國 立 交 通 大 學

電信工程學系

碩 士 論 文

使用膚色比例前處理之
即時性人臉偵測系統

Real-time Face Detection System
with Skin Ratio Preprocessing

研 究 生：張榮勝

指導教授：張文鐘 博士

中 華 民 國 九 十 六 年 八 月

使用膚色比例前處理之

即時性人臉偵測系統

Real-time Face Detection System

with Skin Ratio Preprocessing

研　究　生：張榮勝	Student: Jung-Sheng Chang
指導教授：張文鐘　博士	Advisor: Dr. Wen-Thong Chang

國　立　交　通　大　學

電信工程學系

碩　士　論　文

A Thesis
Submitted to Department of Communication Engineering
College of Electrical and Computer Engineering
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master
In Communication Engineering

中　華　民　國　九十六　年　八　月

使用膚色比例前處理之
即時性人臉偵測系統

研究生:張榮勝　　　　　　　指導教授:張文鐘　博士

國立交通大學電信工程學系碩士班

摘要

在此篇論文中，我們建構出即時性人臉偵測系統，此系統能偵測出-45 度至 45 度的人臉。我們的人臉偵測包含兩部份：第一部份尋找出整張影像中的膚色區域，以獲得可能為臉的區域。由於偵測區域只在膚色區域上，故處理時間可節省 20~95%；第二部份為人臉偵訊程序，我們使用積分影像(integral image)快速算出矩形特徵(rectangle feature)，且使用 AdaBoost 演算法選出重要的特徵。我們的系統建構三層 cascade 系統。第一層為 0 度人臉偵測，第二層為 45 度人臉偵測，最後一層為-45 度人臉偵測。我們也比較與分析以下性能: (1) 使用與非使用 AdaBoost 的特徵選取，(2) 兩個不同的系統設計， (3)我們系統使用與非使用膚色前處理的不同階段。我們系統的偵測率能達到 88.31%。

在即時性人臉偵側方面，根據不同大小的膚色面積，我們需要 20~400ms 去處理一張 320*240 的影像。因此，我們的系統能廣泛地使用於不同的應用程式上。

Real-time Face Detection System with Skin Ratio Preprocessing

Student: Jung-Sheng Chang Advisor: Dr. Wen-Thong Chang

The Department of Communication Engineering

National Chiao Tung University

Abstract

In the thesis, we construct real-time face detection system which can detect -45~45 degree face. Our face detection system consists of two parts. The first part searches skin color regions over the whole image to segment potential face regions. Detection regions only focus on skin regions so it can save 20~95% processing time. The second part is face detection procedure. We use integral image to compute rectangle features rapidly and AdaBoost algorithm to select important features. Our system constructs three-layer cascade structure. First layer is 0-degree face detection, second layer is 45-degree face detection, and final layer is -45-degree face detection. We also compare and analyze the performance: (1) feature selection with and without AdaBoost, (2) two different systems, and (3) different layers in our system with and without skin ratio preprocessing. The detection rate of our system can achieves 88.31%.

In real-time face detection, it requires 20~400ms to process a 320*240 image depending on skin area size. Therefore, our system can be widely used in different application programs.

Acknowledgements

I appreciate my advisor Dr. Wen-Thong Chang for helping me complete this thesis. I would also like to thank my parents, my classmates, and my friends. They supported and encouraged me a lot during my life and studies.

Contents

List of Figures

List of Table

CHAPTER 1
INTRODUCTION

1.1 Motivation

In recent ten years, the face detection issue received more attention in computer vision field. It has many applications in human daily life. For example, face detection technique is recently used on digital cameras [1]. When the face is detected in a scene, the camera automatically optimizes focus and exposure so your photos come out great. Besides, it also applies on the safety of the building. If the human face is detected, the image is transferred to the human face recognition module. The recognition result shows a person corresponding to the image is a legal or illegal member.

There are a lot of approaches which have been proposed in face detection field. They mainly focused on frontal face detection issue. But in real-time detection, the human face rotation is unavoidable. In order to solve this problem, we construct real-time face detection which can detect -45^{o}~45^{o} face. Besides, we add skin color preprocessing to improve detection speed. In a 320*240 pixel image, it requires 20~400ms to scan the whole image. Therefore, our system can be widely used in different application programs.

1.2 Related Works

The direct method of face detection is to detect skin color regions. Hsu et al. [2] and Kovac et al. [3] perform skin detection in YCbCr color space. Hsu et al. [2] find face candidates based on skin patches firstly, and construct eye, mouth, and boundary maps for verifying each face candidate. They provide high detection rate over a wide variety of facial variations but detection speed is slow.

In 2001, Viola and Jones [4] introduced robust method which can process rapidly and achieve high detection rate. They provided a new image representation called integral image which computes the feature value quickly. The training algorithm is based on two-class AdaBoost which selects a small number of critical features from a large feature set and constructs efficient classifiers. Finally, they used cascade structure which consists of some classifiers to reject a large number of non-face images. There have been some related works such as multi-pose face detection and facial expression recognition [5] [6].

1.3 Thesis Overview

Our face detection system consists of two main parts. Firstly, we segment skin regions over the entire image based on YCbCr color space. Secondly, we use different-size sub-windows to scan the image. The face detection algorithm is executed according to two conditions. One is detection sub-windows must locate at skin pixels, and the other one is the sum of skin pixels within the rectangle is larger than threshold. These two conditions can reduce scan region to improve computational time and decrease error. The flowchart of our system is shown as Fig. 1.1.

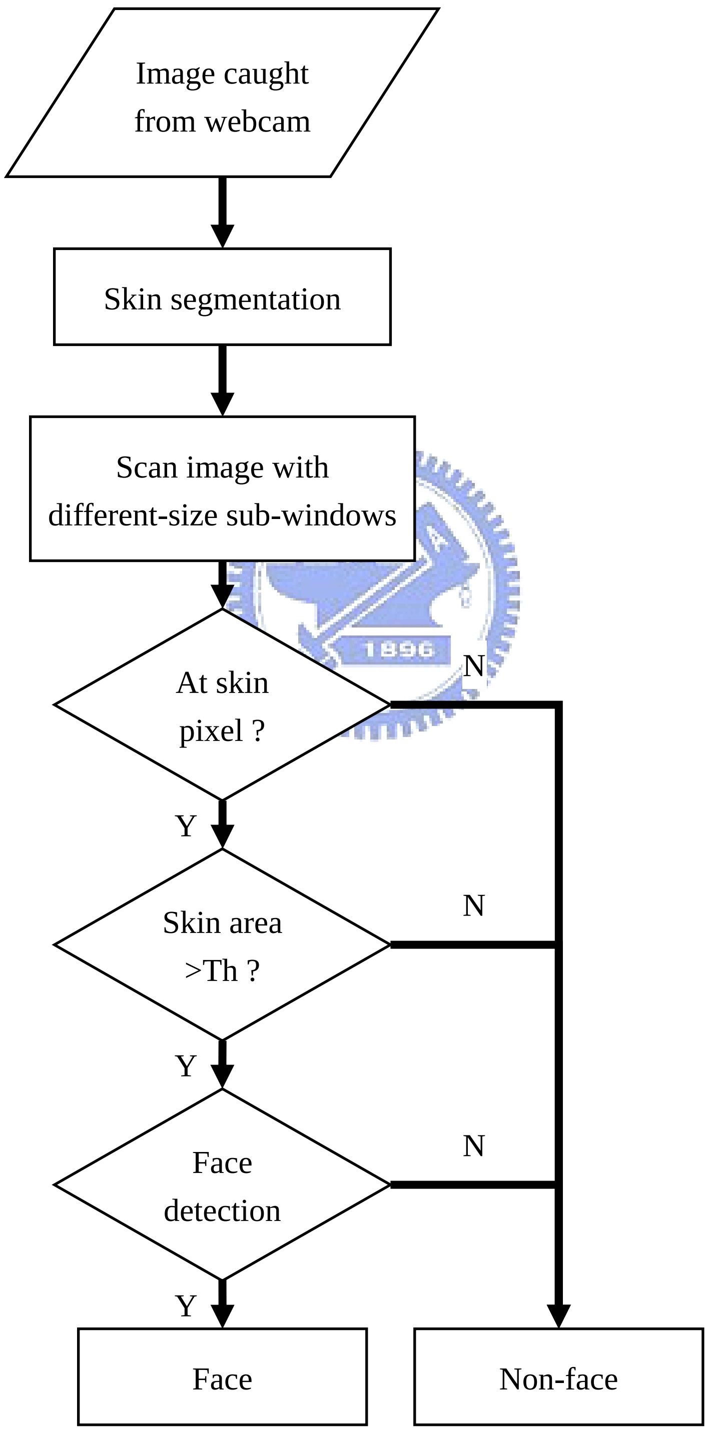


Fig. 1.1 The flowchart of our real-time face detection system

1.4 Thesis Outline

The organization of this thesis is as follows. Chapter 2 introduces feature selection with AdaBoost algorithm and cascade classifiers. Chapter 3 introduces skin segmentation and preprocessing of face detection. Chapter 4 shows experimental results, implementation of real-time system, and monitoring system with face detection. At last, the thesis is concluded in Chapter 5.

CHAPTER 2
FEATURE SELECTION AND FACE DETECTION BY USING ADABOOST

In this chapter, we will introduce two main parts. The first part of this thesis is the training process that allows computer system to learn human face classification. Viola and Jones introduced a method for constructing a classifier by selecting a small set of critical features using AdaBoost and combining more complex classifiers in a cascade structure [4]. More and more face detection techniques use this method because the feature-based system operates much faster than the pixel-based system. This method is also used in our experiment to achieve real-time object. The training process flowchart is shown in Fig. 2.1.

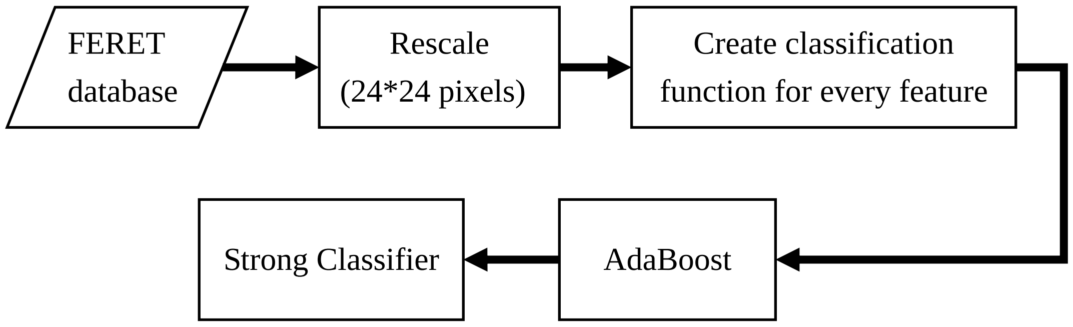


Fig. 2.1 The training process flowchart

The second part of this thesis is test process to detect face with features selected by AdaBoost. We scan the image with different-sized detection sub-windows. When some critical features are matched in a detection sub-window, the face is detected. Our system can detect face with three different pose (-45^{o}, 0^{o}, 45^{o}), as shown in Fig. 2.2. Firstly, all sub-window images enter 0^{o} face detection system, and the rejected images are regarded as non-0^{o} faces, including 45^{o}& -45^{o} faces and non-faces. Secondly, the rejected images from 0^{o} face detection enter 45^{o} face detection system, and the rejected images at this layer are regarded as non-0^{o}&45^{o} faces, including -45^{o} faces and non-faces. Finally, the remaining images enter -45^{o} face detection system, and the rejected images at this layer are regarded as non-faces.

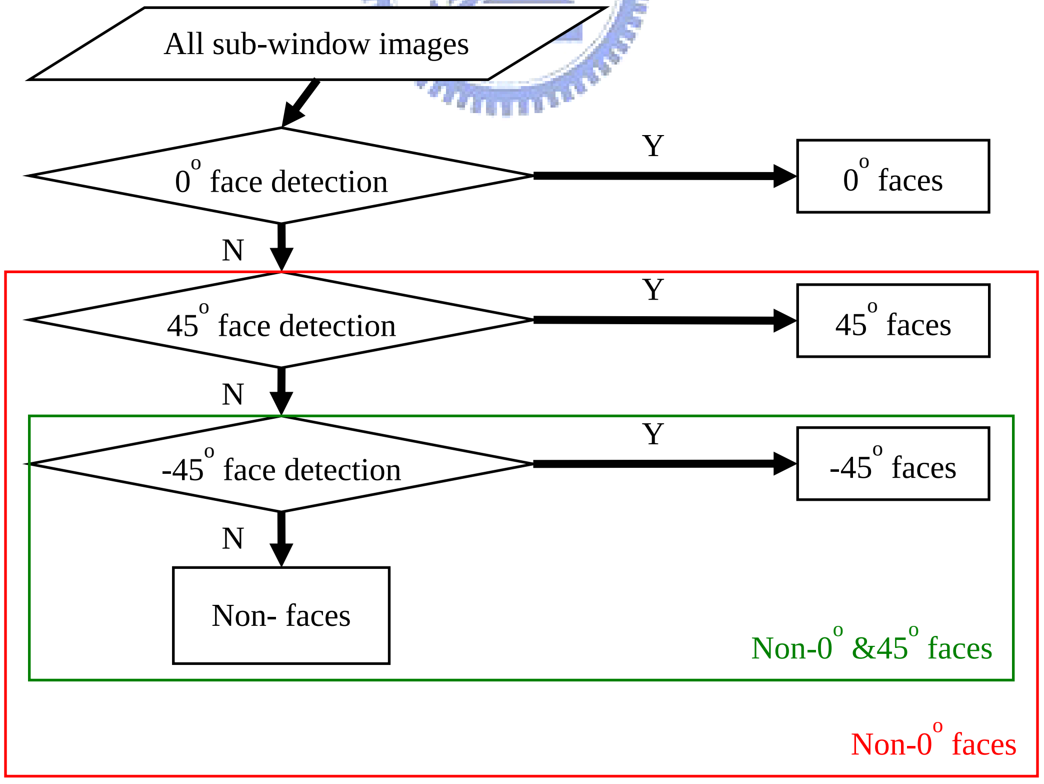


Fig. 2.2 The flowchart of our different-posed face detection

2.1 Training Samples

The training samples of three different poses (-45^{o}, 0^{o}, 45^{o}) face come from the FERET image database [7]. The training samples of non-face are created by our image database. To eliminate the influence of location, all training samples are normalized to 24*24 pixels. Fig. 2.3 and Fig. 2.4 show some examples of the normalized training samples.

