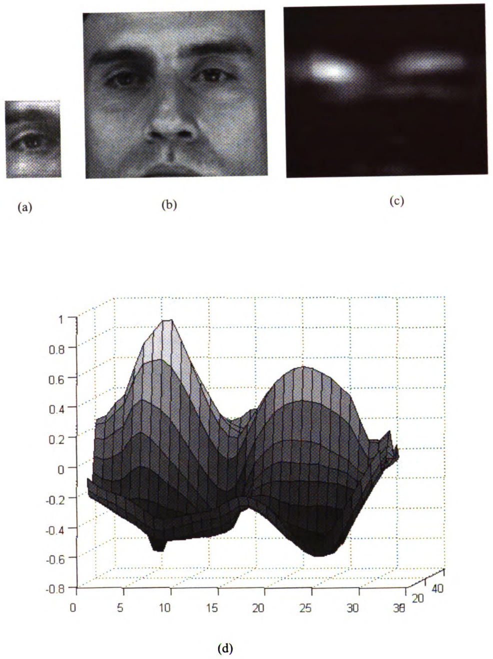
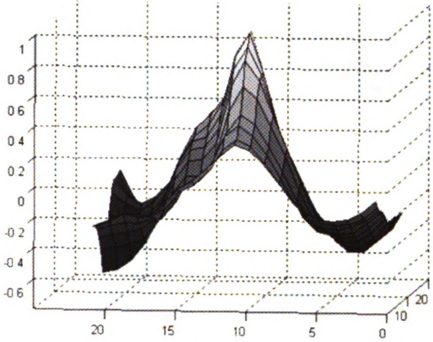


Figure 3.10

Figure 3.10(continued)

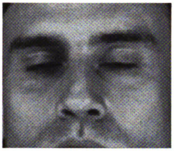


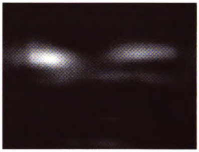
(e)

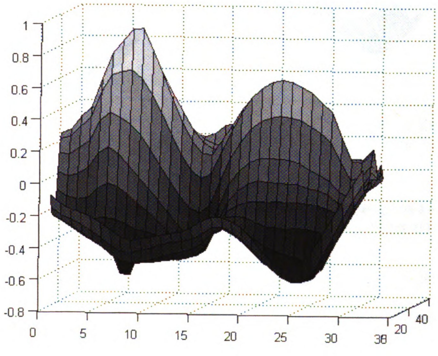
Figure 3.10 Cross-correlation based face tracking. (a) Template Eye Image (b) Face Image (c) Correlation Matrix (d) Surface plot of correlation matrix. (e) Surface plot of left half of the correlation matrix

Another example of template matching of an eye when the eye is closed is shown in Figure 3.11. Figure 3.11 (a) is an open eye template *T* that must be tracked in the face image *I* with the eyes closed. A maximum correlation coefficient is centered at the left eye in spite of the eyes in the face image being closed.

(a)


(b)

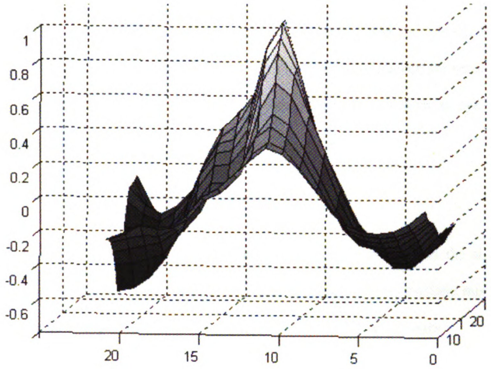

(c)



(d)

Figure 3.11

Figure 3.11(continued)



(e)

Figure 3.11 Cross-correlation based face tracking. (a) Template Eye Image (b) Face Image with eyes closed (c) Correlation Matrix (d) Surface plot of correlation matrix. (e) Surface plot of left half of the correlation matrix

The above results show that tracking an eye is possible in a real situation with eye blinks and face rotation. The results of eye-tracking on a 15 second video sequence are presented in the next section.

3.1.3 Results

Template based tracking was applied on various videos of VidTIMIT database. Video sequences with faces rotating sideways and upwards are chosen to test the robustness of the tracking algorithm. Figure 3.12 shows eye tracking results on a video sequence. Pixel intensity is used as the template feature and the cross-correlation parameters θ, p are 0.7

and 0.5 respectively. After the initialization of eye template, tracking did not fail for the entire video sequence of 15 seconds. Tracking between each frame (384 x 512) takes an average of 0.0757 seconds on a standard computer with 2GB RAM and 1.6GHz Pentium processor. With this speed, a maximum of only 13 frames can be tracked per second. This tracking speed must be improved to facilitate other computations such as eye closure classification and PERCLOS estimation.

Tracking speed can be improved in one of two ways –

- Skip alternate frames for tracking analysis. This method improves the algorithm efficiency roughly by a factor of two. This will lead to more tracking misses due to abrupt changes, resulting in extra initializations of detection stage.
- Reduce the resolution of the video data for tracking analysis. This method can improve the algorithm efficiency depending on the reduced resolution. It was found that tracking is largely unaffected by reducing the resolution by a factor 2 or even 4.

Tracking the quarter resolution frames (96 x 128) takes an average of 0.0192 seconds, an improvement factor of 4 over the full resolution frames. A maximum of 52 frames per second can be tracking with the improved speed. Figure 3.13 shows the eye tracking results on frames with quarter resolution. The tracking results remained unaltered with the quarter resolution frames.

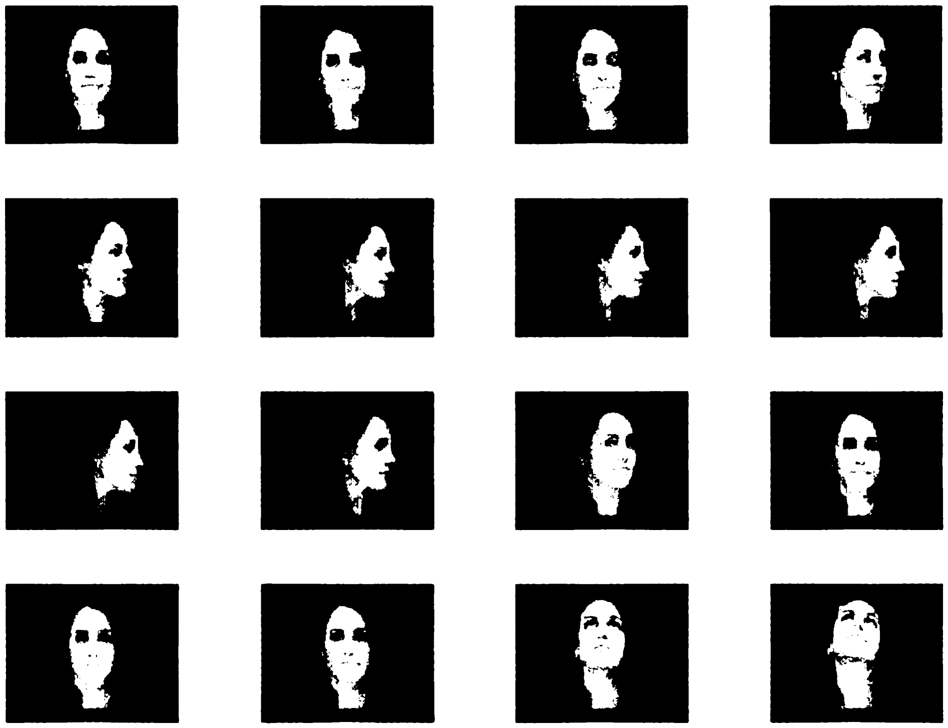


Figure 3.12 Face tracking results with pixel intensity feature. White rectangle shows the tracking of left eye.

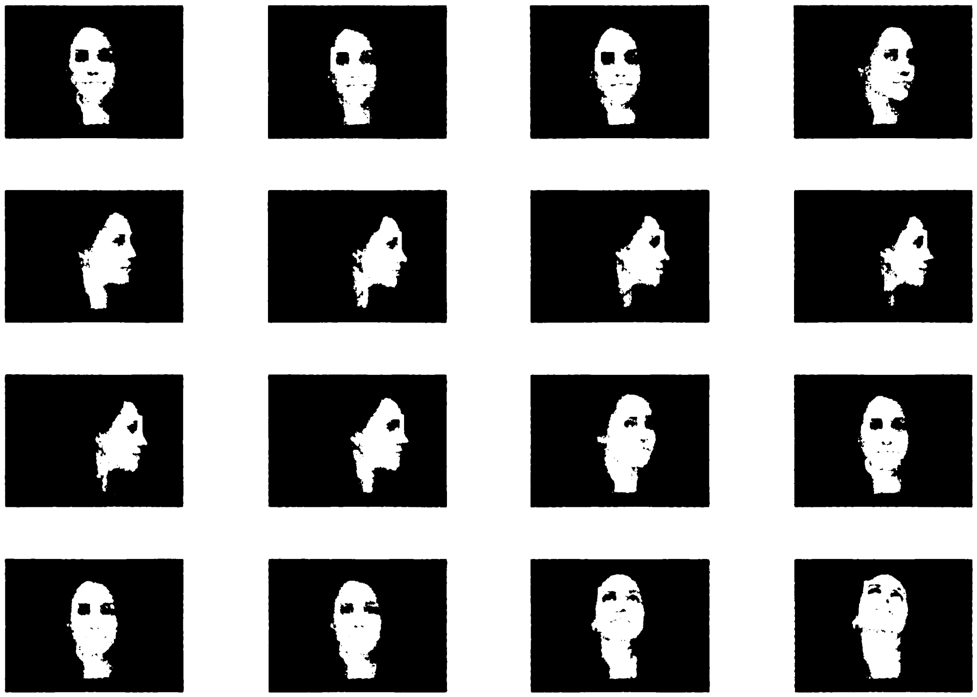


Figure 3.13 Face tracking results with pixel intensity feature on quarter resolution frames. White rectangle shows the tracking of left eye.

Tracking results are fairly similar with the other two features discussed earlier such as skin probability and the combined feature. Figure 3.14 and Figure 3.15 show the tracking results with the skin probability feature and combined skin feature respectively on quarter resolution frames. The drawback with both these features is the algorithmic efficiency. Even with quarter resolution video data, both these methods take 0.1102 seconds to track each frame. The maximum number of frames that can be tracked per second reduces to 9. Therefore, the pixel intensity feature is preferred over the other features.

