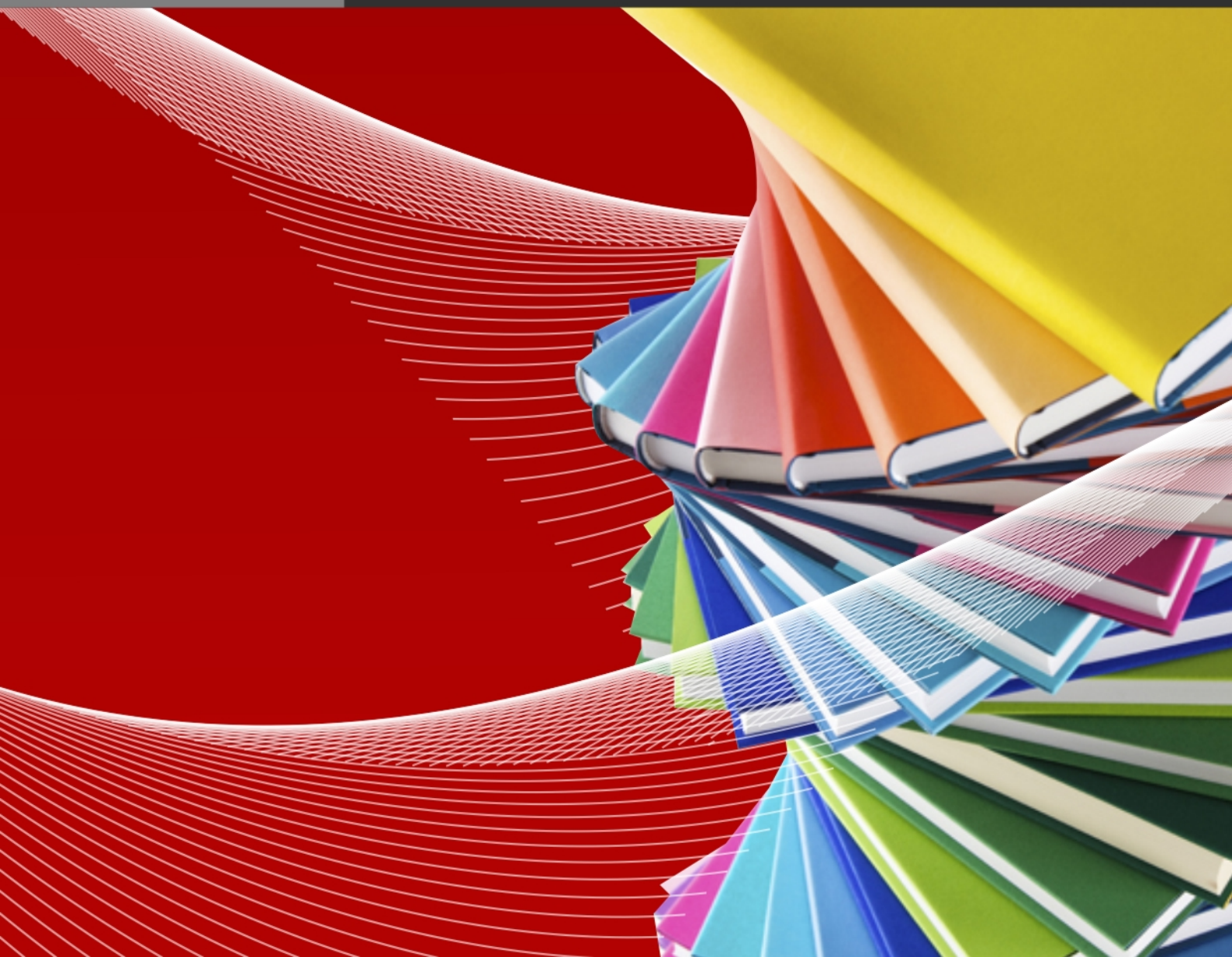
澳門大學
UNIVERSIDADE DE MACAU
UNIVERSITY OF MACAU

Outstanding Academic Papers by Students
學生優秀作品



University of Macau

Department of Computer and Information Science

澳門大學

UNIVERSIDADE DE MACAU

UNIVERSITY OF MACAU

Design of "Hands-Free" human-computer interface

by

Meng Chu Cheong, Student No: D-B0-2768-9

Final Project Report submitted in partial fulfillment
of the requirements of the Degree of
Bachelor of Science

Project Supervisor
Dr. ZHOU Yicong

06 June 2014

Declaration

I sincerely declare that:

1. I and my teammates are the sole authors of this report,
2. All the information contained in this report is certain and correct to the best of my knowledge,
3. I declare that the thesis here submitted is original except for the source materials explicitly acknowledged and that this thesis or parts of this thesis have not been previously submitted for the same degree or for a different degree, and
4. I also acknowledge that I am aware of the Rules on Handling Student Academic Dishonesty and the Regulations of the Student Discipline of the University of Macau.