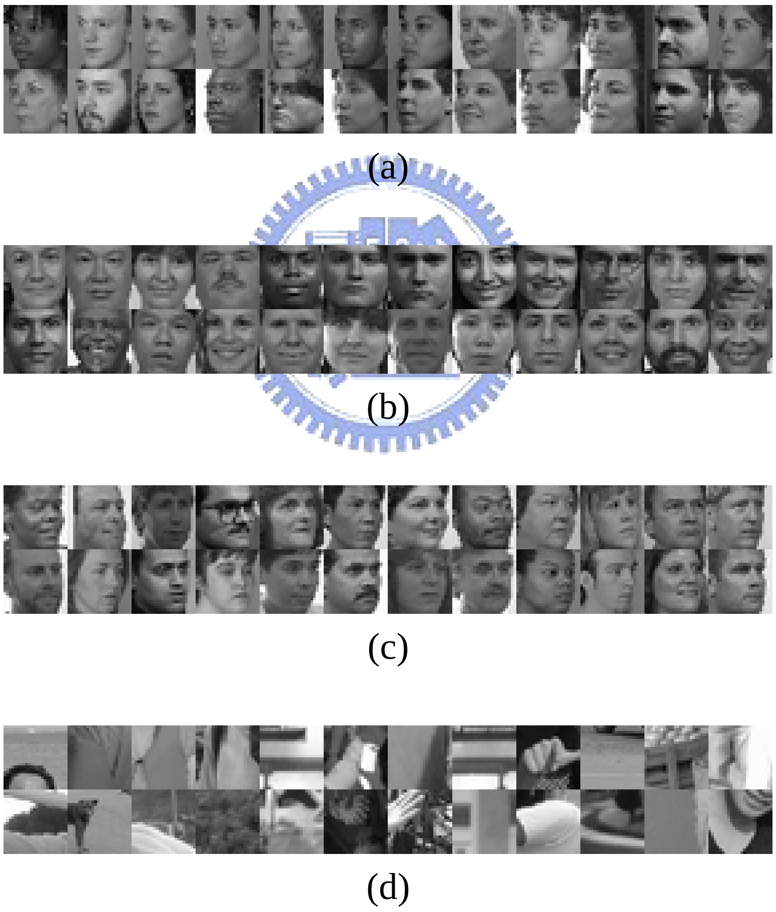
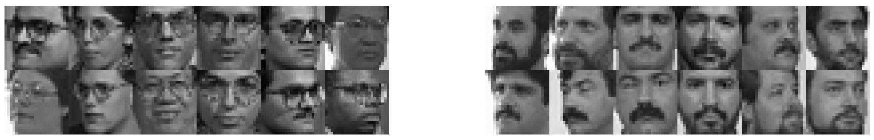


Fig. 2.3 (a)-(c) The normalized training samples of three different poses (-45^{o}, 0^{o}, 45^{o}) face. (d) The normalized training samples of non-face.



(a) (b)

Fig. 2.4 Some examples with special characteristics. (a) examples with glasses. (b) examples with moustache.

2.2 Features

2.2.1 Rectangle feature

Our face detection procedure classifies images based on the value of rectangle features. Rectangle features are sensitive to the presence of edges, lines, and other simple image structure. They provide rich image representation to support effective training. Fig. 2.5 shows seven different type features used in our experiment. Among a sample with 24*24 pixels, the width and height of Figs. 2.5(a)(b) have 12 different scales, Figs. 2.5(c)(d)(g) have 8 different scales, and Figs. 2.5(e)(f) have 6 different scales. Thus, we have 2*12*12*24*24 + 3*8*8*24*24 + 2*6*6*24*24= 165888 + 110592 + 41472 = 317952 possible features among single training sample.

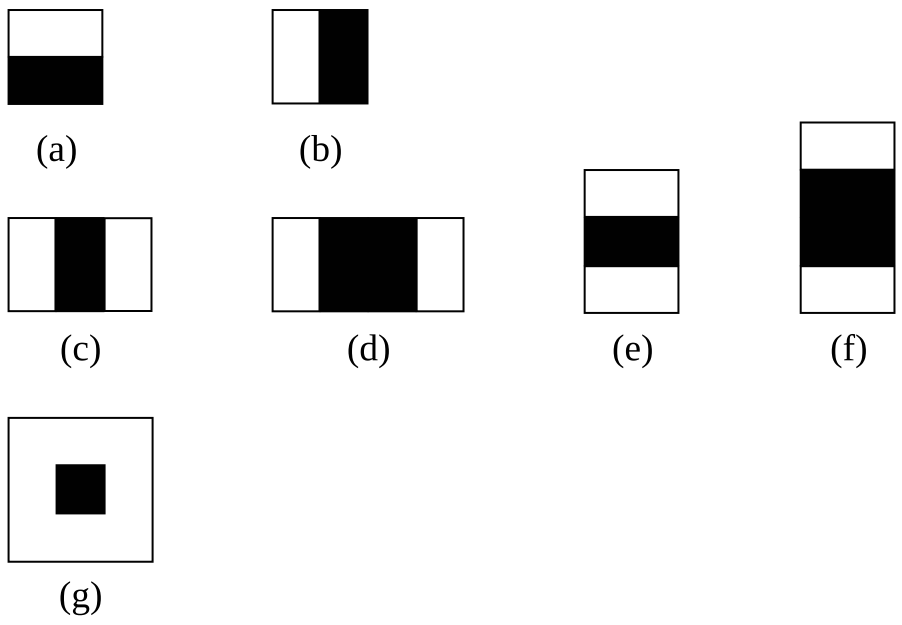


Fig. 2.5 Seven different type rectangle features used in our experiment. The value of rectangle feature is the difference between the sums of the pixel gray level values within the black and white rectangular regions. Two- rectangle features are shown in (a), (b) and (g). Three-rectangle features are shown in (c)-(f).

2.2.2 Integral image

Rectangle features can be computed very quickly using an intermediate representation for the image which we call the integral image. The integral image at location *x*, *y* contains the sum of the pixels above and to the left of *x*, *y*:

$$ii(x, y) = \sum_{x' \le x, y' \le y} i(x', y') \qquad (3\text{-}1)$$

where $ii(x, y)$ is the integral image and $i(x, y)$ is the original image.

Any rectangular sum can be computed in four array references using the integral image. As shown in Fig. 2.6, the value of the integral image at location 1 is the sum of the pixels in rectangle A. The value at location 2 is A + B, at location 3 is A + C, and location 4 is A + B + C + D. The sum within D can be computed as 4 + 1 - (2 + 3).

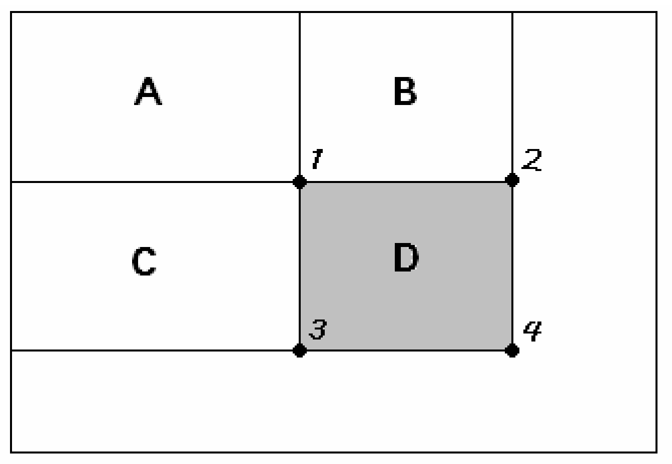


Fig. 2.6 The sum of the pixels in rectangle D can be computed as 4 + 1 - (2 + 3).

The two-rectangle features defined above involve adjacent rectangular sums so they can be computed with six array references. As shown in Fig. 2.7, the sum of the pixels in rectangle A can be computed as 4 + 1 - (2 + 3), and the sum of the pixels in rectangle B can be computed as 6 + 3 - (4 + 5). So the value of this rectangle feature is difference between the sum of the pixels within A and the sum of the pixels within B, which equals (4 - 3) - (2 - 1) + (4 - 3) - (6 - 5). In the case of the three- rectangle features, they can be computed in eight array references.

As mentioned above, the integral image can be computed from an image using a few addition and subtraction operations per pixel. Once computed, any one of these rectangle features can be computed rapidly at any scale or location.

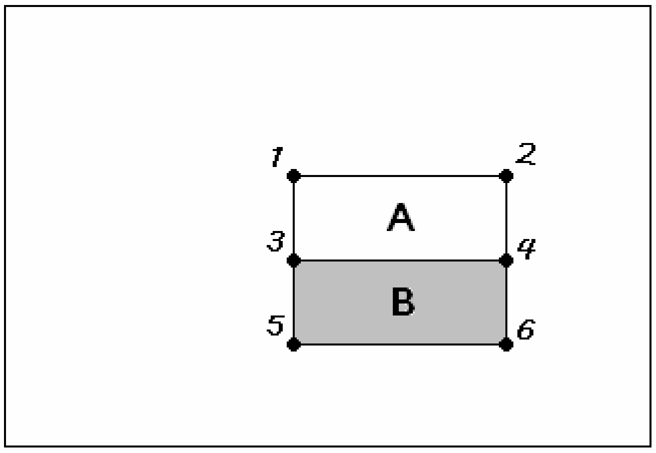


Fig. 2.7 The value of two-rectangle feature can be computed in six array references.

2.3 AdaBoost

In each 24*24 pixel image sub-window, there are 317952 rectangle features. Even if we can compute a feature very efficiently, selecting a small number of features to form an effective classifier is a difficult task. In order to ensure fast classification, the learning algorithm must exclude a majority of features, and focus on some important features.

In our system, we use AdaBoost learning algorithm to select a small set of features and train the classifier [8]. AdaBoost learning algorithm provides a simple method to boost the classification performance. A weak classifier can depend on only a single feature. AdaBoost learning algorithm selects a new weak classifier at each round, and combines a small number of weak classifiers to form a stronger classifier.

Given a training set, we separate it to two classes, positive and negative. Positive examples consist of face images, and negative examples consist of non-face images. The weak learning algorithm selects the single rectangle feature to best separate positive and negative examples. For each feature, the weak learner finds out the optimal threshold classification function, which minimizes the number of misclassified examples. The function is shown as below:

$$h_j(x) = \begin{cases} 1 & \text{if } p_j f_j(x) < p_j \theta_j \\ 0 & \text{otherwise} \end{cases} \tag{3-2}$$

where j is the number of each feature, h_j is a weak classifier, f_j is the value of feature, θ_j is a threshold, and a polarity p_j indicates the direction of the inequality sign.

AdaBoost algorithm is shown as below:

- Given example images $(x_1, y_1), \dots, (x_n, y_n)$

  where $y_i = \begin{cases} 1 & \text{for positive examples} \\ 0 & \text{for negative examples} \end{cases}$

- Initialize weights:

$$w_{1,i} = \begin{cases} \frac{1}{2l} & \text{for } y_i = 1 \\ \frac{1}{2m} & \text{for } y_i = 0 \end{cases}$$

  where l and m are the number of positives and negatives respectively.

- For $t = 1, \dots, T$:

  1. Normalize the weights,

$$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$

     so that w_t is a probability distribution.

  2. For each feature, j, the error is $\varepsilon_j = \sum_i w_i \,|h_j(x_i) - y_i|$.

  3. Choose the classifier, h_t, with the lowest error ε_t.

  4. Update the weights:

$$w_{t+1,i} = \begin{cases} w_{t,i}\beta_t & \text{if example } x_i \text{ is classified correctly} \\ w_{t,i} & \text{if example } x_i \text{ is classified incorrectly} \end{cases}$$

where $\beta_t = \dfrac{\varepsilon_t}{1-\varepsilon_t}$

- The final strong classifier is:

$$h(x) = \begin{cases} 1 & \dfrac{\sum_{t=1}^{T} \alpha_t h_t(x)}{\sum_{t=1}^{T} \alpha_t} \geq \dfrac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

where $\alpha_t = \log \dfrac{1}{\beta_t}$

Each round of boosting selects one feature from the 317952 potential features. After T rounds of boosting, we can get T weak classifiers. At each round, the weights, w, will be updated; the weights of misclassified examples become larger. For example, there are ten training samples S_1, … , S_{10} and the weight of each sample is 0.1. After first round of boosting, selected feature makes S_1 and S_2 misclassified, and total error rate is 0.2. The weights of these two samples misclassified become 0.2778, and the weights of remaining eight samples become 0.0556. After second round of boosting, selected feature makes S_1 misclassified again, and total error rate becomes 0.2778. The weight of S_1 becomes 0.5806, the weight of S_2 becomes 0.1613, and the weights of remaining eight samples become 0.0323. Hence, when a sample is misclassified again and again, its weight becomes lager. Relatively, the high weight value increases total error rate and that also influences which feature we should select. Besides, if a feature misclassifies sample with large weight, total error rate becomes lager and the possibility of this feature selected is reduced. But if a feature has the capability of correctly classifying sample with large weight, total error rate becomes smaller and the possibility of this feature selected is increased relatively. Compared with tradition feature selection, the tradition method selects all critical features with error rate order at only one round of boosting. The weight of each sample is uniform, so selected features are not capable of classifying "hard classified example" and the possibility of false alarm increases. Fig. 2.8 illustrates the number of misclassified examples among 1000 training data with different number of features selected. From the figure, more features selected by AdaBoost can decrease misclassified example. On the contrary, more features selected without AdaBoost can not separate positive and negative efficiently. In Chapter 4, we will compare the performances between two different feature selections. Fig. 2.9 shows the first five features selected with two different feature selections and their error rate.

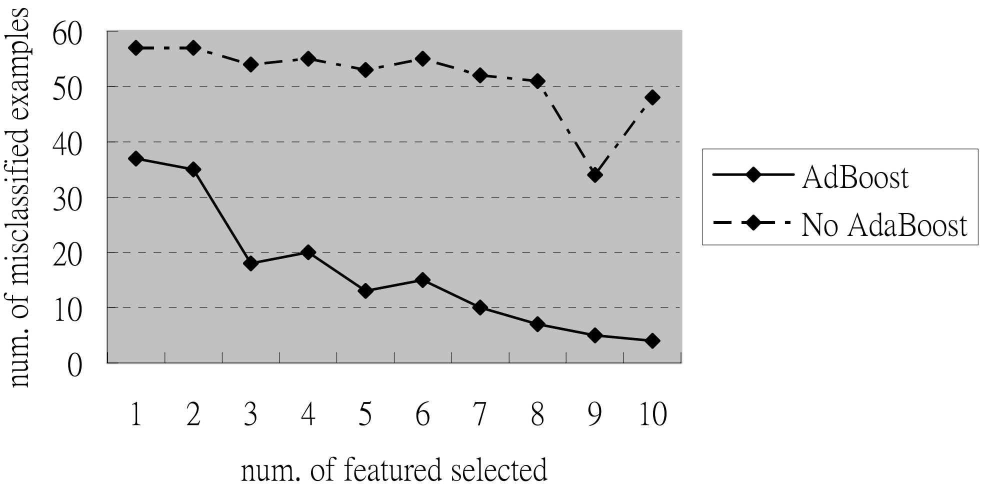
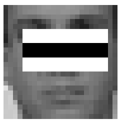
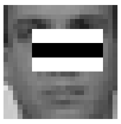
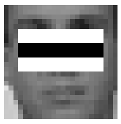
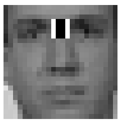
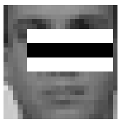


Fig. 2.8 The number of misclassified examples among 1000 training data with different number of features selected.