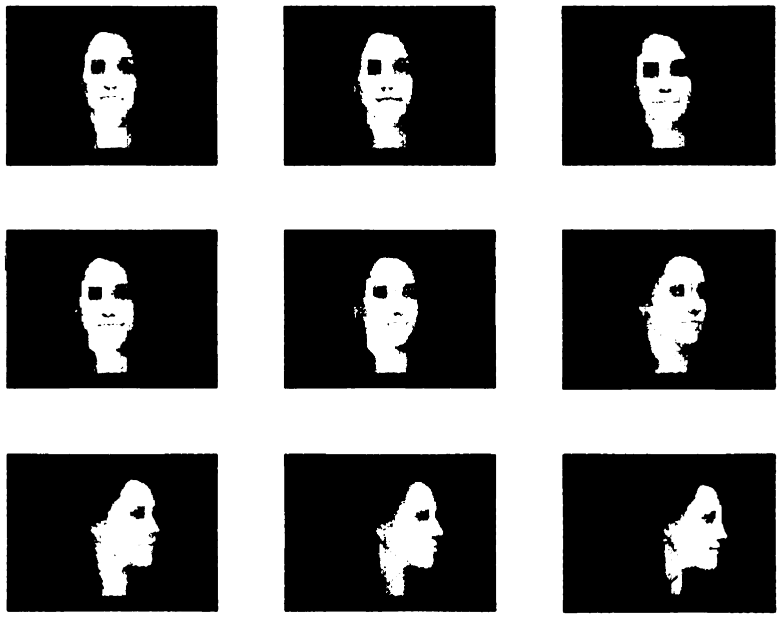


Figure 3.14 Face tracking results with skin probability feature on quarter resolution frames. White rectangle shows the tracking of left eye.

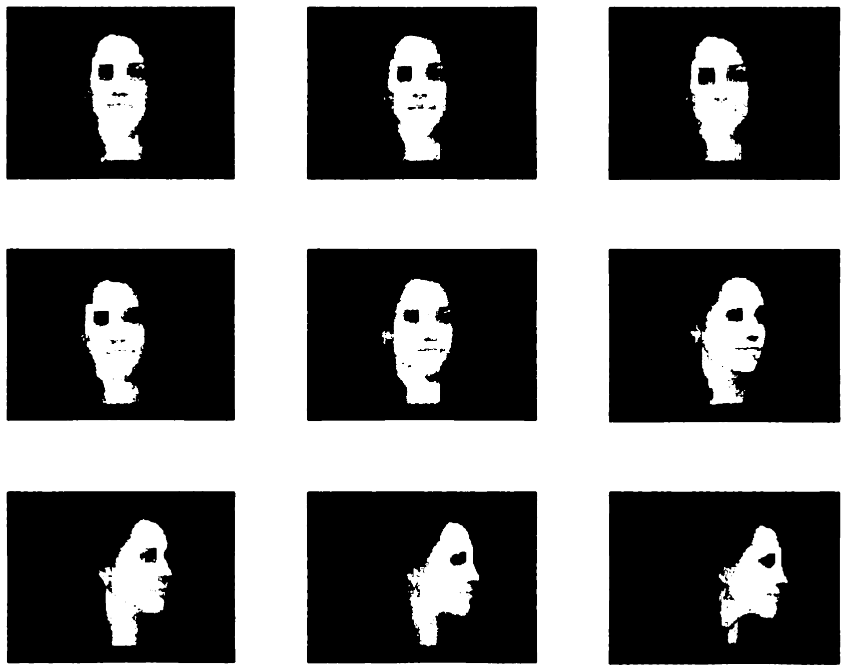


Figure 3.15 Face tracking results with combined skin feature on quarter resolution frames. White rectangle shows the tracking of left eye.

Tracking can fail due to various reasons such as occlusion and sudden changes in the object of interest. In eye tracking, if the face rotates sideways and the eye is completely occluded, then tracking fails. Cross-correlation method cannot find a maximum correlation coefficient and the eye template has to be reinitialized.

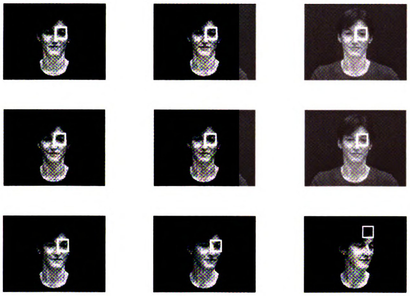


Figure 3.16 Face tracking results with pixel intensity feature on quarter resolution frames. White rectangle shows the tracking of left eye. Tracking fails at the last frame.

Figure 3.16 shows an example of tracking failure due to occlusion of the eye. In the last frame, eye is partially occluded by the nose resulting in a tracking failure.

3.2 Level Set based Tracking

Level Set method is a numerical technique for tracking curves. Unlike template tracking, level set method tracks the exact region of interest using curves.

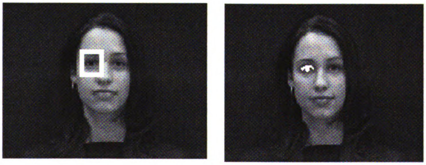


Figure 3.17 Differences between template and level set tracking. (a) Template tracking of the left eye region. (b) Level set tracking of the left eye region.

Figure 3.17 shows the difference between template tracking and level set tracking. In template tracking, shape of the template remains unchanged, while the location of template moves between each frame. But, in level set tracking the region is tracked continuously with small changes to the curve in each frame. The advantage of level set method is that the motion of curve can be controlled with numerical computations on a fixed Cartesian grid without parameterizing the curve [31]. In addition, level set method permits changing topology as encountered in merging of two objects without parameterization [31].

3.2.1 Level Set Representation of Curves

Curves in two-dimensional space can be defined as an interface that separates $\Re^2$ domain into two subdomains with non-zero areas. The above definition is valid only for *closed* curves, with clearly defined interior and exterior regions. For example, consider $\phi(\vec{x}) = x^2 + y^2 - 1$, where the interface (curve) defined by $\phi(\vec{x}) = 0$ is the unit circle $\partial\Omega = \{\vec{x} \mid |\vec{x}| = 1\}$. The interior region is the unit disk $\Omega^- = \{\vec{x} \mid |\vec{x}| < 1\}$, and the exterior region is $\Omega^+ = \{\vec{x} \mid |\vec{x}| > 1\}$. These regions are shown in Figure 3.18.

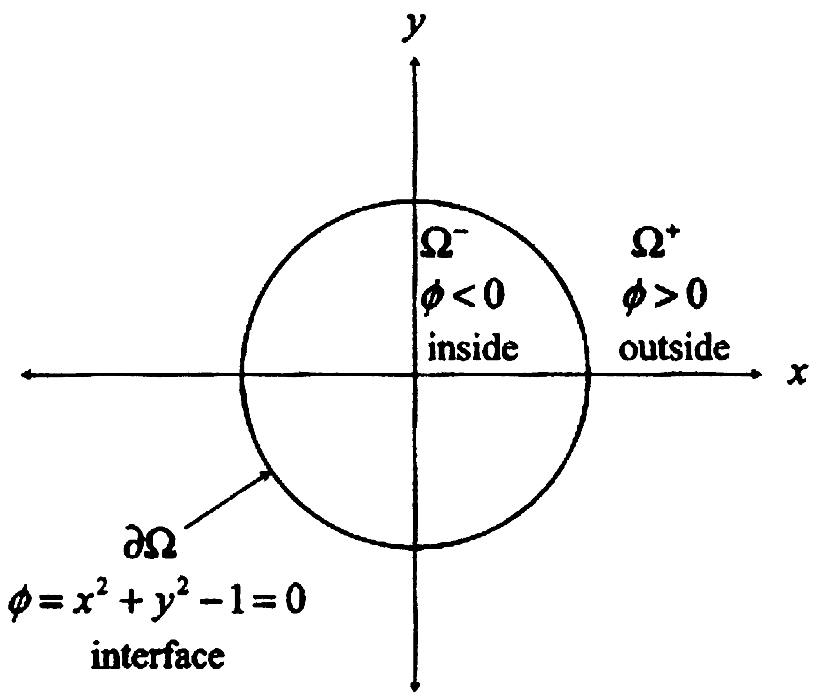


Figure 3.18 Level set representation of the curve $x^2+y^2=1$. Exterior and interior regions are represented by Ω^+ and Ω^- respectively. $\partial\Omega$ represents the interface $\phi(\vec{x})=0$.

The explicit representation of this curve is simply the unit circle defined by $\partial\Omega=\{\vec{x}\,|\,|\vec{x}|=1\}$. On the contrary, level set offers an implicit representation of the curve *C*, using a surface function $\phi(\vec{x})$ as depicted in equation 3.4.

$$\mathrm{C}=\{x\in\Re^2:\phi(\vec{x})=0\} \tag{3.4 (a)}$$

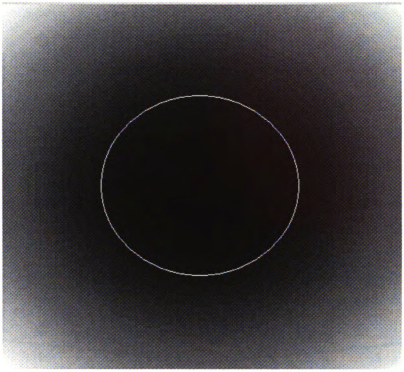
$$\phi(\vec{x})=x^2+y^2-1 \tag{3.4 (b)}$$

Explicit representation of curves

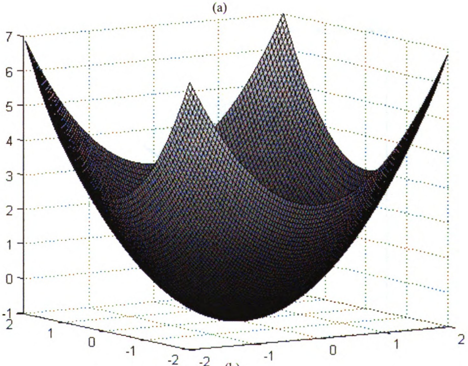
In two spatial dimensions, the explicit curve definition needs to specify all the points on the curve C. While it easy to do so with a unit circle, it can be more difficult for general arbitrary curves. In general, one needs to parameterize the curve with a two-dimensional vector function $\vec{x}(s)$, where the parameter s belongs to $[s_o, s_f]$. The condition that the curve be closed implies that $\vec{x}(s_o) = \vec{x}(s_f)$. The parametric representation of the unit circle in Figure 3.18 is $\vec{x}(s) = [\cos(s), \ \sin(s)]$, where s goes from 0 to 2π. A convenient way of approximating an explicit representation is to discretize the parameter s into a finite set of points $s_o < \cdots < s_{i-1} < s_i < s_{i+1} < \cdots < s_f$, where the subintervals $[s_i, s_{i+1}]$ are not necessarily of equal size. For each point s_i in the parameter space, we then store the corresponding two-dimensional location of the curve denoted by $\vec{x}(s_i)$. As the number of points in the discretized parameter space is increased, so is the resolution of the two-dimensional curve.