Signature : _____

Name : Meng Chu Cheong

Student No. : D-B0-2768-9

Date : 06 June 2014

Acknowledgements

I would like to thank my supervisor, Dr. ZHOU Yicong, for his support throughout the development of my thesis.

I also would like to thank my teammate, Wai Kin Ho, for all the efforts and this thesis really would not have been completed without his contribution.

Abstract

Nowadays, electronic devices are everywhere, people normally control electronic devices using hands. However, the traditional interfaces limit the physically disabled people to access. Therefore, we need to develop a new kind of interface which is not only for the normal person, but also for the disabled people. This work presents a video-based "hands-free" human-computer interface designed to offer a friendly user experience and accessibility for everyone, including disabilities. The interface automatically captures the movement of users' eyes with camera and helps them to accomplish some simple tasks with computer by eyes and head rotation. At last, we show the first prototype of the interface which is implemented by Kinect for Windows.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

As so many new technologies have been developed, computers become more important in our lives. We have smart phones, tablets, even smart-houses today. Every day, we need computer to accomplish our works, receive the latest information, entertainment, or communicate to each other. Currently, the most general way to control a computer is through the keyboard and mouse. Moreover, there are some new kinds of interfaces, such as touch screen, become a trend of controlling electronic devices. However, no matter using keyboards, mouses, or touch screens, people interact with computers or electronic devices still with their hands. From a different point of view, most of the designs of interface are assumed that all the users have hands, good fingers and can act normally. They do not consider the disabled people whose hands are injured, paralyzed or even do not have any hand. Especially in nowadays, electronic devices are all around us. If a man can't control these devices, he even is not able to take good care of himself. For that reason, the main purpose of this paper is to develop a more user-friendly "hands-free" computer interface to manipulate the electronic devices, especially for disabilities.

Although it is not easy to design a very nice interface for any people with different demands, there are some interfaces are specially designed for disabilities. For example, "mouth stick" is an interface which can help disabled people to type characters with his mouse instead of hands. "Head wand" is a head-mounted equipment which can provide the functions such as typing, navigating through web documents, etc., by head rotation [11]. However, those equipments have some

limitations and therefore can't fit every disability. For the previous examples, they are not suitable for people who can't move his head, and it requires extra training.

In order to develop a system which can be accessible to everyone, the selection of a common physical device and approach to control computer is the most crucial part. Nowadays, camera is the most common sensor which can be found in most of the electronic devices. We have camera (at least one) attached on our phone, computer and most of the hand-held device. Besides, eyes are the most salient features of human face, we can get different information of others through eye-contact, e.g. thought, needs, cognitive processes, emotion, and even the interpersonal relations [8]. Thus, our interface is designed to capture eye rotation through cameras to manipulate the computer.

In this paper, a system structure for video-based "hands-free" human-computer interaction interface is proposed. This four-layer structure enables our system to capture the environment image as an input, locate the user's eyes and head, detect and analysis their motions, and help user to accomplish simple operations of electronic device. In addition to the system structure, our work mainly focuses on locating user's iris since it's the most difficult part throughout the structure. Once the iris is detected, the user's eye motion capturing becomes much easier.

The first prototype of our system interface is also presented in this paper. This prototype can capture the motions of eye rotation and support the *turning-page* function while user is reading the Power Point.

My contribution of the work is mainly focus on two parts. The first part is to implement the image analysis layer. That is, to find a robust method to locate the user's eye from the input image. It not only needs high accuracy rate and high efficiency, but also can be applied into different application easily, like detect the user's head instead of eye. The second part is to implement the application interface layer, which is mainly focus on the implementation of the interface of our first prototype.

The remainder of this paper is organized as follows. Some related works are introduced in Chapter 2. Chapter 3 presents our proposed system design. Chapter 4 presents the features of the system. The implementation presents in Chapter 5, experimental results in Chapter 6. Chapter 7 includes conclusion and

future works is includes in Chapter 8.