No AdaBoost

Feature					
Error rate	0.0709	0.0718	0.0718	0.0736	0.0736

AdaBoost

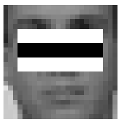
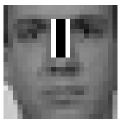
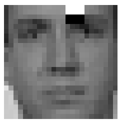
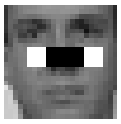
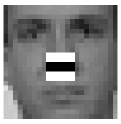
Feature					
Error rate	0.0683	0.0820	0.1133	0.1527	0.1484

Fig. 2.9 The first five features selected with and without using AdaBoost and their error rate

The confident weight, α, of each selected feature is determined by the error of each feature. It represents the importance of each selected feature. The confident weight is higher, the selected feature is more important, and it can separate positive and negative examples more efficiently.

2.4 Cascade Classifiers

Cascade classifier is a structure which achieves good detection performance and reduces computation time. Stages in the cascade structure are constructed by training classifier using AdaBoost.

As shown in Fig. 2.10, the initial stage classifier can be constructed from a small number of features to reject a large number of negative examples. Subsequent stages eliminate additional negative examples but require additional computation. Only positive examples can pass through all stages and they take the longest computation time. In each stage, the number of features and the method of feature selection are designed by user. For example, the first stage classifier can be constructed from two-feature strong classifier by adjusting threshold to minimize false negatives (mistakes positive as negative). The threshold can be adjusted to false negative rate of 0%, which detects 100% of the faces with false positive rate (mistakes negative as positive) of 40%. The subsequent stage classifiers are constructed from strong classifiers which consist of 5~10 features. They are designed by adjusting threshold to yield false negative rate under 5% and false positive rate may reduce to 20%. Final several stage classifiers are used more features which are designed by adjusting threshold to minimize the sum of false negative rate and false positive rate.

However, cascade structure saves a lot of time because a majority of sub-images in a picture are negative. It is also the key factor that system can detect faces in real time.

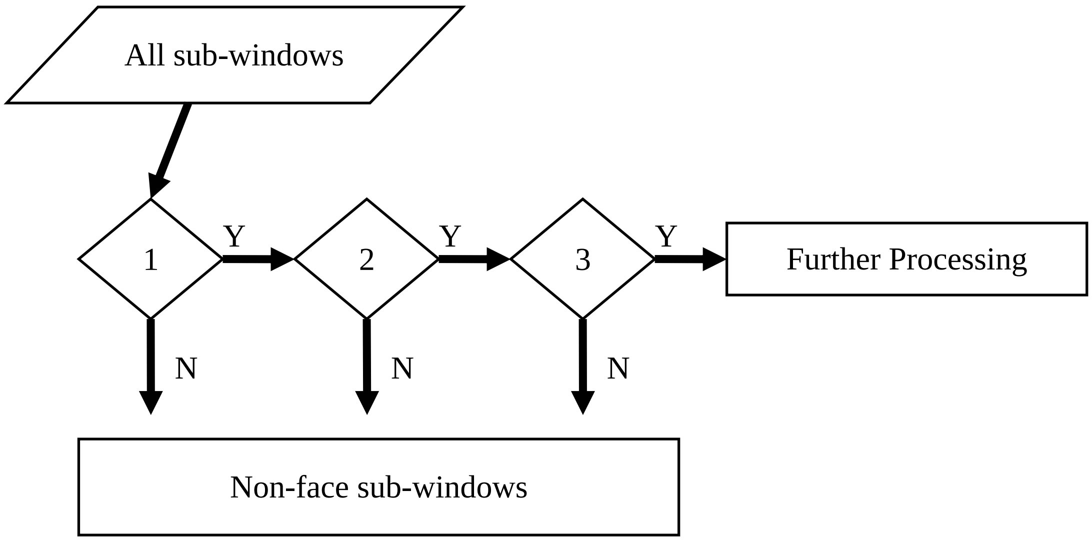


Fig 2.10 Cascade structure

2.5 System Structure in Our Experiment

In our face detection system, each sub-window image must pass skin detection procedure to segment potential face regions first (we will talk in Chapter 3), and then get a qualification for face detection. All qualified sub-window images can pass our face detection procedure, as shown in Fig. 2.11. The system can be separated to three layers. Each layer is composed of two-stage cascade structure. The first stage of each layer is a weak classifier constructed from only single feature. This feature is selected by adjusting threshold to pass all positive examples. That means false negatives rate is 0%. A majority of negative sub-window images are discarded at first stage classifier, so it can reduce processing time. The second stage of each layer is a strong classifier constructed from 200 important features which are selected by adjusting threshold to minimize the sum of false negative rate and false positive rate, so it can detect faces very accurately. Hence, our system is designed to take a balance between time-saving and accuracy.

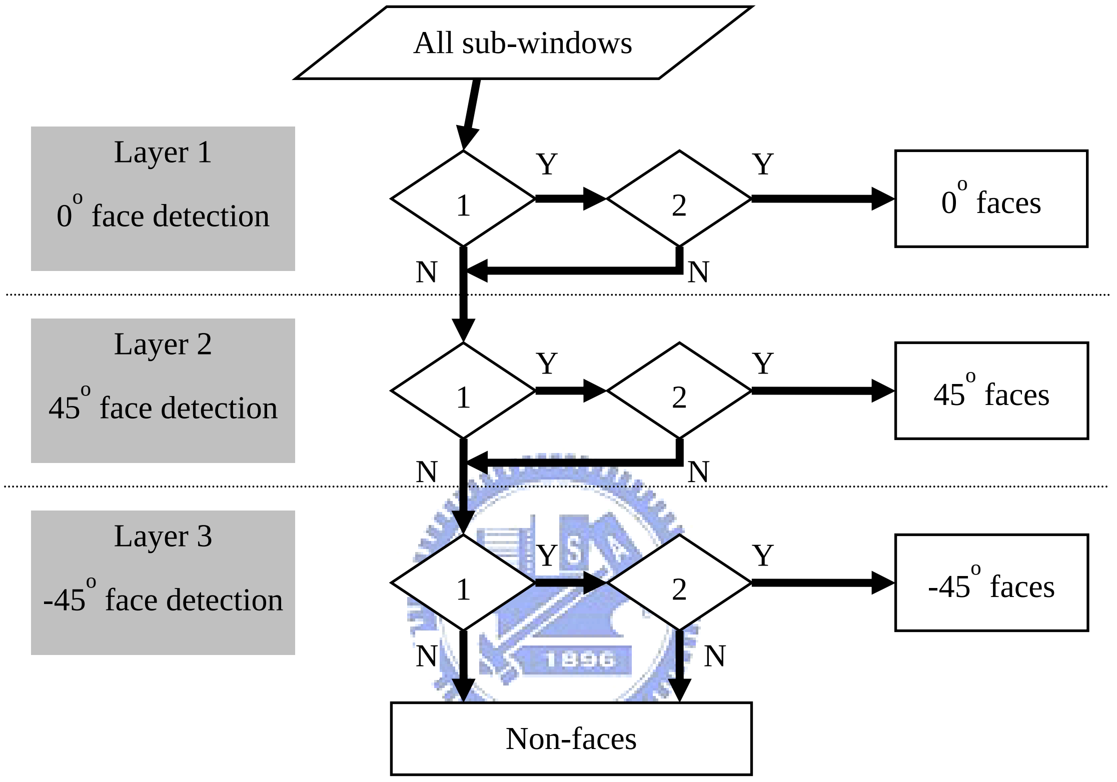


Fig 2.11 The structure of our face detection system

CHAPTER 3
SKIN COLOR REGION DETECTION

In real-time face detection, efficiency is the most important issue. In order to improve detection speed, we reduce the scan region. The main part of the image consists of non-skin color pixels. The first step in face detection algorithm is using potential face regions to reject no-face regions of the image.

Given a color image, we can detect the skin color regions and non-skin color regions. The color image is converted to the binary image. The skin color pixels are converted to “white” pixels and non-skin color pixels are converted to “black” pixels. The face detection algorithm is only proceeded at the white pixels. Besides, before face detection algorithm, we will do an additional procedure to decide if face detection algorithm need to be executed. The procedure is re-checking skin color area in the detection sub-window. If the human face is detected with sub-window, the main part of sub-window consists of skin color pixels. Hence, we set threshold to reject non-faces that are mistaken for faces. Fig. 3.1 shows the preprocessing of face detection.

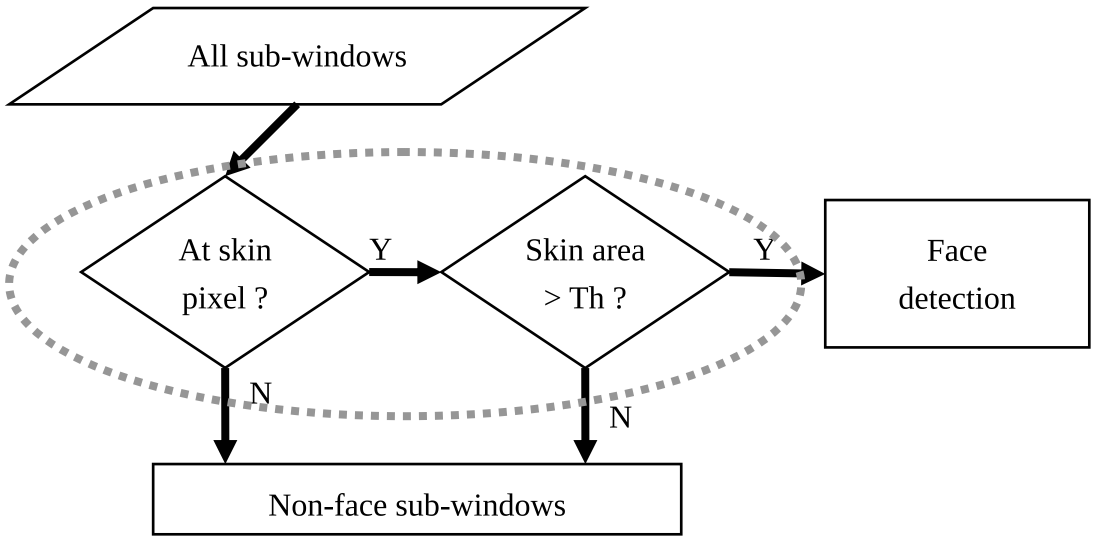


Fig. 3.1 The structure of preprocessing of face detection.

3.1 Skin Segmentation

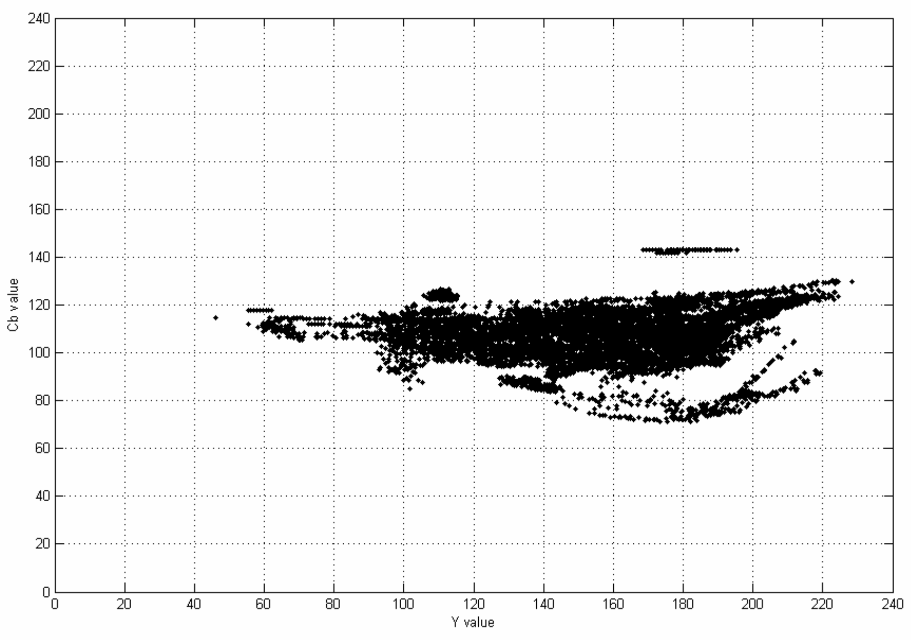
In this thesis, we will choose YCbCr domain [9] as our color space to segment skin color. There are two reasons why we choose YCbCr color space. The first reason is that the YCbCr color space is used broadly in computer vision applications such as JPEG compression, MPEG and H.263 video compression. We can use it directly without converting it to another color space. The second reason is that the YCbCr color space has good performance in skin segmentation. According to the research [10], the skin pixels of different-raced people have similar distribution between Cb and Cr values. So we can easily segment skin color of the different-colored person.

In the RGB domain, each component of the picture (red, green, and blue) has a different brightness. However, in the YCbCr domain all information about the brightness is given by the Y component and its value has range from 16 to 235, which represent from darkness to lightness. The Cb and Cr components are independent from the luminosity and its value has range from 16 to 240. The following conversions are used to segment R, G and B components into Y, Cb and Cr components:

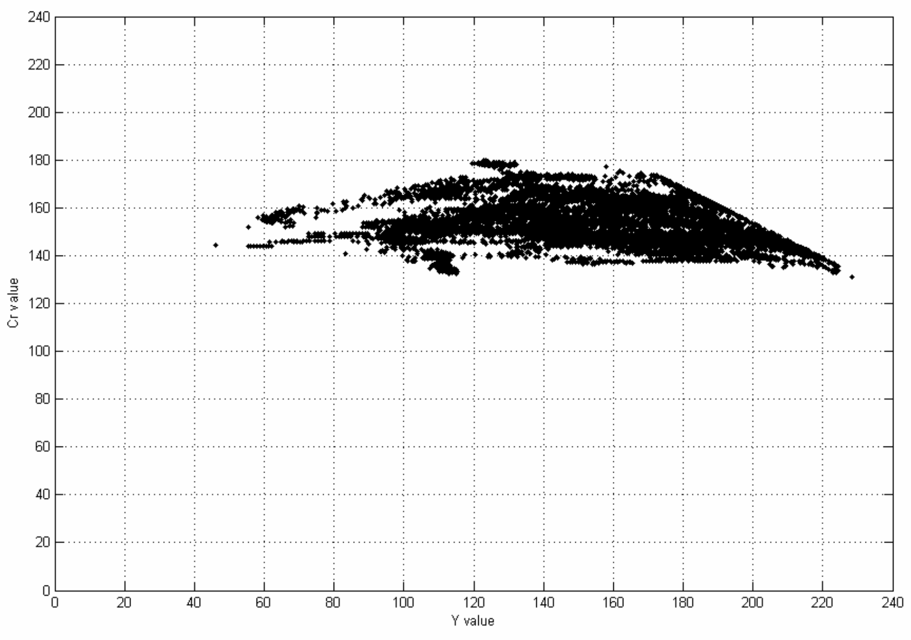
$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0.257 & 0.504 & 0.098 \\ -0.148 & -0.291 & 0.439 \\ 0.439 & -0.368 & -0.071 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \qquad (3\text{-}1)$$

3.1.1 Thresholds for Skin Segmentation

We take 10000 skin samples to get their Y, Cb, Cr components. Fig. 3.2(a) shows the distribution between Cb and Y values of skin samples. Fig. 3.2(b) shows the distribution between Cr and Y values of skin samples. It is unwise to apply maximum and minimum of Cb and Cr components for the thresholds of skin segmentation. We decide to choose thresholds of Cb and Cr components with different Y intervals, as shown in Fig. 3.3. Table 3.1 clearly lists the conditions with different Y intervals.