Implicit representation of curves

Implicit representation of the curve C can be stored with a discretization of the entire $\Re^2$ domain, which is impractical. Instead, discretizing a bounded subdomain $D \subset \Re^2$ is sufficient. Within this domain, we choose a finite set of points (x_i, y_i) for $i = 1, \ldots, N$ to discretely approximate the implicit function ϕ as shown in Figure 3.19.

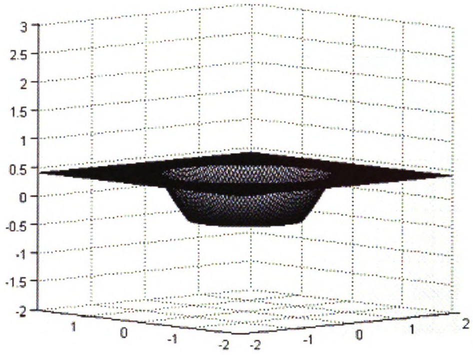


(a)

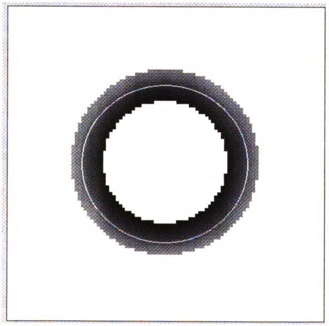


(b)

Figure 3.19 (a) Implicit function $\phi(\vec{x})$ of a unit circle. Circle at $\phi(\vec{x}) = 0$ is highlighted in white. (b) Surface plot of $\phi(\vec{x})$ in the domain [-2, -2] to [2, 2].



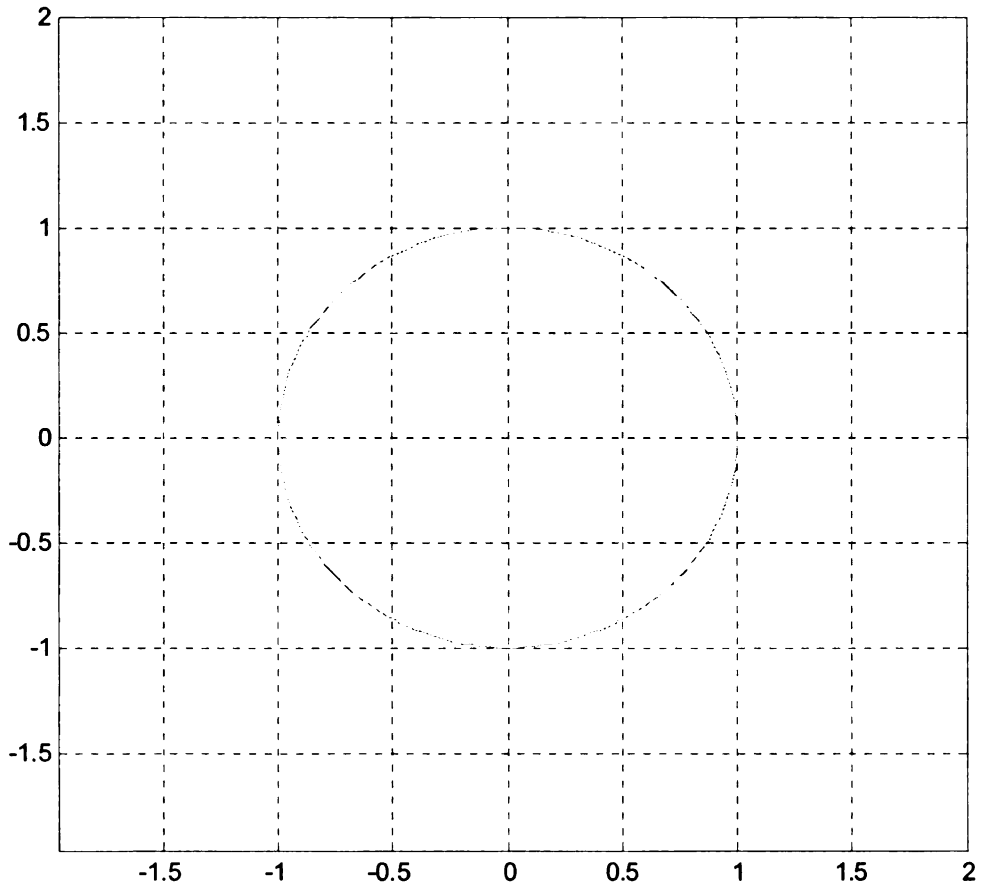
(a)



(b)

Figure 3.20

Figure 3.20 (continued)



(c)

Figure 3.20 (a) Discretized implicit function $\phi(\vec{x})$ of a unit circle. Circle at $\phi(\vec{x}) = 0$ is highlighted in white. (b) Surface plot of discretized $\phi(\vec{x})$ in the domain [-2, -2] to [2, 2]. Only the region around unit circle is discretized. (c) Unit circle

This is a drawback of the implicit surface representation. Instead of resolving a one-dimensional interval $[s_o, s_f]$, one needs to resolve a two-dimensional region D. This can be avoided in part, by placing all the points $\vec{x}$ very close to the interface, leaving the rest of D unresolved. Since only the curve, $\phi(\vec{x}) = 0$ is important, only points $\vec{x}$ near this curve are needed to accurately represent the curve. The rest of the domain D is unimportant. Clustering points near the curve is a common approach to discretize implicit representation. Once we have chosen the set of points that make up our discretization, we store the values of the implicit function $\phi(\vec{x})$ at each of these points. For example a unit circle in implicit representation can be discretized as shown in the Figure 3.20. Figure 3.20 (a) shows the small cluster around the unit circle that is discretized. A surface plot of $\phi(\vec{x})$ with the implicit surface representation is shown in Figure 3.20 (b). Figure 3.20 (c) shows the unit circle curve $\phi(\vec{x}) = 0$.

3.2.2 Level Set Curve Evolution

Tracking requires not just the ability to represent a curve *C*, but also to modify it continuously in each frame. The changes in curve *C* in each frame can be modeled using the velocity in the normal direction. The curve evolution differential equation of an initial implicit surface ϕ and a speed field *F* is given by

$$\frac{d\phi}{dt} + F|\nabla\phi| = 0 \tag{3.5}$$

where *F* is the speed of the curve in the normal direction $\vec{N}$ and pointing outward. Figure 3.21 shows the normal direction on a unit circle.

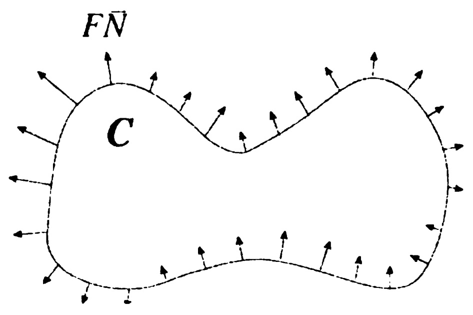


Figure 3.21 Evolution of curve *C* under speed $F\vec{N}$

The speed field F can be either positive or negative. When $F > 0$ the curve moves in the normal direction and when $F < 0$ the curve moves opposite the normal direction. Speed F is specific to a problem and controls the evolution of curve C. It is generally composed of two parts: an external speed derived from image data and an internal speed that depends on the geometric properties of curve C. Speed F can be expressed as a combination of external and internal speeds as in equation 3.6.

$$F = a - b\kappa \tag{3.6}$$

where a is the external speed and $b\kappa$ is the internal regularization term. Using equations 3.5 and 3.6, we can convert any image tracking problem into a level set curve evolution problem. Equation 3.5 can be solved by numerical techniques after recognizing that it is a Hamilton-Jacobi equation [31].

The advantages of level set evolution come from the implicit function representation. Instead of the parameterized curve, discretizing the implicit surface $\phi(\vec{x})$ helps in handling sharp corners and topological changes. For example, a curve representing one interior region splitting into two interior regions is shown in Figure 3.22. From left to right Figure 3.22 (a) – (c), the curve representing gray interior region splits into two. It is very hard to describe this topology change using parameterization. Whereas level set technique uses a well defined implicit surface in Figure 3.22 (d) – (g), and the topology changes naturally.

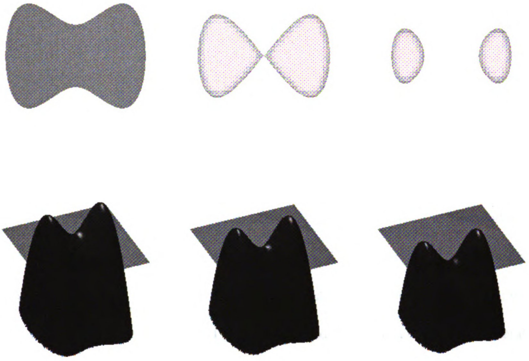


Figure 3.22 Topology changes in level set methods. (a) Curve with one interior region. (b) Interior region just before splitting (c) Curve with two interior regions. (d) – (g) Level set representation of the curve with an implicit surface and changing plane of interest. [43]

3.2.3 Level Set method implementation

The implementation of level set method evolution requires discretization of an implicit surface ϕ on a Cartesian grid of size (Δx, Δy, Δz).

$$\phi_t + (F\frac{\phi_x}{|\nabla\phi|}, F\frac{\phi_y}{|\nabla\phi|}, F\frac{\phi_z}{|\nabla\phi|}).\nabla\phi = 0 \tag{3.7}$$

The derivatives in the equation 3.5 are approximated using finite difference techniques as shown in equation 3.8 [31].

$$\begin{aligned} \frac{\partial \phi}{\partial x} &\approx \frac{\phi_{i+1} - \phi_{i-1}}{2\Delta x} \\ \frac{\partial^2 \phi}{\partial x^2} &\approx \frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2} \end{aligned} \tag{3.8}$$

Level set evolution starts with a well-defined implicit surface of any standard shape. Signed distance functions are a preferred choice due to the existence of first and second derivatives at all points on the grid. At each iteration, ϕ is updated using the equation 3.5. The level set evolution stops when ϕ_t reaches a small value indicating that ϕ is close to the solution. While level set method gives us a good solution, the number of iterations results in a significant computational burden [32].