Chapter 2

Related works

In this chapter, some related methods and interfaces are going to be introduced. Section 2.1 presents the learning algorithm called "Adaboost" which is used in our system in order to locate user's eye.

In Section 2.2, different existing eye detection methods are shown. The basic concept of these methods will be introduced and discussed. Moreover, the reason why they are not chosen in our system will also be explained.

In Section 2.3, some user interfaces for disability are presented and explained how they affect our design.

2.1 Adaboost algorithm

Adaboost, short for "Adaptive Boosting" [12], is a machine learning algorithm, using the Harr-like features (Fig. 2.1), to detect objects.

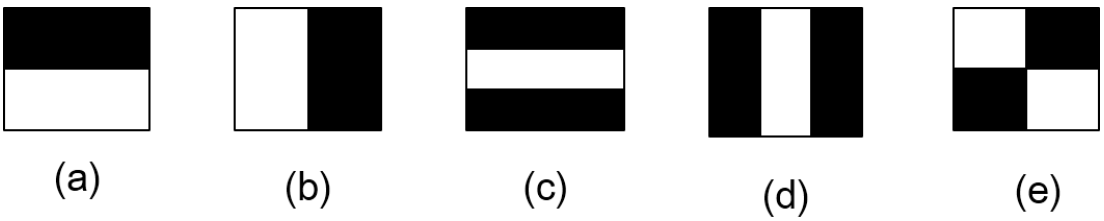


Figure 2.1: Features are used in cascaded Adaboost.

Well-trained of the cascaded Adaboost is needed before detection. Firstly, massive examples (including positive and negative examples) are input into the training program. It calculates the feature values and generates some weak classifiers. Then, it forms a strong classifier by combining the weak classifiers linearly. The strong classifier classifies out the target by scanning the image in different scales. One of the advantages of Adaboost algorithm is that it provides high accuracy rate. More details of Adaboost algorithm will be discussed in Section 3.1.1.

2.2 Other existing eyes detection methods

D. Sidibe et al. [2] proposed a skin color model to locate user's head and detect their eye by fitting an ellipse to the potential eye regions. The method is simple and efficient. However, it requires a less complexity background and it can't be used for different color of people.

M. Hassaballah et al. [5] proposed another eye detection method with iris detection at the same time. It calculates the entropy of the gray intensity on the facial images and the regions of eye can be indicated by high values of entropy. However, it requires the pre-knowledge of the user's face and it is very sensitive to the noise.

Hironobu Takano et al. [3] proposed an eye detection method with particle filter. This method is able to detect user's eye accurately according to its experimental results. However, the results also show that the computation of method is a little bit too complex.

To summarize, most of the eye detection methods are color-based. They might occur some problems like easy influenced by the background, sensitive with noise, computational complexity or not suitable for people with different color. Some methods need the pre-knowledge of the user's face.

2.3 User interface for disability

There are many researches focus on the design of user interface for disabilities. Some of them need to wearing extra device and some need special sensor, like the infrared emitter and tracker, in order to detect user's motions.

The "tongue-computer interface" is proposed by Lotte N.S. Andreasen Struijk [7]. It's using a palatal plate which is placed on mould of the upper part of the mouth and an activation unit glued to the tongue as input device. This system can help user to type characters with moving his tongue. However, it needs extra sensor and the user needs to put a special device into his mouth. Also, tongue movement is not so natural for human to express their thought and command.

Jonathan Lombardi [4] developed another interface for controlling the mouse click. The interface tracks the user's eyebrows with a camera. When user raises his eyebrows, the interface will activate the left-click event of the mouse. It's a very innovative design to control mouse-click since using eyelids may cause problem due to blinking when people are interacting with computer.

Finger counter [1] is a camera-based interface which is able to recognize the number of fingers user held up. It is using the background differences and edge detection to locate user's hand. After that, the system extracts the feature of hand by processing the pixels with polar coordinate and analysis the state of fingers. Under multiple testing, this system can work under different lightening and background. Although it is very promising under different environment, it hasn't included the computer users with physical disabilities.

Chapter 3

System design

Our proposed system structure is shown in Fig. 3.1. It is composed of four main parts: **sensor input layer**, **image analysis layer**, **motion detection layer** and **application interface layer**.

Sensor input layer captures the depth and color image from the environment through Kinect sensor. In image analysis layer, the cascaded Adaboost algorithm is used to locate the position of eyes of user from the color input image. Motion detection layer mainly detects user's motions according to the located eyes and the depth image captured from the first layer, which makes use of the iris detection method. The fourth layer, the application interface layer, triggers the corresponding pre-defined events and response to user according to the motions the system detected.