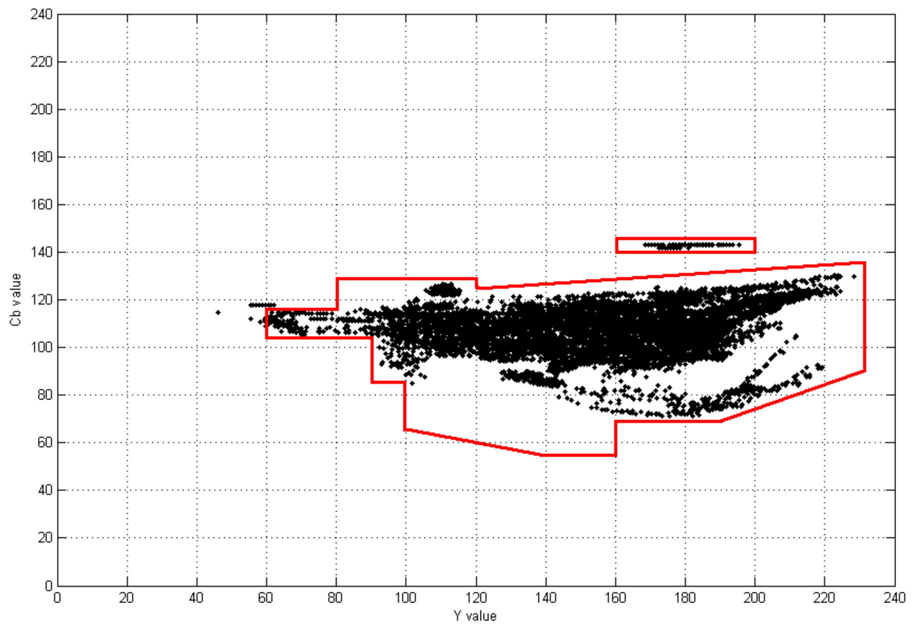


(a)

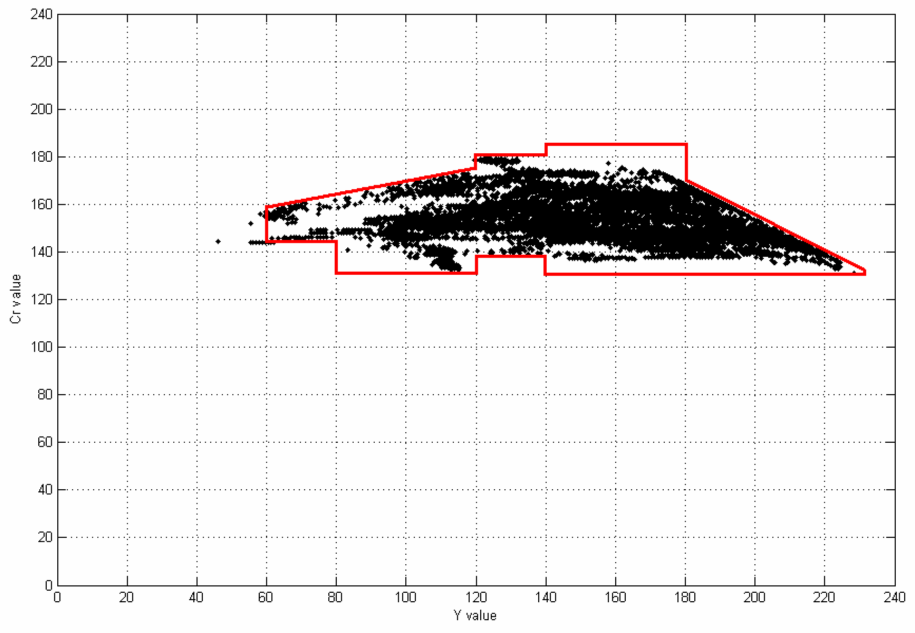


(b)

Fig. 3.2 (a) Distribution between Cb and Y values of 10000 skin samples. (b) Distribution between Cr and Y values of 10000 skin samples.



(a)



(b)

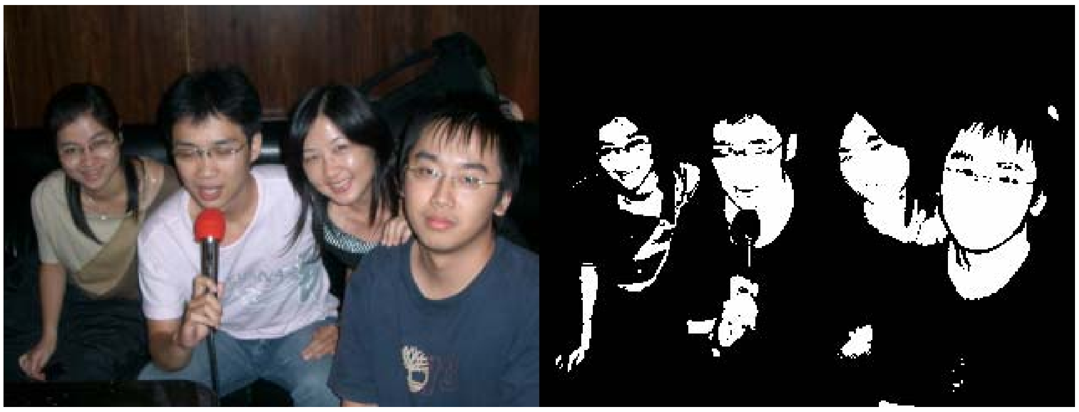
Fig. 3.3 The skin region is inside red threshold curves.

$60 < Y \le 80$	$105 < Cb < 115$	$145 < Cr < 0.25 * Y + 144$
$80 < Y \le 90$	$105 < Cb < 128$	$132 < Cr < 0.25 * Y + 144$
$90 < Y \le 100$	$85 < Cb < 128$	$132 < Cr < 0.25 * Y + 144$
$100 < Y \le 120$	$(-0.5) * Y + 125 < Cb < 128$	$132 < Cr < 0.25 * Y + 144$
$120 < Y \le 140$	$(-0.5) * Y + 125 < Cb < 0.1 * Y + 113$	$139 < Cr < 180$
$140 < Y \le 160$	$55 < Cb < 0.1 * Y + 113$	$130 < Cr < 185$
$160 < Y \le 180$	$70 < Cb < 0.1 * Y + 113$ $140 < Cb < 145$	$130 < Cr < 185$
$180 < Y \le 190$	$70 < Cb < 0.1 * Y + 113$ $140 < Cb < 145$	$130 < Cr < (-0.75) * Y + 305$
$190 < Y \le 200$	$(-0.5) * Y + 165 < Cb < 0.1 * Y + 113$ $140 < Cb < 145$	$130 < Cr < (-0.75) * Y + 305$
$190 < Y \le 232$	$(-0.5) * Y + 165 < Cb < 0.1 * Y + 113$	$130 < Cr < (-0.75) * Y + 305$

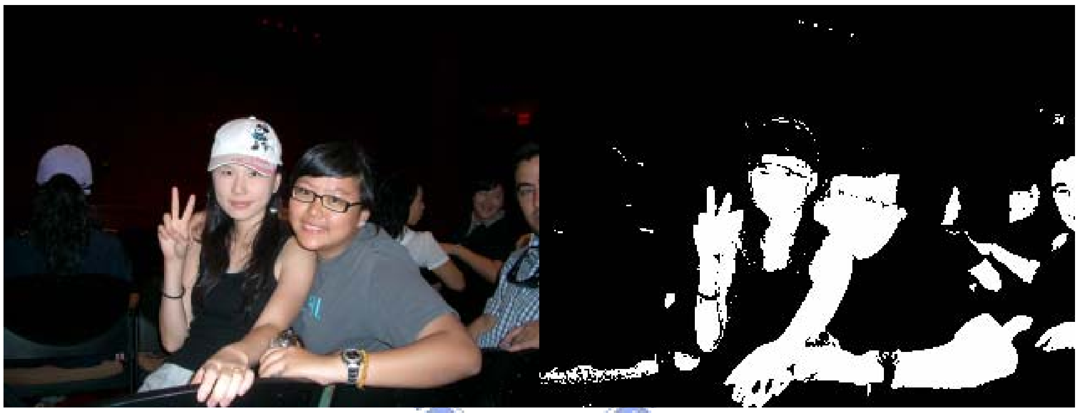
Table 3.1 For skin segmentation, Cb and Cr values should correspond to conditions with different Y intervals.

3.1.2 Binary Image Processing

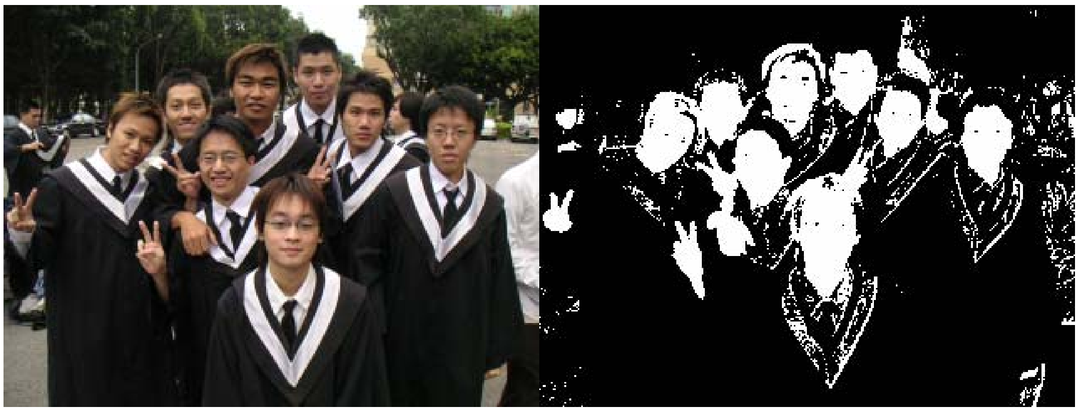
Based on these conditions in Table 3.1, a binary image is obtained. The white pixels represent skin pixels and the black pixels represent non-skin pixels, as shown in Fig. 3.4.



(a)



(b)

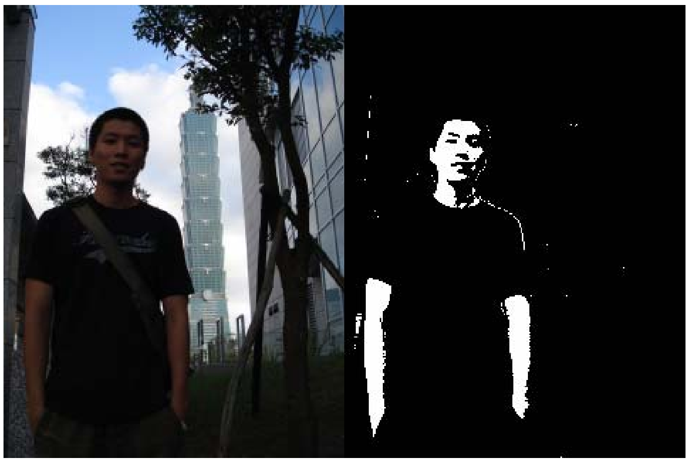


(c)

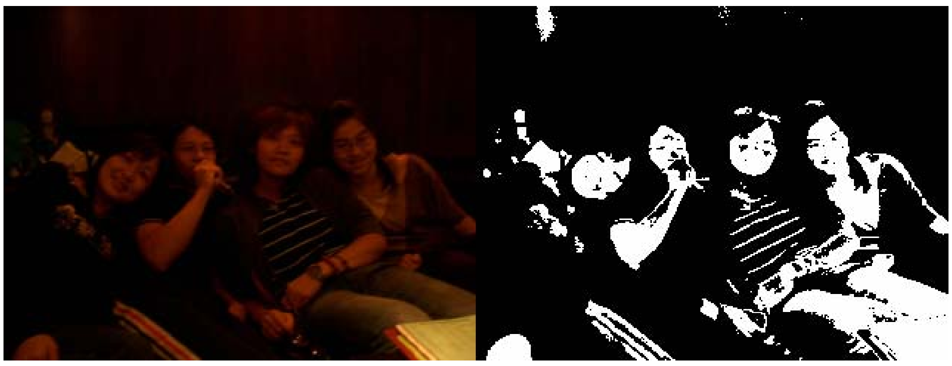
Fig. 3.4 Some examples of converting original images to binary images. (a)(b) indoor environment, (c)outdoor environment.

3.1.3 Experiment Results and Discussions

As shown in Figs. 3.4(a)-(c), we can see the skin color can be segmented correctly in indoor or outdoor environment. There are some noises in Fig. 3.4(c), but the parts of faces and hands still can be segmented. Fig. 3.4 shows two special cases. In Fig. 3.5(a), the man stands with his back to light. Although his face and his arm are dark, the skin segmentation performance is good. Fig. 3.5(b) shows in the poor illumination situation, the result is still acceptable.



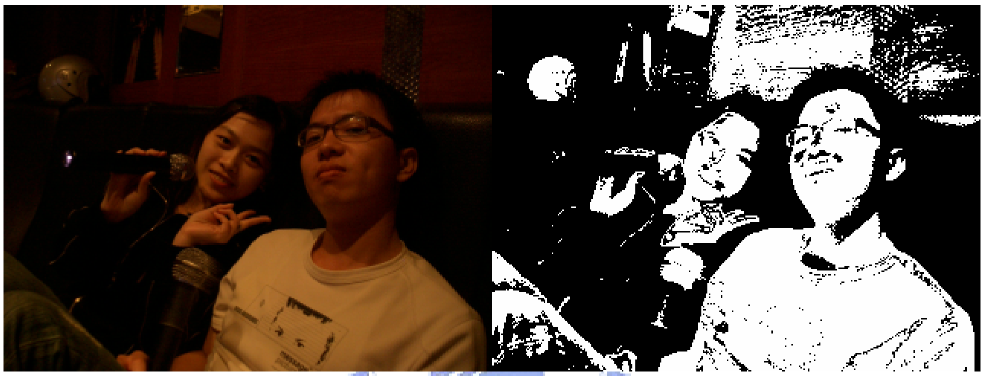
(a)



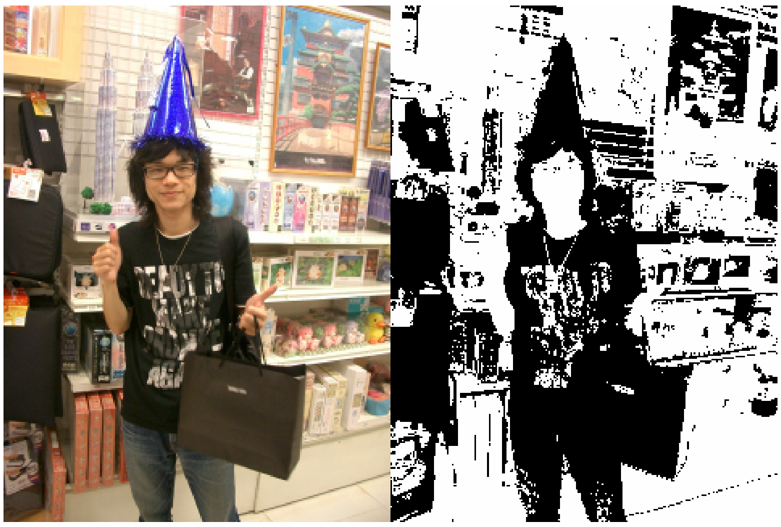
(b)

Fig. 3.5 Two cases in poor illumination environment.