Evolution process can be accelerated by updating equation 3.5, only around a local neighborhood of the curve [33]. The narrow-band level set methods reduce the computation time significantly. Furthermore, in image tracking problem, the goal is to extract the object boundary after each frame. Here, the evolution process of the level set function is not of interest. By taking advantage of this, a fast level set method without solving Partial-differential equations is proposed by Shi et al, 2005 [34].

The aim of the fast level set algorithm is to evolve the just the curve C, instead of the surface ϕ until it converges to the optimal solution. Instead of defining the surface ϕ, the curve is represented by two lists of boundary points L_{in} and L_{out} as shown in Figure 3.23. These list points are updated continuously until the optimal solution is reached.

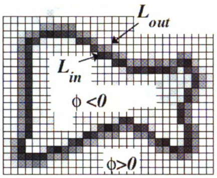


Figure 3.23 Level set surface representation using boundary lists.

For faster computation integer values are chosen for surface ϕ.

$$\phi(\mathrm{x}) = \begin{cases} 3, & \text{if x is outside C, and x} \notin \mathrm{L_{out}}; \\ 1, & \text{if x} \in \mathrm{L_{out}}; \\ -3, & \text{if x is outside C, and x} \notin \mathrm{L_{in}}; \\ -1, & \text{if x} \in \mathrm{L_{in}}; \end{cases} \tag{3.9}$$

The above method is two orders of magnitude faster than ordinary level set evolution for image tracking problems. This accelerated evolution enables us to use level set tracking on a video with 25 frames per second.

3.2.3 Results

Level set tracking requires initialization for the first curve and the speed field F for the subsequent frames. Initialization must provide a starting curve with speed $F < 0$.The speed field F can be a binary mask representing any feature of interest. For example eyes can be tracked by using the darkness of the pupil region. Speed field F for the eyes can be created by thresholding the pixel intensity as shown in Figure 3.24.

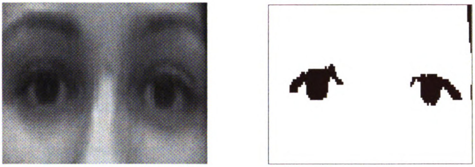


Figure 3.24 Speed field *F* in level set tracking of eyes. (a) Pixel intensity (b) Speed field *F* - Thresholded pixel intensity.

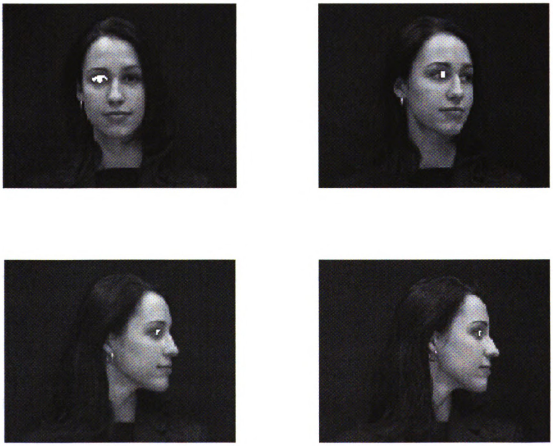


Figure 3.25 Level set tracking results of the left eye pupil.

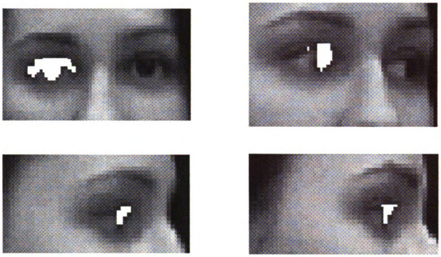


Figure 3.26 Level set tracking results of the left eye pupil (close-up)

The results of level set tracking with the initial curve set as the left eye are shown in Figure 3.25. The left eye pupil being tracked is shaded with white color. Figure 3.26 shows the close-up images of the left eye. It was seen that the level set tracking algorithm never lost track of the left eye for the entire duration of the video sequence. Tracking between each frame (384 x 512) takes an average of 0.0719 seconds on a standard PC with 2GB RAM and 1.6 GHz Pentium processor. With this computational speed, a maximum of only 14 frames can be tracked per second. Reducing the resolution of the video images does not affect tracking in most cases. Level set tracking of half resolution images (192 x 256) take an average of 0.0221 seconds. The maximum number of frames that can be tracked per second improves to 45.

To summarize this chapter, both the template based methods and level set methods track the eyes of different people successfully. In processing data from a 90 second video stream (i.e. 2250 frames) the tracking algorithm had to re-initialize the face detection on only two instances. The tracking algorithms primarily failed when the face of the subject is rotated sideways. Tracking one eye is sufficient for the computation of PERCLOS feature, as both the eyes are closed simultaneously. A critical measure of the usefulness of a tracking algorithm is its tracking speed per frame. Table 3.1 shows the average tracking speeds of different methods.

Table 3.1 Speed comparision of various tracking methods

Method	Feature	Resolution	Tracking speed per frame (in seconds)	Maximum # of frames tracked per second
Template matching	Pixel Intensity	384 x 512	0.0757	13
Template matching	Pixel Intensity	96 x 128	0.0192	52
Template matching	Skin Probability	96 x 128	0.1102	9
Level set tracking	Pixel Intensity	384 x 512	0.0719	14
Level set tracking	Pixel Intensity	192 x 256	0.0221	50

Chapter 4 PERCLOS feature computation

PERCLOS feature is a reliable measure of drowsiness [10] [22]. It measures the percentage of time in a minute when the eyes are at least 80 % closed. An eye closure classifier combined with tracking results from the Chapter 3 can provide a reliable PERCLOS feature. The block diagram of PERCLOS feature estimation is shown in Figure 4.1.

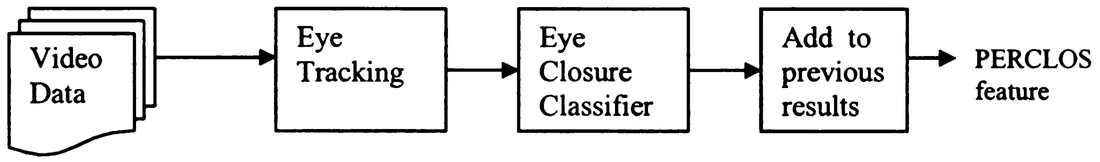


Figure 4.1 Block diagram of PERCLOS estimation

Eye tracking module tracks the eye region continuously on the incoming video data as discussed in Chapter 3. The eye closure classifier takes the output of tracking module and classifies the eye region as open or closed. The classifier results from frames obtained in the previous one minute are accumulated to estimate the PERCLOS feature.

The template based and level-set based tracking methods give different tracking outputs of the eye region. The level set tracking output follows the eye pupil region exactly. An eye closure classifier can make use of this output to differentiate between the open and closed eyes. The tracking output from the template method can be pre-processed to generate an output similar to that of level-set tracking.

4.1 Pre-Processing

The main objective of the pre-processing stage is to identify the eye pupil area from the template tracking output as shown in the Figure 4.2. Eye pupil area has dark intensity and non-skin color usually black, brown or blue. Therefore the block diagram of the pre-processing stage contains a skin removal step and an intensity threshold step as depicted in the schematic in Figure 4.3.

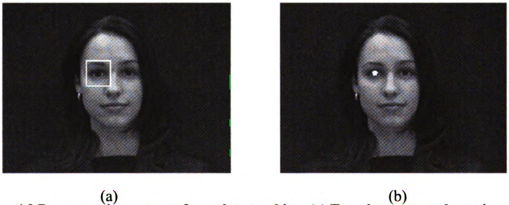


Figure 4.2 Preprocessing output of template tracking. (a) Template output shown in white rectangle (b) Eye pupil area highlighted in white

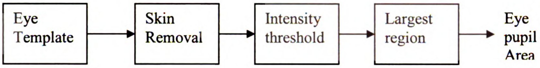
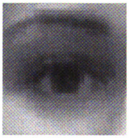
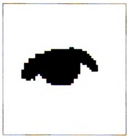


Figure 4.3 Pre-processing stages for generating the eye pupil area from eye template

For the skin removal step, any of the skin detection models from Chapter 2 can be used. Skin pixels are detected using explicit RGB skin model and are removed from the eye template as shown in Figure 4.4 (b). The output from the skin removal stage is thresholded for dark intensities. The eye pupil area constitutes around 10% of the total template window area. Therefore a threshold of 10 percentile intensity is chosen as the intensity threshold. Figure 4.4 (c) shows the output after intensity thresholding. The largest 4-neighborhood region is selected as the eye pupil area. Figure 4.4 (d) and (e) show the segmented eye pupil area by itself and when overlayed on the eye template respectively.


(a)


(b)

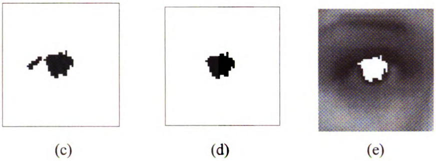
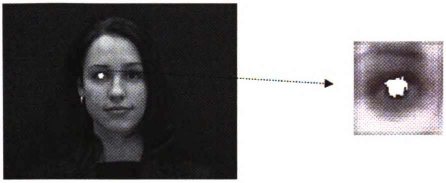


Figure 4.4 Pre-processing at various stages (a) Eye template (b) Skin removal (c) Intensity threshold (d) Eye pupil area (e) Eye pupil are overlapped on the eye template

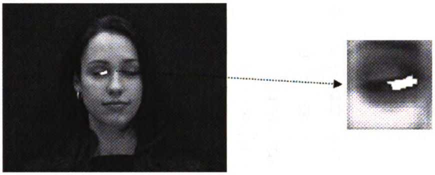
Eye pupil area which is the final output of the pre-processing stage is used as the primary feature in eye closure classification.

4.2 Eye Closure Classification

Eye closure classifier must detect whether the eyes are open or closed. Eye pupil area is a reliable feature containing information about eye closure. If the eye pupil area is short and elongated in the horizontal direction, then the eyes are probably closed. Figure 4.5 shows the eye pupil area for open and closed eyes.



(a)



(b)

Figure 4.5 Eye pupil area of open and closed eyes. (a) Eye pupil area of an open eye (b) Eye pupil area of a closed eye.

A training data set containing about 200 eye images is created manually from the VidTIMIT video dataset [24] to test the eye closure classifier. Eyes from multiple people are chosen at different stages of eye closure. Overall, 30 closed and 130 open eyes are extracted. There are also about 40 eye images in which the eyes are partially closed and a part of the eye pupil is visible. All these eye regions are pre-processed to extract the eye pupil region for template based tracking. The level set tracking already provides the eye pupil region. The height and width of the eye pupil region are computed and plotted in Figure 4.6.

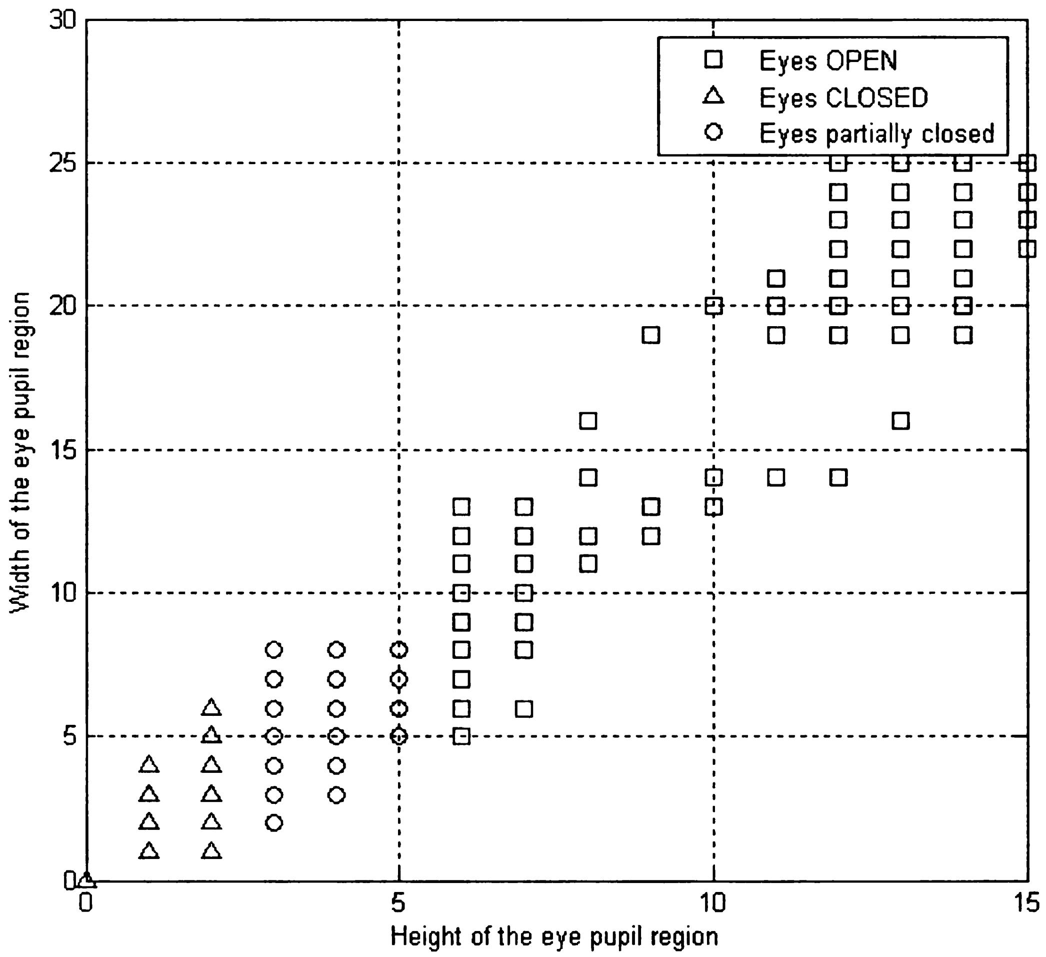


Figure 4.6 Plot of Height Vs. Width of the eye pupil region.

It is seen in Figure 4.6 that the height of eye pupil clearly discriminates well between open and closed eye classes. A threshold has to be chosen based on the training eye image data. For a video resolution of the 192 x 256 (length x width), a threshold of 3 pixels is chosen. An eye pupil area can be classified as open if its height is greater than 3 pixels.

PERCLOS feature computation is straightforward after the eye closure classification. As PERCLOS is the percentage of time in 60 seconds the eyes are 80% closed, the eye closure classification results on frames of the last 60 seconds can be stored and used for computing the PERCLOS feature as described below

$$PERCLOS = \sum_{n=1}^{60*25} EyeClosure\{frame(n)\} \qquad (4.1)$$

A PERCLOS based drowsy driver detection system uses the following rule to detect and alert drivers.

```
if $PERCLOS > \alpha$
        Drowsy Driver – Alert action
else
        Attentive Driver – No action
.
Reset PERCLOS and Continue monitoring
```

Threshold α typically ranges from 0.7 to 1 and can be determined more accurately using a large database containing ground truth of drowsiness and PERCLOS feature [45].

4.3 Results

The eye closure classifier is applied to the tracking output of multiple video sequences chosen from VidTIMIT dataset. The entire sequence of face detection, eye tracking and eye closure classification are applied on each frame automatically. Using a threshold value of 3 pixels for the height of the pupil region, the algorithm was implemented on a test dataset of 100 eye images. The results of eye closure classification on video sequences with different subjects are shown in Figure 4.7 - 4.10.

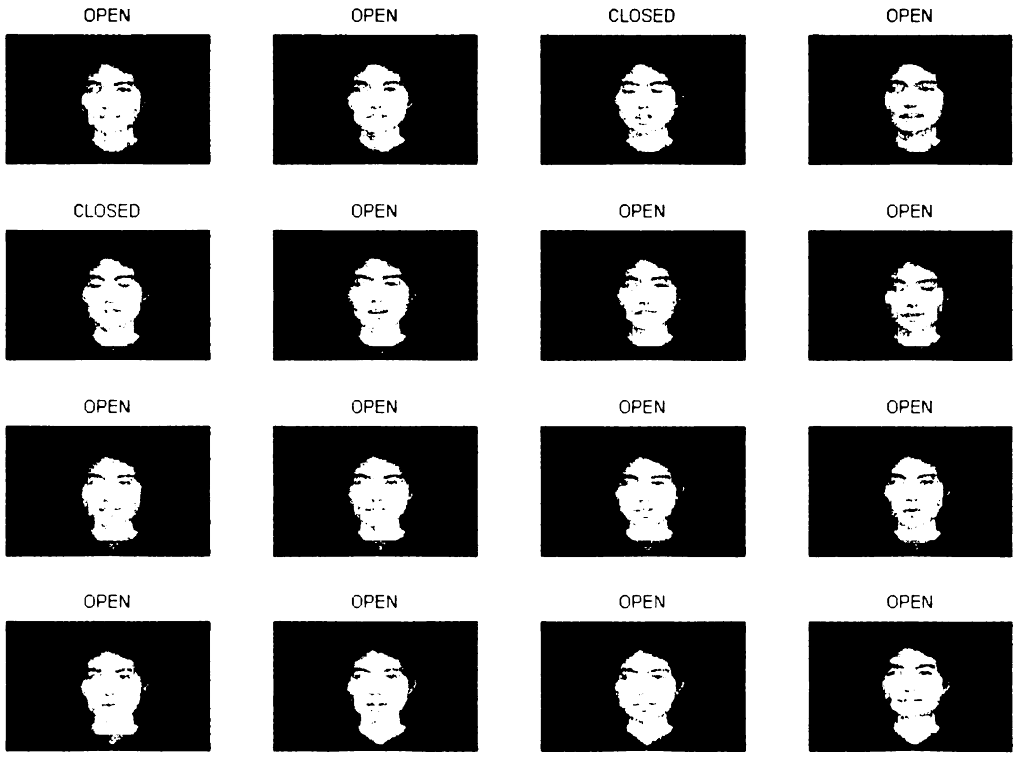


Figure 4.7 Snapshots of a video sequence with an eye class tag for each frame - 1. Only misclassification is on the frame located at 2nd row and 4th column.

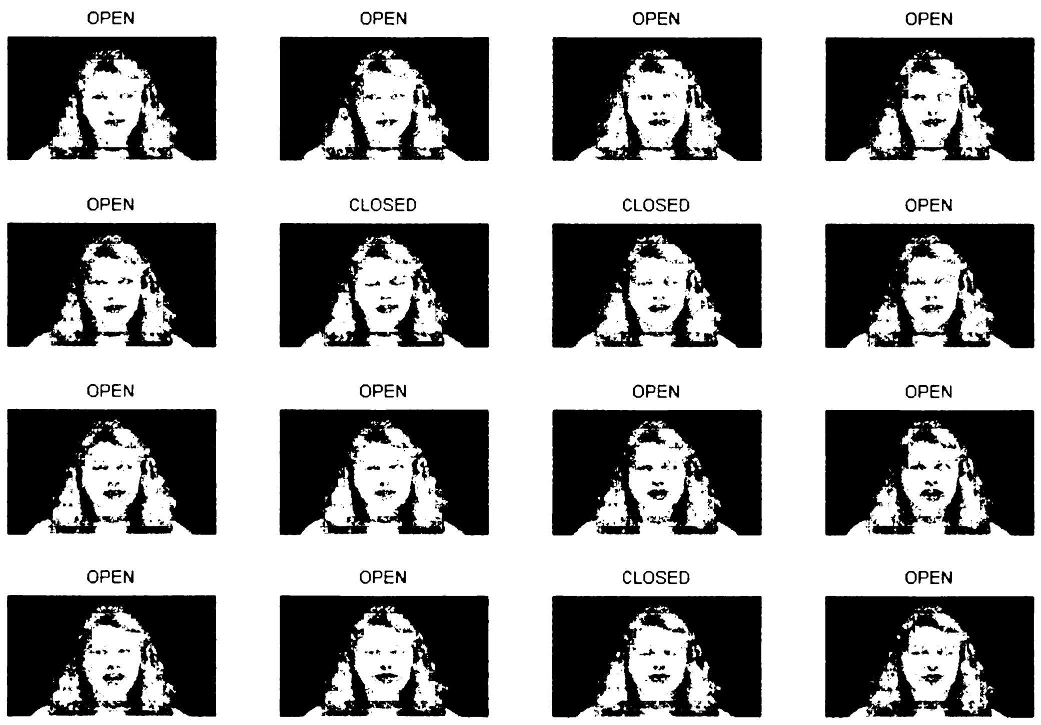


Figure 4.8 Snapshots of a video sequence with an eye class tag for each frame – 2. All the frames were correctly classified