The following two examples in Figs. 3.6(a)-(b) shows bad results for skin segmentation process. Fig. 3.6(a) shows the result in the dark indoor environment. In the poor illumination situation, the white shirt of the man is similar to skin color and so are some parts of background. Fig. 3.6(b) shows the result in complex background. There are too many skin-color elements in the background to segment human skin color clearly.



(a)



(b)

Fig. 3.6 Two bad results of skin segmentation.

3.2 Preprocessing of Face Detection

Given a color image, the first step is segmenting skin part of the image. At every skin pixel, the face detection algorithm is performed with different sub-windows. The result of detection sometimes makes mistake, it recognizes no-faces as faces, as shown in Fig. 3.7.

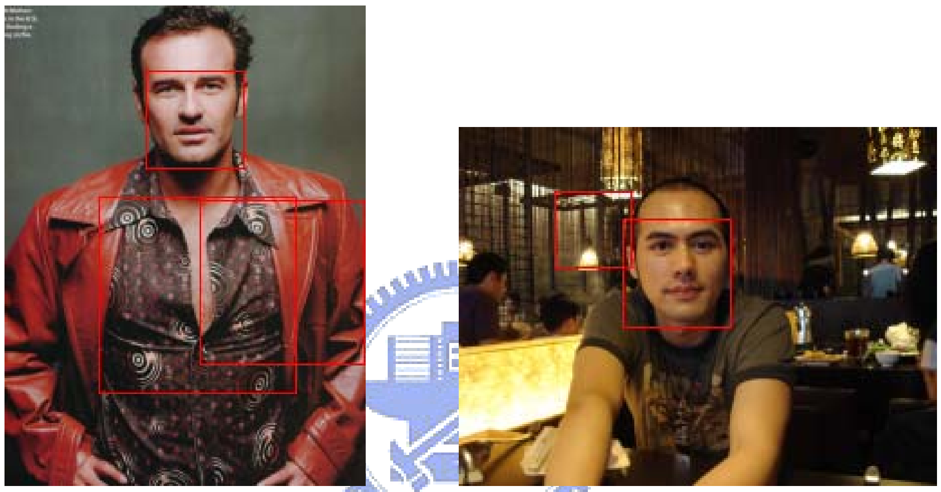

(a) (b)

Fig. 3.7 Two examples of bad results of face detection.

To eliminate these false results, we use a simple method before face detection. This method is based on the ratio of skin region area to total area in the sub-window, as shown in Eq. 3-2:

$$skin\ ratio = \frac{the\ area\ of\ skin\ region\ in\ the\ sub\text{-}window}{the\ total\ area\ in\ the\ sub\text{-}window} \times 100\% \qquad (3\text{-}2)$$

The *skin ratio* represents the percentage of skin pixels in the sub-window. It is apparent that non-faces and faces can be distinguished by applying an appropriate threshold value. The non-face samples will be rejected when the *skin ratio* is less than the threshold

The threshold is evaluated from 535 correct detected face samples in our experiment. We compute the *skin ratio* of every sample and sort these data by their scale. Fig. 3.8 shows some examples in our experiment. Fig 3.9 shows distribution of all correct detected face samples.

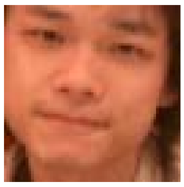
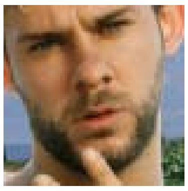
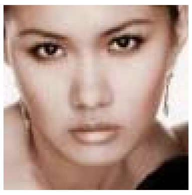
samples				
Skin ratio	83%	99%	76%	91%

Fig. 3.8 Some examples in our experiment and their skin ratio.

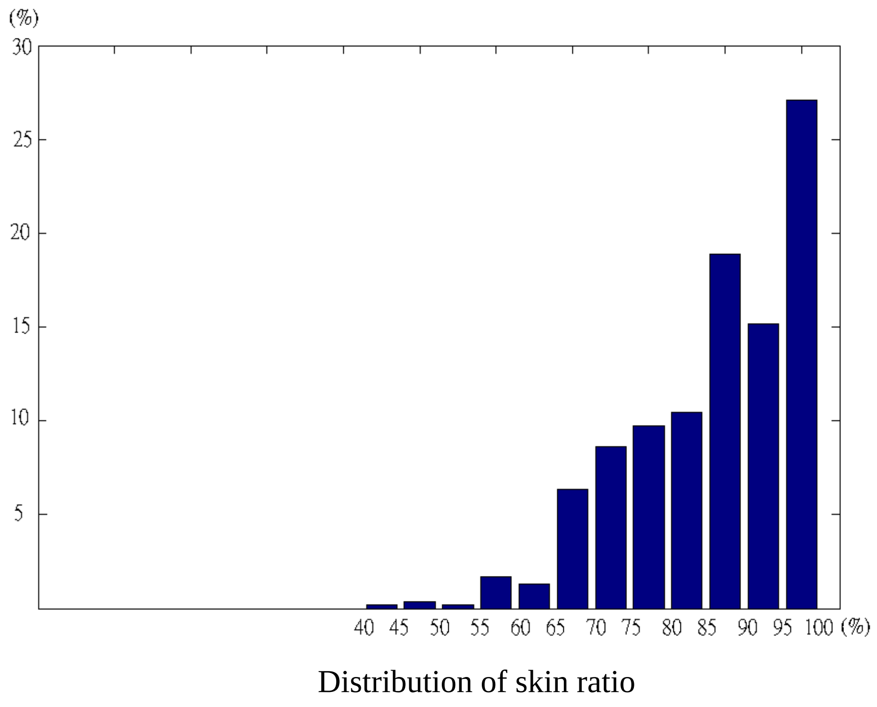
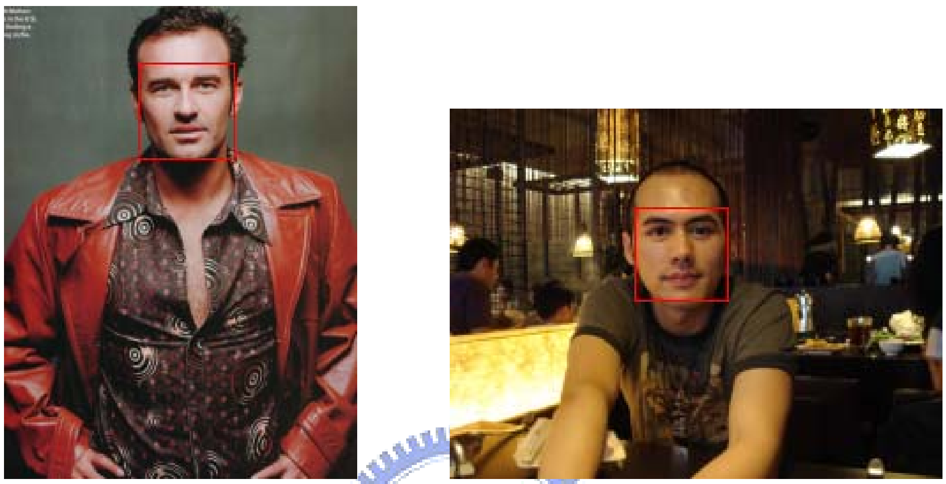


Fig. 3.9 Distribution of skin ratio of 535 correct detected face samples.

From Fig. 3.9, over 95% of the samples can be detected correctly when skin ratio is greater than 65%, so the threshold 65% is used in our experiment. Fig. 3.10 shows two examples from Fig. 3.6, the results with preprocessing are satisfied.



(a) (b)

Fig. 3.10 Two examples with preprocessing from Fig. 3.6.

CHAPTER 4
EXPERIMENTAL RESULTS AND DISCUSSIONS

4.1 Detection sub-windows

4.1.1 Nearest neighbor interpolation

For face detection, we scan input image with different-size detection sub-windows. Usually, the faces in the image have different scales. So we need to normalize all sub-windows to a standard size (24*24 pixels). The method of resizing the sub-windows is using the nearest neighbor interpolation [11].

Nearest neighbor interpolation is shown as below:

- Given a sub-windows with width m and height n,
  its pixel value of position (x, y) is $p(x, y)$,
  where $0 \le x < m$ and $0 \le y < n$.
- Create subsampled image with width m' and height n'.
  Its pixel value of position (x, y):

$$p'(x, y) = p(\ \text{floor}(\frac{m}{m'})x\ ,\ \text{floor}(\frac{n}{n'})y\ )$$

where $0 \le x < m' < m$, $0 \le y < n' < n$, and floor() is the function that rounds a float number to an integer.

For nearest neighbor interpolation, the subsampled pixel value is placed by the nearest pixel value. Fig. 4.1(a) shows the original image in the sub-windows, and Figs. 4.1(b)-(c) are the images after transformation. The subsampled image may appear step-like after transformation, but it causes a little effect in following procedure.

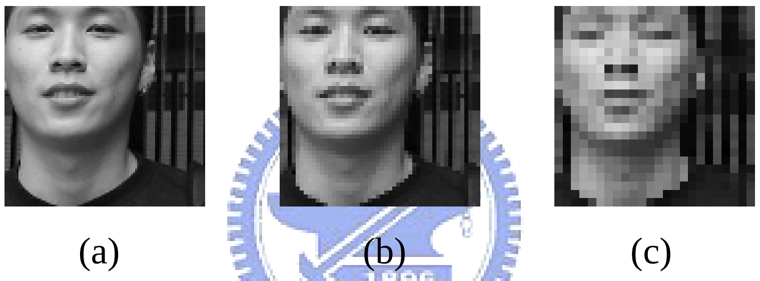


Fig. 4.1 (a) The original image with 100*100 pixels. (b)(c) The subsampled images with 50*50 pixels and 24*24 pixels.

4.1.2 The threshold of confidence

As mentioned in Section 2.2, the sub-windows are classified as positive if the final hypothesis exceeds the AdaBoost threshold (i.e. $\frac{\sum_{t=1}^{T} \alpha_t h_t(x)}{\sum_{t=1}^{T} \alpha_t} \ge \frac{1}{2}$). We define the confidence of a sub-window as:

$$Conf(x) = \frac{\sum_{t=1}^{T} \alpha_t h_t(x)}{\sum_{t=1}^{T} \alpha_t} \tag{4-1}$$

The *Conf*(*x*) is an index that decides the output of classifier is acceptable or not. The high confidence value represents the sub-image satisfies more features. In test procedure, when *Conf*(*x*) is larger than the threshold, *Th*, this sub-image is regarded as positive:

$$Conf(x) > Th \tag{4-2}$$

The threshold, *Th*, will be adjusted according to different test image set. In the beginning, the threshold is initialized to 0.5. Then, the threshold is gradually increased until minimizing the sum of miss rate and false alarm rate. In real-time system, optimal threshold is selected according to illumination and background conditions at that time. In our photo test experiment, optimal threshold of 0^{o} face detection is assigned to 0.6, and optimal thresholds of $\pm 45^{o}$ face detection are assigned to 0.67, as shown in Fig. 4.2. In our real-time system, optimal threshold of 0^{o} face detection is assigned to 0.56, and optimal thresholds of $\pm 45^{o}$ face detection are assigned to 0.6.

Layer 1
0^{o} face detection

Th	Miss Rate	False Alarm Rate
0.57	10.06%	11.50%
0.59	11.03%	9.89%
0.6	14.69%	6.16%
0.61	20.78%	5.43%

Layer 2
45^{o} face detection

Th	Miss Rate	False Alarm Rate
0.65	12.01%	19.10%
0.67	12.66%	11.80%
0.69	14.61%	9.93%

Layer 3
-45^{o} face detection

Th	Miss Rate	False Alarm Rate
0.65	11.03%	14.91%
0.67	11.69%	12.26%
0.69	12.34%	11.76%

Fig. 4.2 The optimal threshold selected at each layer.

4.1.3 Classification of potential faces

In the experiment, we illustrate the detected faces by red square. As shown in Fig. 4.3(b), there are usually a lot of red squares for single face. In order to eliminate this phenomenon, we combine adjacent red squares to one.

The first step is computing the centers ($\mathbf{x}_1, \ldots, \mathbf{x}_N$) of all potential face sub-windows, where N is the number of potential face sub-windows. Secondly, we assign $\mathbf{x}_1$ to $\mathbf{y}_1$, which is first member of the first class $\boldsymbol{C}_1$ and we will meet two different conditions:

<u>Condition 1</u> (when the number of class k is 1):

We compute the distances of $\mathbf{y}_1$ and $\mathbf{x}_1, \ldots, \mathbf{x}_N$ respectively. Once distance of $\mathbf{y}_1$ and $\mathbf{x}_i$ is larger than threshold, we assign $\mathbf{x}_i$ to $\mathbf{y}_2$, which is first member of the second class $\boldsymbol{C}_2$. If distance of $\mathbf{y}_1$ and $\mathbf{x}_i$ is smaller than threshold, we assign $\mathbf{x}_i$ to $\boldsymbol{C}_1$.

<u>Condition 2</u> (when the number of classes k is larger than 1):

We compute the distances of $\mathbf{y}_1, \ldots, \mathbf{y}_k$ and remaining centers respectively. Once distances of $\mathbf{y}_1, \ldots, \mathbf{y}_k$ and $\mathbf{x}_i$ are all larger than threshold, we assign $\mathbf{x}_i$ to $\mathbf{y}_{k+1}$, which is first member of the k+1th class $\boldsymbol{C}_{k+1}$. If distances of some $\mathbf{y}_j$ and $\mathbf{x}_i$ are smaller than threshold, we assign $\mathbf{x}_i$ to $\boldsymbol{C}_{j_0}$ which has the minimal distance of $\mathbf{y}_{j_0}$ and $\mathbf{x}_i$.

The remaining centers do Condition 2 procedure repeatedly until all potential face sub-windows are classified. Finally, we select the member of each class with the highest confidence value. The result is shown in Fig. 4.3(c).