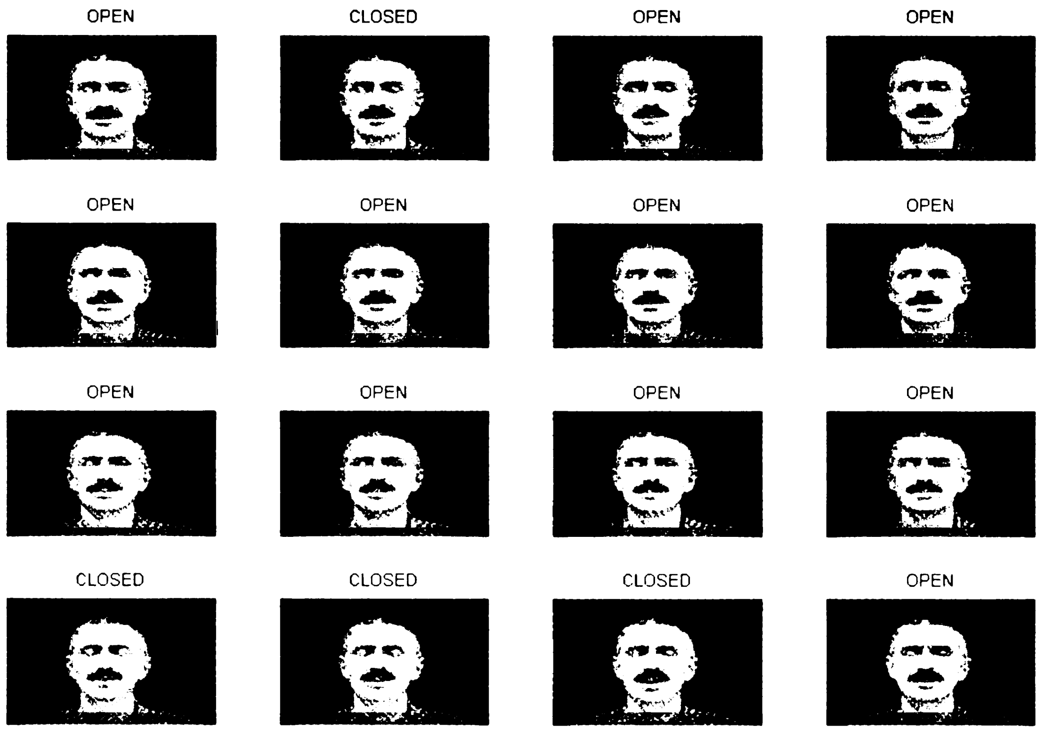


Figure 4.9 Snapshots of a video sequence with an eye class tag for each frame – 3. Only misclassification is on the frame located at 1st row and 1st column

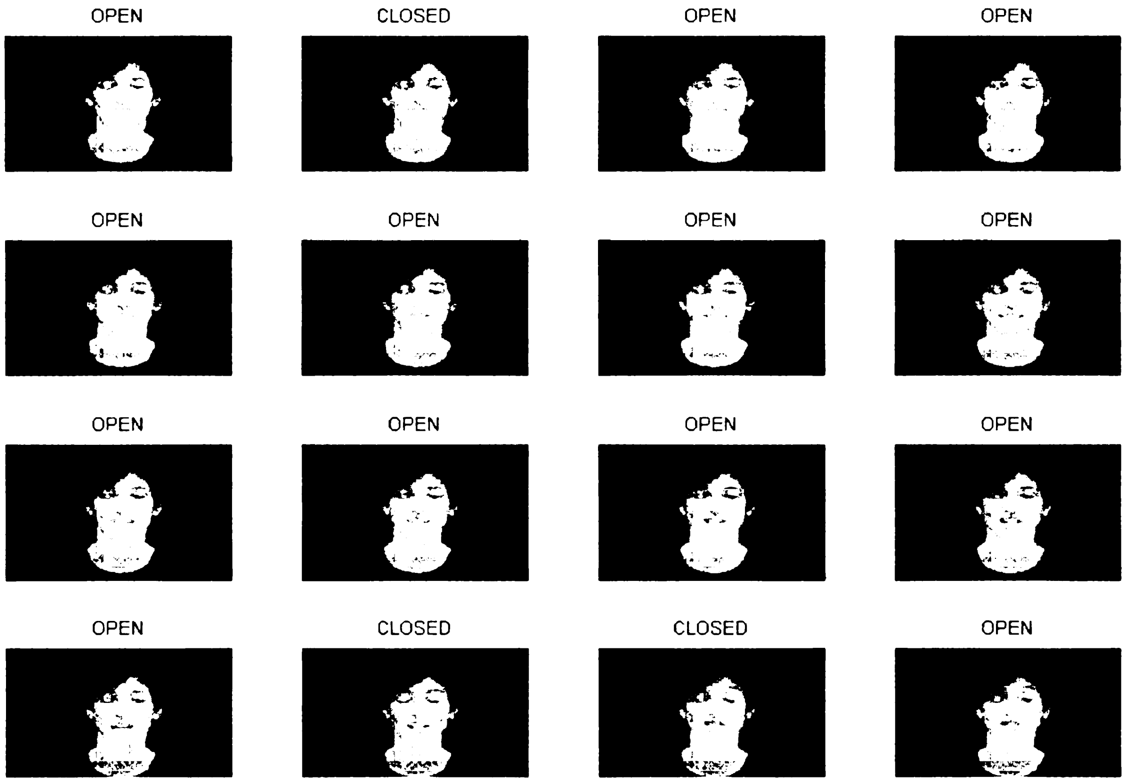


Figure 4.10 Snapshots of a video sequence with an eye class tag for each frame – 4. All the frames were correctly classified

The confusion matrix of the eye closure classifier on 64 images which are manually verified is as shown in the table 4.1

Table 4.1 Confusion matrix of the eye closure classifier on 64 test images

	Eyes Closed (Classifier)	**Eyes Open (Classifier)**
Eyes Closed (Ground Truth)	11	2
Eyes Open (Ground Truth)	0	51

On the test data, all the 11 closed eyes are classified correctly by the eye closure classifier while 51 of the 53 open eyes are correctly classified. This translates into a correct classification performance of 97%.

The performance of tracking algorithms was also evaluated in Chapter 3 and it was seen that the algorithm rarely lost track of the eyes. In processing data from a 90 second video stream (i.e. 2250 frames) the tracking algorithm had to re-initialize the face detection on two instances. Furthermore the algorithm is also computationally efficient and provides real-time tracking. The tracking algorithms primarily failed only when the face of the subject is rotated sideways. In this case, the eye pupil region is not visible. In a practical setup, the momentary loss of tracking does not affect PERCLOS estimation. Because, when the face is rotated sideways, the subject is physically active and attentive to surroundings. In other words, we can classify the eye as open, when the face is rotated sideways without affecting PERCLOS feature.

Chapter 5 Conclusion and Future Work

5.1 Conclusion

About one third of the U.S. drivers had fallen asleep while driving during the last year alone [8]. *Drowsy driving* results in 1550 fatalities and 100,000 vehicle crashes annually [3]. These crashes could be preventable with a drowsy driver detection system that predicts drowsiness and alerts the driver.

Most drowsy driver detection systems rely on PERCLOS feature that measures the amount of time the driver's eyes are closed in the last 60 seconds. The proposed system features an on-board color camera monitoring the driver's face in real-time. Image processing algorithms track the eyes of a driver and determine whether his / her eyes are open or closed. Aggregating the eye closure information over 60 seconds can provide us a reliable PERCLOS estimate, and hence a drowsy driver detection system.

This thesis proposes a PERCLOS based detection system, which involves three major components –

1. Skin Detection Modeling
2. Face / Eye Tracking
3. Eye Closure Classifier

Three skin detection models – explicit RGB model, non-parametric histogram model and parametric Gaussian models were implemented. An adaptive training methodology was proposed for non-parametric and parametric models to compensate illumination variations. It was found that the histogram based non-parametric skin model gives optimum performance in terms of face detection and is robust to illumination variations. Template-based and level-set based algorithms based on intensity and skin-probability features were implemented. A new feature which is a combination of both the intensity and skin probability feature was proposed. Both the tracking algorithms provided comparable results on reduced resolution data. Real-time efficiency on a standard computer is achieved on quarter resolution (96 pixels x 128 pixels) image data. A new eye closure classifier based on eye pupil shape was tested for determination of eye-closure and was seen to perform consistently on multiple video data. A combination of face and eye tracking algorithms and eye closure classifier is shown to give a reliable PERCLOS estimate on a large face video database (VidTIMIT [24]). The above algorithms provide a prototype for drowsy driver detection system using PERCLOS with an on-board video camera.

5.2 Future Work

The detection system must be tested on a more exhaustive data set. Extensive testing must be done by systematically varying illumination, pixel noise, driver ethnicity etc. The real-world performance of the detection system could be validated on drivers operating driving simulators in a drowsy state. In the future it would be of great research value to create a benchmark video database for drowsiness testing. This video database could be used to compare the performance of various drowsy detection systems developed by different researchers with respect to both time and accuracy. Performance accuracy should be quantified in terms of probability of detection drowsiness as well as probability of false alarm.

The tracking results could be further improved by the fusion of tracking output from template and level set methods. Though data fusion is computationally intensive, a hardware implementation of the system using a digital signal processor (DSP) board could achieve real-time performance. A complete prototype system using a camera and DSP module should be assembled and tested.