The detailed algorithm is shown as below:

- Defined:

  $\mathbf{x}_1, \ldots, \mathbf{x}_N$: the centers of selected sub-windows

  $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_k$: the set of classes, where k is the number of classes

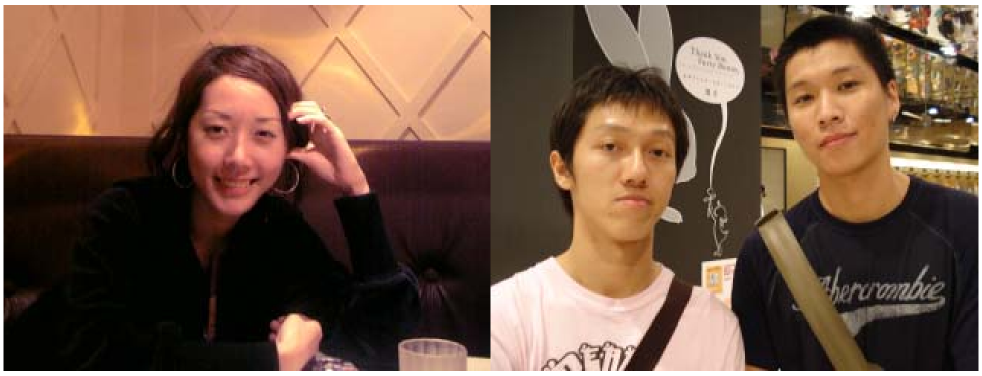
  $\mathbf{y}_1, \ldots, \mathbf{y}_k$: the set of first member of the classes

- Choose $\mathbf{x}_1$ as first member of $\boldsymbol{C}_1$: $\mathbf{y}_1 \leftarrow \mathbf{x}_1$, and $k = 1$.

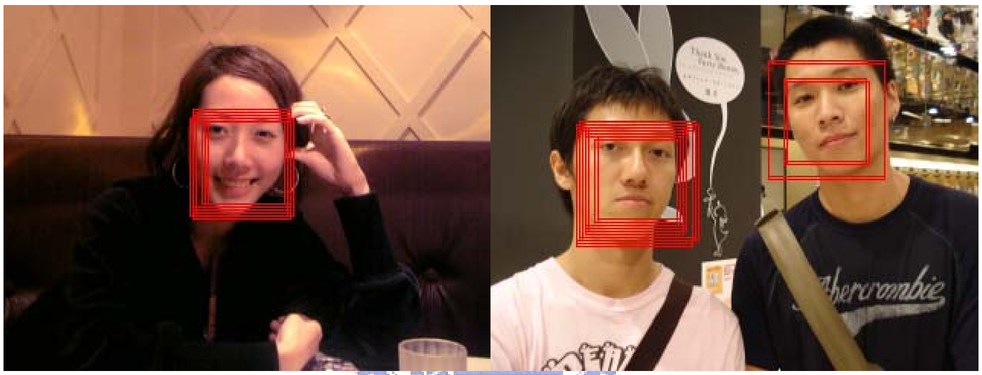
- For $i = 2, \ldots, N$:

```
For $j = 1, \ldots, k$ :
    if $\| \mathbf{x}_i - \mathbf{y}_j \| > Th$
        then    $\mathbf{y}_{k+1} \leftarrow \mathbf{x}_i$
                $k \leftarrow k + 1$
    else
        find $j_0$ such that $\| \mathbf{x}_i - \mathbf{y}_{j_0} \| = \min_{j=1}^{k} \| \mathbf{x}_i - \mathbf{y}_j \|$
        $\mathbf{x}_i \in \boldsymbol{C}_{j_0}$
```

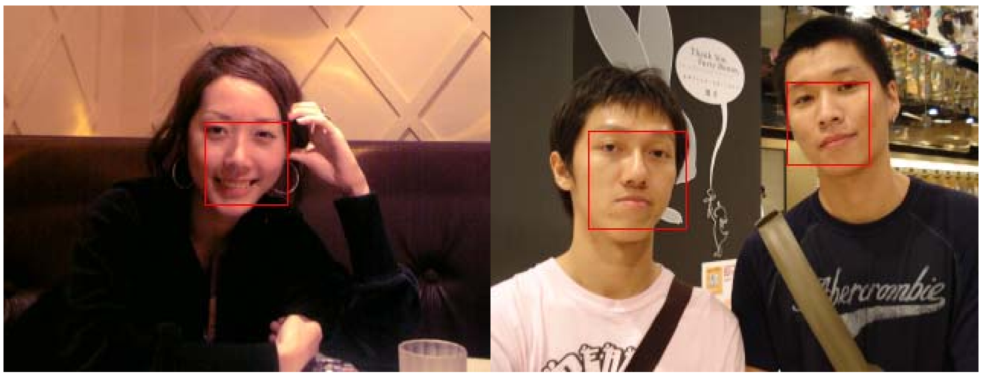
- Select one member with the highest confidence value in $\boldsymbol{C}_j$, $\forall j = 1, \ldots, k$



(a)



(b)



(c)

Fig. 4.3 (a) The original images. (b) The detection results without applying our algorithm. (c) The detection results with applying our algorithm.

4.2 Experiments on Real-World Photos

Our experimental environment is based on AMD 3500+ processor. The 100 test photos with 308 faces are all resized to 320*240 pixels.

4.2.1 Comparison with the Performances with Two Different Feature Selection Types

Here, we compare two different feature selection types. One is AdaBoost we already mentioned before, and the other one is feature selection depending on the order of error rate among all features. Unlike AdaBoost, the second type selects features at only one round. In order to prove that AdaBoost is more robust, we test our database and compare performances of two different types, as shown in Table 4.1. The table illustrates that the two feature selection methods almost have the same performances in detection rate. But on the other side, the false alarm rate without using AdaBoost is too high to tolerable, as shown in Fig. 4.4.

Feature Selection Method	Detection Rate	Miss Rate	False Alarm Rate
Order	87.99%	12.01%	78.94%
AdaBoost	88.31%	11.69%	12.26%

Table 4.1 Comparison to the performances with two different feature selection types.

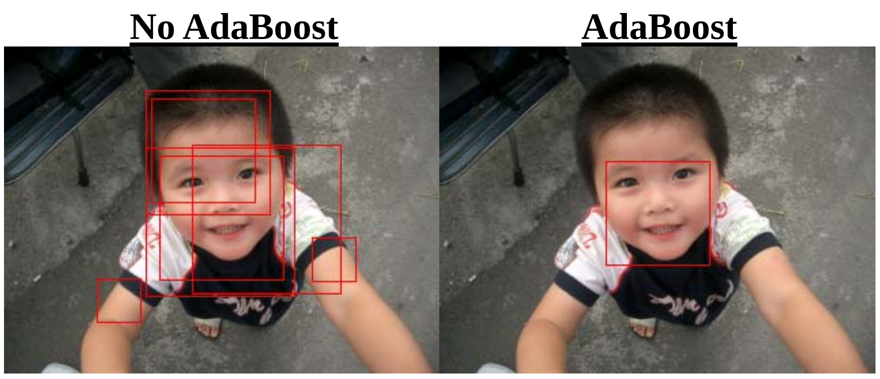


Fig. 4.4 Comparison between results by using feature selection without AdaBoost and with AdaBoost.

From Table 4.1, we can see 200 features selected without AdaBoost each layer introduce high false alarm rate 78.94%. Hence, in order to improve the performance, we select 100 better features as a new feature set. These new 100 features come from original 200 features, and we select them by number of each feature regarded as positive among all false alarm cases. We use new feature set to test our database, and comparison to the performance with old is shown as Table 4.2. We find the new method is useless, and the performance with new feature set is worst than old one. That is 100 features eliminated may be important at other position in an image, so the possibility of false alarm becomes higher.

.

Feature set	Detection Rate	Miss Rate	False Alarm Rate
Old	87.99%	12.01%	78.94%
New	87.66%	12.34%	83.08%

Table 4.2 Comparison to the performances with two different feature sets. **Old**: 200 features. **New**: 100 better features.

4.2.2 Comparison with the Performances using features with and without confident weight, α, selected by AdaBoost

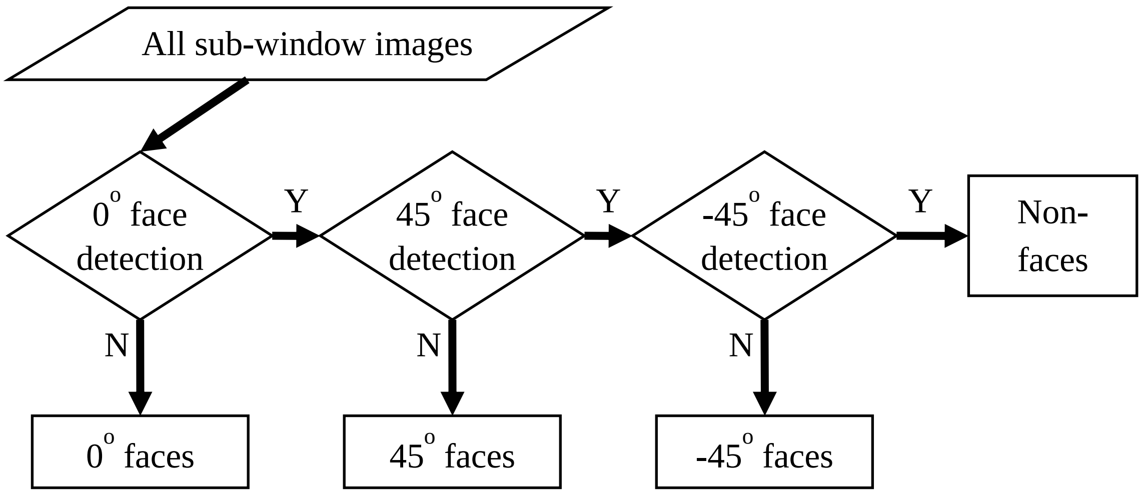
Here, we compare the performances using features with and without confident weight, α, selected by AdaBoost. As we mentioned in section 2.3, confident weight represents the importance of each selected feature. The confident weight is higher, the selected feature is more important. Table 4.3 shows the performances with and without confident weight. From the table, we can see the method with confident weight has better performance than without confident weight, but the method without confident weight is still much better than feature selected without AdaBoost. Therefore, the features selected by AdaBoost are critical and important for testing real-world photos.

	Detection Rate	**Miss Rate**	**False Alarm Rate**
With α	88.31%	11.69%	12.26%
Without α	84.09%	15.91%	15.08%

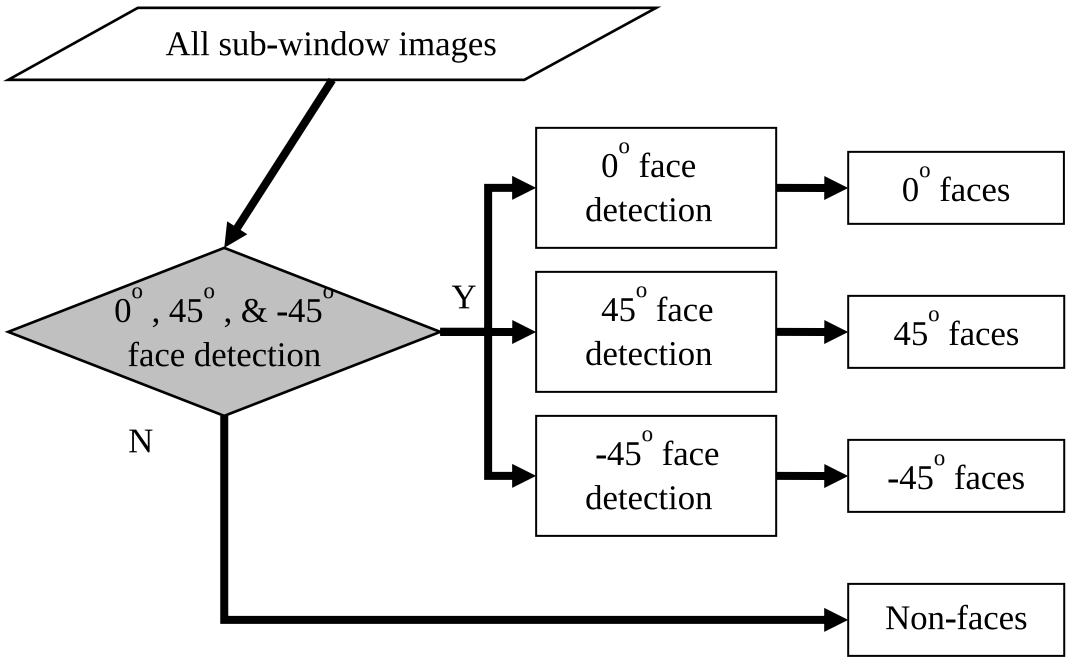
Table 4.3 Comparison to the performances with and without confident weight.

4.2.3 Comparison with the Performances of Two Different Systems

Fig. 4.5 shows two different systems. Fig. 4.5(a) is the system we mentioned before. As shown in Fig. 4.5(b), all sub-windows first do detection to determine if human face images or not, and then face images will be separated to three classes (-45^o, 0^o, 45^o). In the grey part of the figure, training set is composed of three pose ($-45^o+0^o+45^o$) faces. Table 4.4 shows the performances of (a) and (b). We can see (a) is better in detection rate but worse in false alarm rate. Thus, we compare their performances under equal false alarm condition by adjusting the thresholds of (b), as shown in Table 4.5. From the table, we can see two systems both produce 38 false alarm cases and detection rate of (a) is still better than (b). In our experiment, there are 13.31% positive examples are rejected at the grey part in the Fig. 4.5(b). Hence, the grey part is a major reason that miss rate of (b) is higher than (a).



(a)



(b)

Fig. 4.5 The structures of two different systems.

Result / System	Detection Rate	Miss Rate	False Alarm Rate
(a)	88.31%	11.69%	12.26%
(b)	82.47%	17.53%	10.25%

Table 4.4 The performances of two different systems.

Result / System	Num. of Detection	Num. of Miss	Num. of False Alarm
(a)	272 (88.31%)	36 (11.69%)	38 (12.26%)
(b)	260 (84.41%)	48 (15.58%)	38 (12.75%)

Table 4.5 The performances of two different systems under equal false alarm condition.

4.2.4 Dynamic analysis of systems without and with preprocessing

Table 4.6 shows comparison with dynamic analysis of systems without and with skin ratio preprocessing.

Result / Level	Detection Rate	Miss Rate	False Alarm Rate	Computation Time/Image
A1	82.14%	17.86%	57.76%	2156.19ms
A2	82.79%	17.21%	60.59%	2256.94ms
A3	84.09%	15.91%	62.02%	2312.95ms

(a)

Level A1: Cascade 0^o face detection.
Level A2: Cascade 0^o & $+45^o$ face detection.
Level A3: Cascade 0^o & $\pm 45^o$ face detection.

Level \ Result	Detection Rate	Miss Rate	False Alarm Rate	Computation Time/Image
B1	85.06%	14.94%	6.16%	70~1900ms
B2	86.69%	13.31%	8.59%	75~2000ms
B3	88.31%	11.69%	12.26%	80~2000ms

(b)

Level B1: Cascade 0^o face detection.
Level B2: Cascade 0^o & $+45^o$ face detection.
Level B3: Cascade 0^o & $\pm 45^o$ face detection.

Table 4.6 Comparison with dynamic analysis of systems (a) without skin ratio preprocessing and (b) with skin ratio preprocessing.

From the table, Level A does not use skin ratio preprocessing. We can see Level A1 uses 0^o face detection, and its detection rate is 82.14% and miss rate is 17.86%. After adding $+45^o$ face detection (Level A2), detection rate increases to 82.79% and miss rate reduces to 17.21%. Hence, under 0^o face detection (Level A1), there are 17.86% faces can not be detected. But after adding $+45^o$ face detection (Level A2), only 17.21% faces can not be detected. It means 0.65% (17.86% - 17.21%) faces originally miss at 0^o face detection (Level A1) layer, they can be detected. Finally, after entering -45^o face detection (Level A3) layer, detection rate increases to 84.09% and miss rate reduces to 15.91%. Hence, under Level A2, there are 17.21% faces can not be detected. But after adding -45^o face detection (Level A3), only 15.91% faces can not be detected. It means 1.3% (17.21% - 15.91%) faces originally miss at Level A1. After entering -45^o face detection (Level A3) layer, they can be detected.

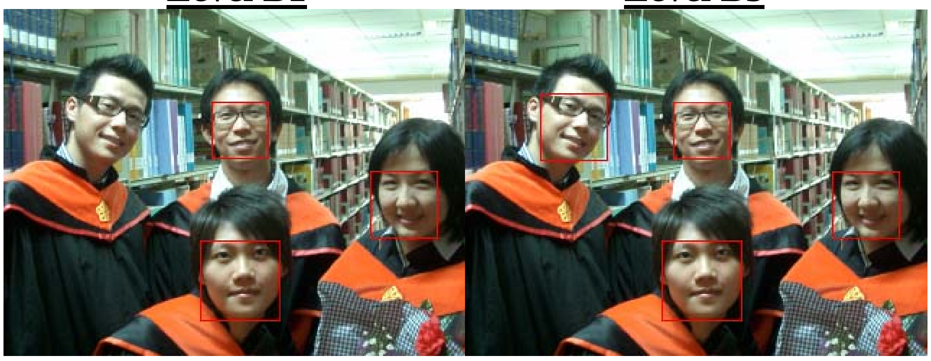
Besides, $\pm 45^o$ face detection increases the possibility of false alarm. The false alarm rate of Level A1 is 57.76%. After adding +45^o face detection (Level A2), the false alarm rate increases to 60.59%. After adding -45^o face detection (Level A3), the false alarm rate increases to 62.02%. It means 2.83% (60.59% - 57.76%) non-faces are originally rejected at Level A1, and 1.43% (62.02% - 60.59%) at Level A2. But after adding $\pm 45^o$ face detection, these 2.83% non-faces are detected and mistaken as +45^o faces at Level A2, and 1.43% are mistaken as -45^o faces at Level A3.

Level B uses skin ratio preprocessing. Under 0^o face detection (Level B1), 14.94% faces can not be detected. After adding +45^o face detection (Level B2), only 13.31% faces can not be detected. It means 1.63% (14.94% - 13.31%) faces originally miss at Level B1. After adding +45^o face detection (Level B2) layer, they can be detected. Finally, after adding -45^o face detection (Level B3), only 12.26% faces can not be detected. It means 1.05% (13.31% - 12.26%) faces originally miss at Level B2. After adding -45^o face detection (Level B3) layer, they can be detected. Fig. 4.6(a) shows Level B3 can detect more pose face than Level B1.

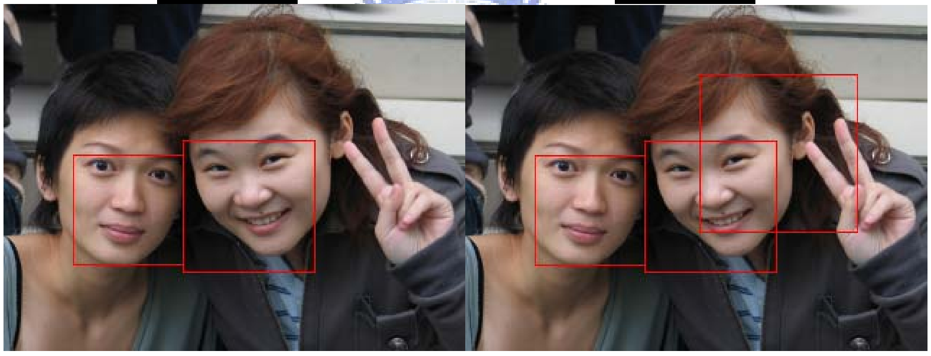
Besides, the false alarm rate of Level B1 is 6.16%. After adding +45^o face detection (Level B2), the false alarm rate increases to 8.59%. After adding -45^o face detection (Level B3), the false alarm rate increases to 12.26%. It means 2.43% (8.59% - 6.16%) non-faces are originally rejected at Level B1, and 3.67% (12.26% - 8.59%) at Level B2. But after adding $\pm 45^{o}$ face detection, these 2.43% non-faces are detected and mistaken as $+45^{o}$ faces at Level A2, and 3.67% are mistaken as -45^{o} faces at Level A3. Fig. 4.6(b) shows the result of Level B1 is better than Level B2 in false alarm rate.

In terms of skin ratio preprocessing, we compare Level A1 and Level B1. The detection rate of Level A1 without skin ratio preprocessing is 82.14%, and the detection rate of Level B1 with skin ratio preprocessing is 85.06%. So the system with using preprocessing has better performance. In terms of false alarm rate, the false alarm rate of Level A1 without skin ratio preprocessing is 57.76% and the false alarm rate of Level B1 with skin ratio preprocessing is 6.16%. The false alarm rate without skin ratio preprocessing (Level A1) is 51.6% (57.76% - 6.16%) higher than the false alarm rate with skin ratio preprocessing (Level B1). This is because a majority of non-face sub-windows are rejected with preprocessing. As we mentioned in section 3-2, if the real human face exists in a sub-window, the majority part of this sub-window must be skin color. Hence, false alarm rate 57.76% reduces to 6.16% with preprocessing. Besides, scanned regions just focused on skin color regions, so correct results were not influenced by complex background and it also can improve detection rate.

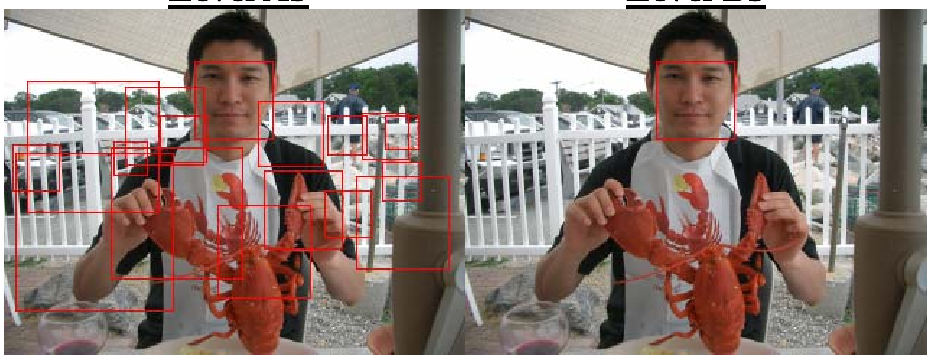
Compared with Level A3 and Level B3, the detection rate of Level A3 without skin ratio preprocessing is 84.09%, and the detection rate of Level B3 with skin ratio preprocessing is 88.31%. In terms of false alarm rate, the false alarm rate of Level A3 without skin ratio preprocessing is 62.02% and the false alarm rate of Level B3 with skin ratio preprocessing is 12.26%. The false alarm rate without skin ratio preprocessing (Level A3) is 49.76% (62.02% - 12.26%) higher than the false alarm rate with skin ratio preprocessing (Level B3). Fig. 4.6(c) shows the system with using preprocessing introduces high false alarm rate.



(a)



(b)



(c)

Fig. 4.6 Comparison the results of different levels in our system.

In terms of computational time, the case without using preprocess requires longer time to scan the whole image, and its computation time is usually constant. From Table 4.6, we can see that the system without using preprocess requires 2100~2300ms but the system with using preprocess just requires 80~2000ms. This is because the majority part of image is non-skin region, and we only need to scan skin color region. When skin color area is small, the detection speed is very quickly. Table 4.7 shows the average processing time per image with different skin color region size of single image among our test set.

Skin Color Area/Image	**Level B1 Computation Time**	**Level B3 Computation Time**
0% ~ 10%	131.33ms	149.47ms
10% ~ 20%	211.98ms	224.19ms
20% ~ 30%	427.35ms	487.11ms
30% ~ 40%	697.11ms	768.94ms
40% ~ 50%	984.99ms	1072.31ms
50% ~ 60%	1082.36ms	1124.75ms
60% ~ 70%	1372.24ms	1725.79ms

Table 4.7 The average processing time per image with different skin color region size.

4.2.5 Testing on Real-life Photos

After comparing with different levels in our system, we show some our experimental results. Fig. 4.7 and Fig. 4.8 separately show some good and bad test results.

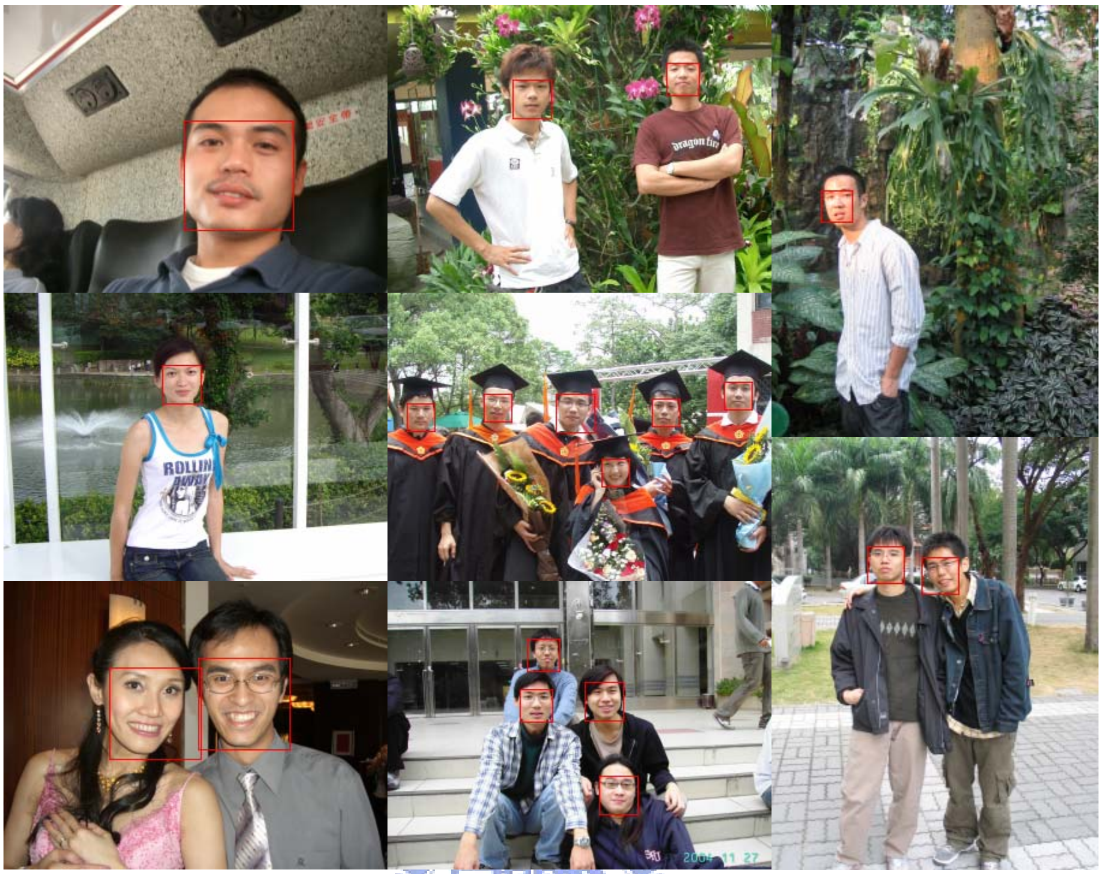


Fig. 4.7 Some good test results by applying our system.

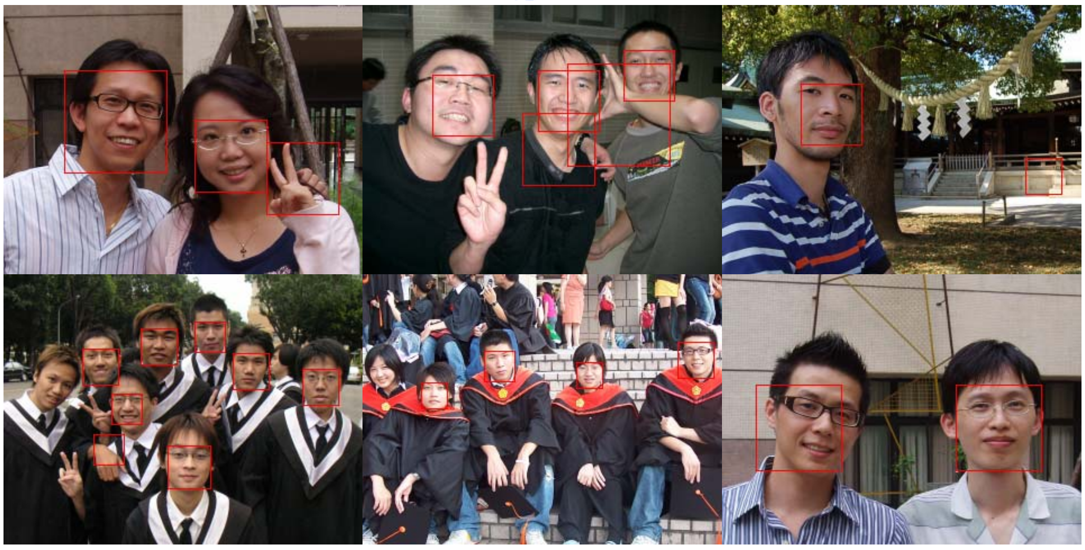
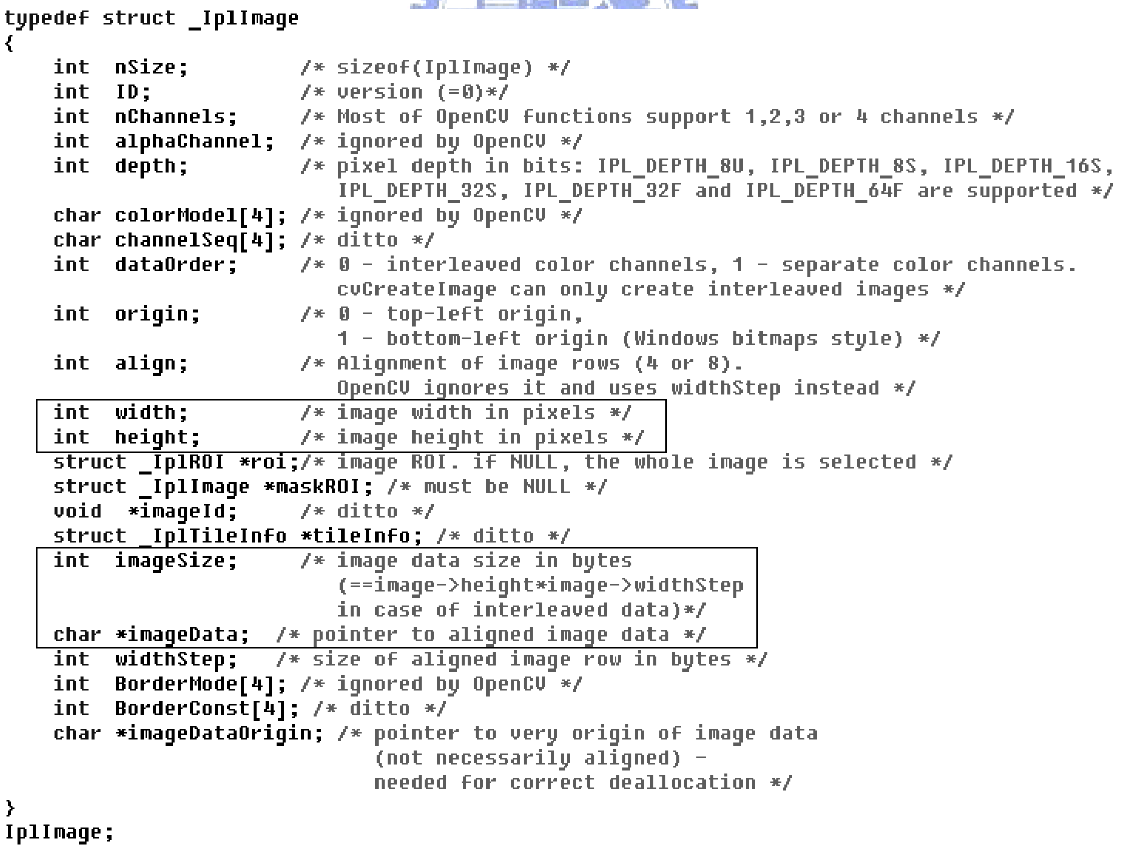


Fig. 4.8 Some bad test results by applying our system.