Appendix A – Connected components algorithm

Connected components algorithm [41] finds and labels the objects in a binary image. Suppose that **B** is a binary image and that **B[r,c]** = **B[r',c']** = *v* where either *v* = 0 or *v* =1. The pixel [r,c] is *connected* to the pixel **[r',c']** with respect to the value *v* if there is a sequence of pixels **[r, c]** = **[r_0,c_0]**, **[r_1,c_1]**, ..., **[r_n,c_n]** in which **B[r_i,c_i]** = v, *i* = 0, ..., **n** and **[r_i,c_i]** neighbors **[r_{i-1}, c_{i-1}]** for each *i* = 1, ..., **n**. The sequence of pixels **[r_0,c_0]**, ..., **[r_n,c_n]** forms a connected *path* from **[r,c]** to **[r',c']**. A *connected component* of value *v* is a set of pixels *C*, each having value *v*, and such that every pair of pixels in the set are connected with respect to *v*. Figure A.1 (a) shows a binary image with five such connected components of 1s; these components are actually connected with respect to either the 8-neighborhood or the 4-neighborhood definition.

1	1	0	1	1	1	0	1
1	1	0	1	0	1	0	1
1	1	1	1	0	0	0	1
0	0	0	0	0	0	0	1
1	1	1	1	0	1	0	1
0	0	0	1	0	1	0	1
1	1	0	1	0	0	0	1
1	1	0	1	0	1	1	1

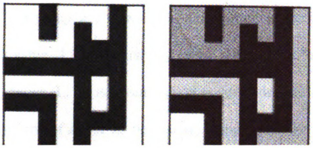
(a)

1	1	0	1	1	1	0	2
1	1	0	1	0	1	0	2
1	1	1	1	0	0	0	2
0	0	0	0	0	0	0	2
3	3	3	3	0	4	0	2
0	0	0	3	0	4	0	2
5	5	0	3	0	0	0	2
5	5	0	3	0	2	2	2

(b)

Figure A.1

Figure A.1 (continued)



(c) (d)

Figure A.1 A binary image with five connected components of the values 1. (a) Binary image matrix (b) Connected components labeling (c) Binary image (d) Labeled image [41]

Definition. A connected components labeling of a binary image **B**, is labeled image **LB** which the value of each pixel is the label of its connected component.

A label is a symbol that uniquely names an entity. While character labels are possible, the positive integers are more convenient and are most often used to label the connected components. Figure A.1 (b) shows the connected components labeling of the binary image of Figure A.1 (a).

Recursive Labeling Algorithm: Suppose that **B** is a binary image with **MaxRow** + 1 rows and **MaxCol** + 1 columns. We wish to find the connected components of 1-pixels and produce a labeled output image **LB** in which every pixel is assigned the label of its connected component. The strategy is to first negate the binary image, so that all the 1-pixels become -1s. This is needed to distinguish unprocessed pixels (-1) from those of component label 1. We will accomplish this with a function called *negate* that inputs the binary image **B** and outputs the negated image **LB**, which will become the labeled image.

Then the process of finding the connected components becomes of finding a pixel whose value is -1 in **LB**, assigning it a new label, and calling procedure *search* to find its neighbors that have value -1 and recursively repeat the process for these neighbor. The utility function *neighbors* **(L, P)** is given a pixel position defined by **L** and **P**. It returns the set of pixel positions of all of its neighbors, using either the 4-neighborhood or 8-neighborhood definition. Only neighbors that represent legal positions on the binary image are returned.

Recursive algorithm pseudo code:

B is the original binary image.

LB will be the labeled connected component image.

```
procedure recursive_connected_components(B, LB);
{
LB := negate(B);
label := 0;
find_components(LB, label);
print(LB);
}

procedure find_components(LB, label);
```

```
{
for L := 0 to MaxRow
        for P := 0 to MaxCol
                if LB[L,P] == -1 then
                 {
                   label := label + 1;
                   search(LB, label, L, P);
                 }
}

procedure search(LB, label, L, P);
{
LB[L,P] := label;
Nset := neighbors(L, P);
for each [L', P'] in Nset
  {
        if LB[L', P'] == -1
        then search(LB, label, L', P');
  }
}
```

Appendix B – Fast Level Set Method

Fast level set method [46] [34] uses a region-based tracking model based on the region competition idea, but this implementation is also applicable for edge-based models.

We assume each scene of the video sequence is composed of a background region 0 and M object regions $\Omega_1,\Omega_2,\ldots, \Omega_M$. The boundaries of these M object regions are denoted as $C_1, C_2, \ldots, C_M$. We model each region with a feature distribution $p(\underline{v}|\Omega_m)$(m = 0,1, … , M), where $\underline{v}$ is the feature vector defined at each pixel. For example, the features can be the color, the output of a filter bank designed to model textures, or other visual cues. Assuming that the feature distribution at each pixel is independent, the tracking result of each frame is the minimum of the following region competition energy:

$$E = -\sum_{m=0}^{M} \int_{\Omega_m} \log p(\underline{v}(\underline{x}) \mid \Omega_m) d\underline{x} + \lambda \sum_{m=1}^{M} \int_{C_m} ds \tag{B.1}$$

where E_d is the data _fidelity term that represents the likelihood of the current scene, E_s is for smoothness regularization and is proportional to the length of all curves, and λ is the non-negative regularization parameter.

By computing the first variation of this energy, the curve evolution equation for the minimization of this energy is:

$$\frac{dC_m}{dt} = (Fd + Fs)\vec{N}_{C_m} \quad (m = 1,2,\ldots,M) \tag{B.2}$$

where $\vec{N}_{C_m}$ is the normal of C_m pointing outward, and F_d and F_s are the speed resulting from E_d and E_s, respectively. The speed F_d represents the competition between two regions and it is $F_d = \log[p(\underline{v}(\underline{x}) | \Omega_m) / p(\underline{v}(\underline{x}) | \Omega_{out})]$, where Ω_{out} denotes the region outside C_m at $\underline{x} \in C_m$. The speed F_s makes the curve smooth and it is $F_s = \lambda_k$, where κ is the curvature.

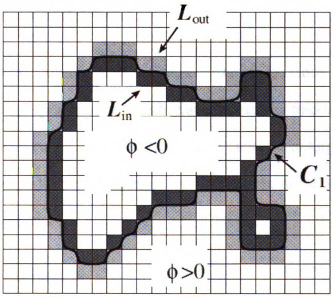


Figure B.1 Implicit representation of the curve C_1 and the two lists L_{in} and L_{out} in the neighborhood of C_1 [46]

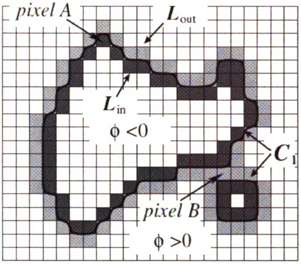


Figure B.2 Illustration of the motion of the curve C_1 by switching pixels between L_{in} and L_{out}. [46]

Since the current is on real-time level set implementation for video tracking, we use a simple tracking strategy as follows. For each frame, we use the tracking results from the last frame as the initial curves, and then evolve each curve according to (B.2) to locate the object boundaries in the current frame. Once it stops, we move on to track the next frame of the video sequence.

Fast Level Set Implementation

In this section, we present a fast level set implementation of the curve evolution process in (B.2) when the scene is composed of only the background Ω_0 and a single object region Ω_1. Extensions are then made to track multiple objects with different feature distributions in the next section.

To represent the background Ω_0 and the object region Ω_1, we use a level set function ϕ which is negative in Ω_1 and positive in Ω_0. Based on this representation, we define two lists of neighboring pixels L_{in} and L_{out} of C1 as shown in Fig B.1. Formally they are defined as:

$$\begin{aligned} L_{out} &= \{\underline{x} \mid \phi(\underline{x}) > 0 \textit{ and } \exists\ \underline{y} \in N_4(\underline{x}) \text{ such that } \phi(\underline{y}) < 0\}, \\ L_{in} &= \{\underline{x} \mid \phi(\underline{x}) < 0 \textit{ and } \exists\ \underline{y} \in N_4(\underline{x}) \text{ such that } \phi(\underline{y}) > 0\} \end{aligned} \tag{B.3}$$

where $N_4(\underline{x})$ is a 4-connected discrete neighborhood of a pixel $\underline{x}$ with $\underline{x}$ itself removed.

In conventional implementations of the level set method, an evolution PDE is solved either globally on the whole domain or locally in a narrow band to evolve the curve according to (B.2). Our fast level set implementation is based on the key observation that the implicitly represented curve C_1 can be evolved at pixel resolution by simply switching the neighboring pixels between the two lists L_{in} and L_{out}. For example, as we show in Figure B.2, the curve C_1 moves outward at pixel A and shrinks and splits at pixel B compared with the curve shown in Fig.1. This motion can be realized by simply switching pixel A from L_{out} to L_{in}, and pixel B from L_{in} to L_{out}. By doing this for all the pixels in L_{in} and L_{out}, the curve can be moved inward or outward for one pixel everywhere in one scan. Since the curve is still represented implicitly, topological changes can be handled automatically. With this idea, we can achieve level-set based curve evolution at pixel resolution and this is usually enough for many imaging applications. Next we present the details of the fast algorithm.

Basic Algorithm

The data structure used in our implementation is as follows:_

- An array for the level set function ϕ;
- An array for the evolution speed F;
- Two bi-directionally linked lists of neighboring pixels: L_{in} and L_{out}.

Besides the inside and outside neighboring pixels contained in Lin and Lout, we call those pixels inside C_1 but not in L_{in} as *interior pixels* and those pixels outside C_1 but not in L_{out} as *exterior pixels*. For faster computation, we define as follows:

$$\phi(x) = \begin{cases} 3, & \text{if x is outside C, and } x \notin L_{out}; \\ 1, & \text{if } x \in L_{out}; \\ -3, & \text{if x is outside C, and } x \notin L_{in}; \\ -1, & \text{if } x \in L_{in}; \end{cases} \tag{B.4}$$

To switch pixels between L_{in} and L_{out}, we define two basic procedures on our data structure. The procedure *switch_in*() for a pixel $\underline{x} \in L_{out}$ moves the curve outward one pixel at x by switching it from L_{out} to L_{in} and adding all its neighboring exterior pixels to Lout. Formally this procedure is defined as follows:

switch_in($\underline{x}$):

- Step 1: Delete x from L_{out} and add it to Lin. Set $\phi(\underline{x}) = -1$.
- Step 2: $\forall\ \underline{y} \in N_4(\underline{x})$ satisfying $\phi(\underline{y}) = 3$, add $\underline{y}$ to L_{out}, and set $\phi(\underline{y}) = 1$

Similarly, the *switch_out*() procedure that moves the curve inward one pixel at $\underline{x} \in L_{in}$ is defined as follows:

*switch*_out(x):

- Step 1: Delete $\underline{x}$ from L_{in} and add it to L_{out}. Set $\phi(\underline{x}) = 1$.
- Step 2: $\forall\ \underline{y} \in N_4(\underline{x})$ satisfying $\phi(\underline{y}) = -3$, add $\underline{y}$ to L_{in}, and set $\phi(\underline{y}) = -1$

To track the object boundary, we compute the speed at all pixels in L_{in} and L_{out}, and store their sign in an array F. We scan through the list L_{out}, and apply a *switch_in*() procedure at a pixel if $F = +1$. After this scan, some of the pixels in Lin become interior pixels and they are deleted. We then scan through the list Lin and apply a *switch_out*() procedure for a pixel with $F = -1$. Similarly, exterior pixels in Lout are deleted after this scan. At the end of this iteration, a stopping condition is checked. If it is satisfied, we stop the evolution; otherwise, we continue this iterative process. In out implementation, the following stopping condition is used:

Stopping Condition. The *curve evolution algorithm* stops if either of the following condition is satisfied:

(a) The speed at each neighboring pixel satisfies:

$$\begin{aligned} F(\underline{x}) &\le 0 \ \forall \underline{x} \in L_{out} \\ F(\underline{x}) &\ge 0 \ \forall \underline{x} \in L_{in} \end{aligned} \tag{B.5}$$

(b) A pre-specified maximum number of iterations is reached.