4.3 Experiments on Real-Time System

In this section, we implement real-time face detect system. The input image from a webcam is caught by function provided by OpenCV [12]. OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real time computer vision. Example applications of the OpenCV library are Human-Computer Interaction, Object Identification, Face Recognition, Gesture Recognition, Motion Tracking, and Mobile Robotics. OpenCV provides a structure named IplImage to process bmp raw data. Fig. 4.9 shows the members of the structure IplImage. Hence, we need to assign the pointer named imageData to input bmp data caught from webcam, and initialize the width and height of the image.



```
typedef struct _IplImage
{
    int  nSize;          /* sizeof(IplImage) */
    int  ID;             /* version (=0)*/
    int  nChannels;      /* Most of OpenCV functions support 1,2,3 or 4 channels */
    int  alphaChannel;   /* ignored by OpenCV */
    int  depth;          /* pixel depth in bits: IPL_DEPTH_8U, IPL_DEPTH_8S, IPL_DEPTH_16S,
                            IPL_DEPTH_32S, IPL_DEPTH_32F and IPL_DEPTH_64F are supported */
    char colorModel[4]; /* ignored by OpenCV */
    char channelSeq[4]; /* ditto */
    int  dataOrder;      /* 0 - interleaved color channels, 1 - separate color channels.
                            cvCreateImage can only create interleaved images */
    int  origin;         /* 0 - top-left origin,
                            1 - bottom-left origin (Windows bitmaps style) */
    int  align;          /* Alignment of image rows (4 or 8).
                            OpenCV ignores it and uses widthStep instead */
    int  width;          /* image width in pixels */
    int  height;         /* image height in pixels */
    struct _IplROI *roi;/* image ROI. if NULL, the whole image is selected */
    struct _IplImage *maskROI; /* must be NULL */
    void  *imageId;      /* ditto */
    struct _IplTileInfo *tileInfo; /* ditto */
    int  imageSize;      /* image data size in bytes
                            (==image->height*image->widthStep
                            in case of interleaved data)*/
    char *imageData;  /* pointer to aligned image data */
    int  widthStep;   /* size of aligned image row in bytes */
    int  BorderMode[4]; /* ignored by OpenCV */
    int  BorderConst[4]; /* ditto */
    char *imageDataOrigin; /* pointer to very origin of image data
                              (not necessarily aligned) -
                              needed for correct deallocation */
}
IplImage;
```

Fig. 4.9 The members of the structure IplImage.

After initializing the information of the input image, we proceed real-time face detection procedure. The detection results of our real-time system are shown in Fig. 4.10. Our face detector can process a 320*240 pixel image in 20~400ms, depending on skin color area. Compared with OpenCV face detector, our detection speed is faster in small skin color area situation, as shown in Table 4.8.

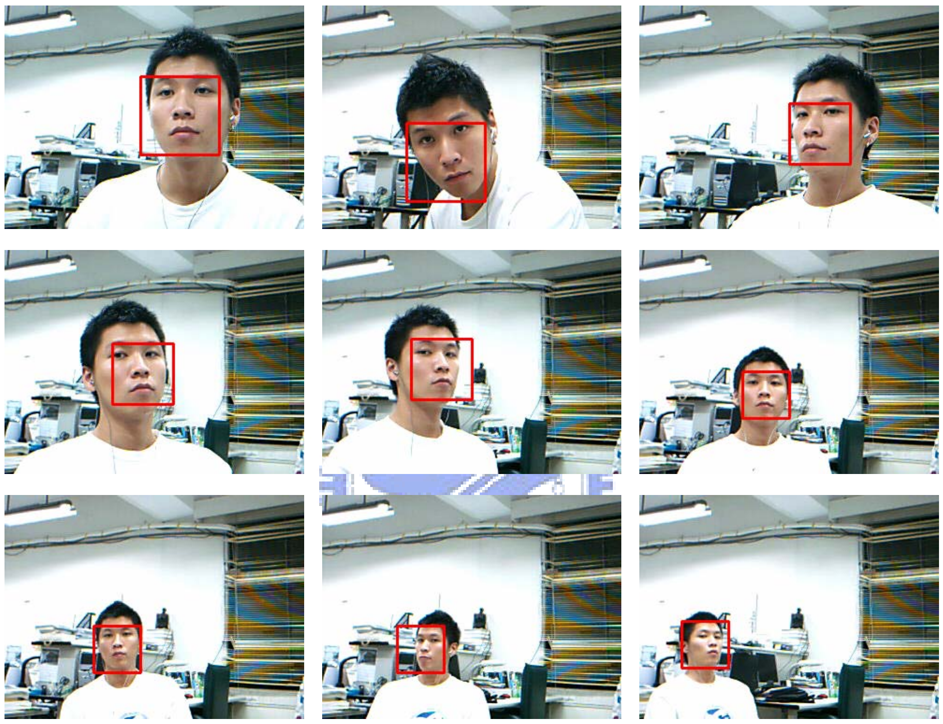


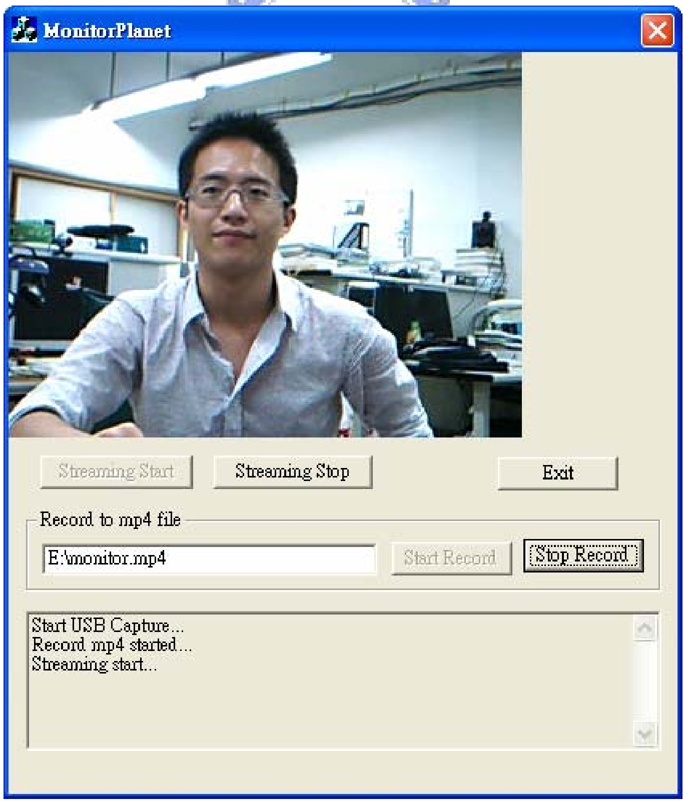
Fig. 4.10 Real-time face detection.

	Detection Rate	**Detection Time**
OpenCV Detector	94.48%	70~85ms
Our Face Detector	88.31%	20~400ms

Table 4.8 Comparison with our real-time face detector and OpenCV real-time face detector. For only single face test, it requires 20~70ms to detect face when the face scale is under 55*55 pixels.

4.4 Implementation of Real-Time Monitoring System

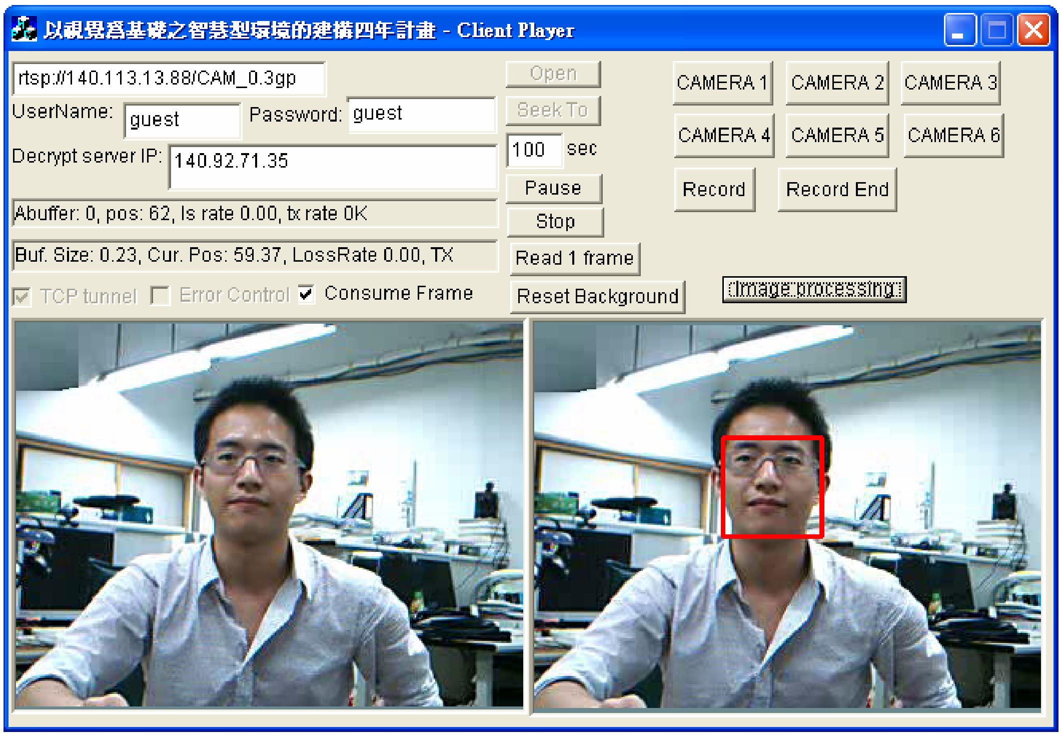
Here, we construct the real-time monitoring system with face detection technique. Fig. 4.11 and Fig. 4.12 show the execution results of server and client. When the server program is started, it also catches the image from USB camera connected to server PC. The streaming service is started by clicking on “Streaming Start” button of sever interface. In Fig. 4.12, the client program is connecting to the server to receive live images by entering IP address of the server and pressing “Open” button of client interface. The right side of client interface is shown that outcome of face detection by pressing “image processing” button.



Record to mp4 file: E:\monitor.mp4

```
Start USB Capture...
Record mp4 started...
Streaming start...
```

Fig. 4.11 Server interface.



rtsp://140.113.13.88/CAM_0.3gp
UserName: guest Password: guest
Decrypt server IP: 140.92.71.35
100 sec
Abuffer: 0, pos: 62, ls rate 0.00, tx rate 0K
Buf. Size: 0.23, Cur. Pos: 59.37, LossRate 0.00, TX
☑ TCP tunnel ☐ Error Control ☑ Consume Frame

Fig. 4.12 Client interface with face detection.

CHAPTER 5
CONCLUSIONS AND FUTURE WORK

In this thesis, we proposed a robust face detection method which have good detection speed and can detect wide face rotation. Experimental results show our system achieve higher detection rate than the system without AdaBoost, the system without using multi-pose face detection, and the system without using skin ratio preprocessing. We also use features selected with and without AdaBoost to test photos. Experimental results prove AdaBoost is so robust that false alarm rate can be reduced efficiently. In term of detection speed, the method with skin ratio preprocessing is much faster than the method without skin ratio preprocessing. Depending on different skin color area over the whole image, it can save 20~95% computational time. Therefore, our system can be widely used in real-time applications.

In the future, we plan to integrate face recognition approach into our system, and try to improve detection rate and speed. It will enhance standard of human living.

References

[1] Face detection technology on digital cameras: http://www.letsgodigital.org/en/14826/face-detection-technology/

[2] R.-L. Hsu, M. Abdel-Mottaleb and A.K. Jain, “Face Detection in color images,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol.24, no.5, pp.696-706, May 2002.

[3] J. Kovac and P. Peer, “Human skin colour clustering for face detection,” *EUROCON 2003. International Conference on Computer as a Tool, Ljubljana, Slovenia,* Sept. 2003.

[4] P. Viola and M. Jones, “Rapid objection using a boosted cascade of simple feature,” *Computer Vision and Pattern Recognition,* vol. 1, pp. 8-14, 2001.

[5] P. Viola and M. Jones, “Fast multi-view face detection,” Tech. Rep. TR2003-96, Mitsubishi Electric Research Laboratories, July 2003.

[6] Y. Wang, H. Ai, B. Wu, and C. Huang, “Real time facial expression recognition with Adaboost,” *ICPR,* 2004.

[7] P. J. Phillips, H. J. Moon, S. A. Rizvi, and P. J. Rauss, “The feret evaluation methodology for face recognition algorithms,” *PAMI,* 22(10). 1090-1104, Oct. 2000.

[8] Y. Freund and R. E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” *Computer Learning Theory: Eurocolt ’95,* pages 23-37, Springer-Verlag, 1995.

[9] CCIR, “Encoding parameters of digital television for studios,” *CCIR Recommendation 601-2, Int. Radio Consult. Committee, Geneva,*

Switzerland, 1990.

[10] D. Chai and A. Bouzerdoum, “A Bayesian Approach to Skin Color Classification in YCbCr Color Space,” *TENCON 2000. Proceedings, IEEE, Kuala Lumpur Malaysia,* Vol. 2, pp. 421-424, Sept. 2000.

[11] Nearest Neighbor Interpolation:
http://www.dpreview.com/learn/?/key=interpolation

[12] Open Source Computer Vision Library (OpenCV):
http://opencvlibrary.sourceforge.net/
http://www.intel.com/technology/computing/opencv/index.htm