The condition in equation (B.5) is very intuitive in the sense of region competition. When the curve is on the object background, all the pixels in L_{out} are in background and all the pixels in L_{in} are in the object region. When equation (B.5) is satisfied, they disagree with each other on which direction to move the curve and convergence is reached. When the data is noisy or there is clutter, regularization is necessary and (B.5) may not always be satisfied in the final curve. Thus part (b) of the condition is also necessary to stop the evolution.

The above algorithm can be applied to arbitrary speed fields and speeds up the evolution process in (B.2) dramatically compared with previous narrow band techniques based on solving the level set PDE. For the curve evolution equation in (B.2), we can achieve a further speedup by introducing a novel scheme that separates the evolution driven by the data dependent speed F_d and the smoothing speed F_s into two different cycles. In spirit, this idea is similar to the work in [10] which proposed a fast method to implement the Chan-Vese model [5] over the whole domain, but the two-cycle algorithm we present next is still based on updating the two linked lists L_{in} and L_{out} to evolve the implicitly represented curve.

Further details of using internal variations by 2D filtering and tracking multiple objects can be found in papers [32] and [46].

Bibilography

1. Storie, V.J., *Involvement of good vehicles in public service vehicles in motorway accidents.* 1113, UK Department of transport, Transport and road research laboratory, 1984.

2. O'Hanlon, J.F., *That is the extent of driving fatigue problem?* EUR6065EN; Commission of the European communities, 1978.

3. Rau, P.S., *NHTSA's drowsy driver research program fact sheet.* Washington, D.C.: National Highway Traffic Safety Administration (1996).

4. Wang, J.S., Knipling, R.R., and Blincco, L.J., *Motor vehicle crach involvements: a multi-dimensional problem size assessment.* Sixth annual meeting of the American Intelligent Traffic System, Houston, TX April 14-18, 1996, Washington D.C.: National Highway.

5. *National Survey of Distracted and Drowsy Driving Attitudes and Behaviors: 2002*

6. Kribbs, N.B., Dinges, D.F.,*Vigilance decrement and sleepiness.* Sleep onset mechanisms. Washington, DC: American ..., 1994

7. Hartley, L., *Fatigue and Driving: Driver Impairment, Driver Fatigue and Driving Simulation. The role of fatigue research in setting driving hours regulations.* ISBN: 0748402624. Publisher: Taylor & Francis, Inc. Pub. Date: January 1995, pp. 41-47

8. Dinges, D.F., *An Overview of Sleepiness and Accidents. Journal of Sleep Research* 1995; 4(2):4-14

9. Stern, J., *Eye Activity Measures of Fatigue and Napping as a Counter-measure.* USDOT Technical Report. FHWA-MC-99-028 Jan. 1999; 26

10. Federal Highway Administration, Office of Motor carriers, PERCLOS, *A Valid Psychophysiological Measure of Alertness as Assessed by Psychometer Vigilance.*

11. Wunsch, P., Hirzinger, G., *Real-time visual tracking of 3D objects with dynamic handling of occlusion*, Robotics and Automation, 1997. Proceedings.

12. Liang, R.H., Ouhyoung, M., *A real-time continuous gesture recognition system for sign language*, Face and Gesture Recognition, 1998 - doi.ieeecs.org,

13. Vezhnevets, V., Sazonov, V., Andreeva, A., *A Survey on Pixel-based Skin Color Detection Techniques*, Faculty of computational mathematics and Cybernetics, Moscow State University, Moscow, Russia.

14. King, D.C., Michels, K.M., *Muscular tension and the human blink rate,* J Exp Psychol (1957) 53,113-116

15. Stoerring, M., *Computer Vision and Human Skin Colour* Ph.D. Dissertation, Computer Vision & Media Technology Laboratory, Aalborg University, Denmark

16. Stoerring, M., Andersen, H.J., Granum, E., *Estimation of the Illuminant Colour from Human Skin Colour*, Computer Vision and Media Technology Laboratory, Aalborg University, Niels Jernes Vej 14, DK-9220 Aalborg, Denmark

17. Yang, J., Lu, W., and Waibel, A., *Skin-color modeling and adaptation.* In Chin, R. and Pong, T.C., editors, *3rd Asian Conf. on Computer Vision*, volume 1352 of *LNCS*, pages 687{694, Hong Kong, China, Jan. 1998.

18. Schwerdt, K., and Crowley. J.L., *Robust face tracking using color,* In International Conference on Automatic Face & Gesture Recognition [1], pages 90.95.

19. Stoerring, M., *Computer Vision and Human Skin Colour* Ph.D. Dissertation, Computer Vision & Media Technology Laboratory, Aalborg University, Denmark

20. U.S. Department of transportation, *Traffic Safety Facts – 1999,* National Highway Traffic Safety Administration. http://www.nhtsa.dot.gov/people/ncsa/809-100.pdf

21. http://ai.volpe.dot.gov/CarrierResearchResults/HTML/2003Crashfacts/2003LargeTruckCrashFacts.htm

22. Wierwille, W.W., Ellsworth, L.A., Wreggit, S.S., Fairbanks, R.J., Kim C.L., *Research on Vehicle-Based Driver Status/Performance Monitoring: Development, Validation and Refinement of Algorithms for Detection of Driver Drowsiness*, National Highway Traffic Safety Administration Final Report: DOT HS 808 247, 1994. (October 1998. Publication No. FHWA-MCRT-98-006)

23. Peer, P., Kovac, J., Solina, F., *Human skin colour clustering for face detection.* In submitted to EUROCON 2003 – International Conference on Computer as a Tool.

24. http://users.rsise.anu.edu.au/~conrad/vidtimit/zips/vidtimit_documentation.pdf

25. Sigal, L., Sclaroff, S., Athitsos, V., *Estimation and prediction of evolving color distributions for skin segmentation under varying illumination.* In Proc. IEEE Conf. on Computer Vision and Pattern Recognition *(2000)*, vol. 2, 152–159.

26. Soriano, M., Huovinen, S., Martinkauppi, B., Laaksonen, M., *Skin detection in video under changing illumination conditions*. In Proc. 15th International Conference on Pattern Recognition (2000), vol. 1, 839–842.

27. Yang, M., Ahuja, N. *Gaussian mixture model for human skin color and its application in image and video databases.* In Proc. of the SPIE: Conf. on Storage and Retrieval for Image and Video Databases (SPIE 99), vol. 3656, 458–466.

28. Hakkanen, H., Summala, H., Partinen, M., Tiihonen, M., Silvo, J. *Blink duration as an indicator of driver sleepiness in professional bus drivers*. Sleep (1999), 22(6), 798–802.

29. Gomez, G., Morales, E., *Automatic feature construction and a simple rule induction algorithm for skin detection.* In Proc. of the ICML Workshop on Machine Learning in Computer Vision (2002), 31–38.

30. Stern, H., Efros, B., *Adaptive color space switching for face tracking in multi-colored lighting environments.* In Proc. of the International Conference on Automatic Face and Gesture Recognition (2002), 249–255.

31. Osher, S., Fedkiw, R., Level Set Methods and Dynamic Implicit Surfaces, Springer publications.

32. Peng, D., Merriman, B., Osher, S., Zhao, H., Kang, M., *A pde-based fast local level set method*, Journal of Computational Physics, vol. 155, pp. 410–438, 1999.

33. Sethian, J., *A fast marching level set method for monotonically advancing fronts*, Proc. Nat. Acad. Sci, vol. 93, no. 4, pp. 1591–1595, 1996.

34. Shi, Y., Karl, W., *A fast level set method without solving PDEs*, Proc. ICASSP'05

35. CIE 1931 color space chromaticity diagram - www.en.wikipedia.org/wiki/CIE_1931_color_space

36. Grace, R., Byrne, V.E., Bierman, D.M., Legrand, J.M., Gricourt, D., Davis, B.K., Staszewski, J.J., Carnahan, B., *A drowsy driver detection system for heavy vehicles*, Digital Avionics Systems Conference, 1998. Proceedings. 17th DASC. The AIAA/IEEE/SAE.

37. Mallis, M.M., *Evaluation of In-flight Alertness Management Technology*, NASA report – 2002

38. Grace, R., *Drowsy Driver Monitor and Warning System*, Robotics Institute, Carnegie Mellon University.

39. Shafer, S., *Using color to separate reflection components*, Technical report for DARPA – 1984, Computer Science Department, University of Rochester, NY.

40. Q Ji, G Bebis, Visual cues extraction for monitoring driver vigilance – Procs. Honda Symposium, 1999.

41. Stockman, G., Shapiro, L., *Computer Vision*, Prentice-Hall Inc, 2001.

42. http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT14/

43. http://en.wikipedia.org/wiki/Level_set_method

44. Duda, R.O., Hart, P.E., *Pattern Classification (2nd ed.)* by and David G. Stork Wiley Interscience.

45. Knipling, R., *PERCLOS: A Valid Psychophysiological Measure of Alertness As Assessed by Psychomotor Vigilance*, Federal Highway Administration, Office of Motor Carriers.

46. Shi, Y., Karl, W., *Real-time tracking using level sets*, CVPR'05, Jun, 2005