About the work distribution, I mainly focus on the second and fourth layer, so in this report, it only contains these two parts. For my groupmate Kin, who focuses on the first and third layer. For more details about these two layers can be referred to his report.

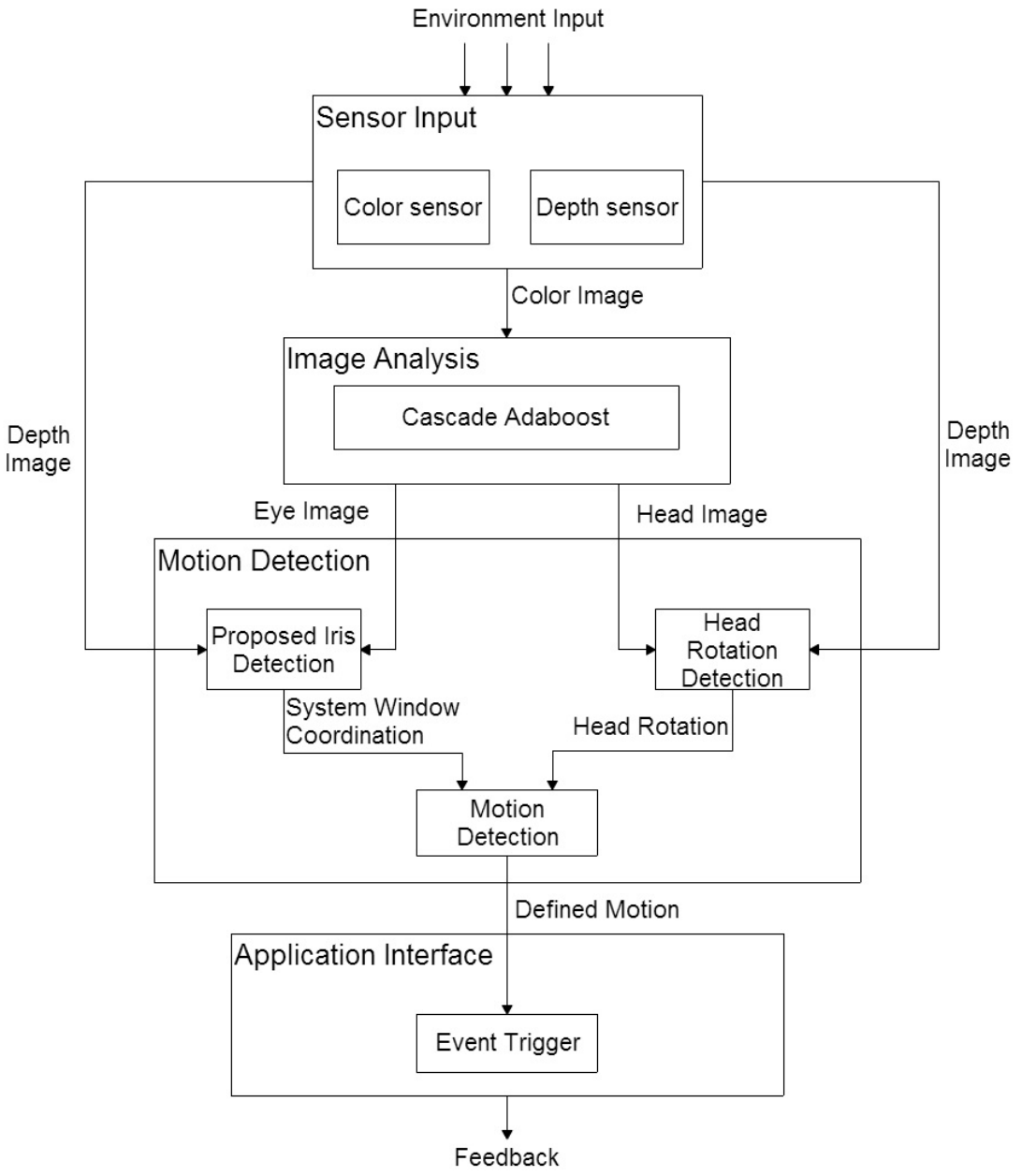


Figure 3.1: The proposed system structure.

3.1 Image analysis layer

The second layer of the structure is the image analysis layer. This layer is used for locating the position of eyes and head from the color image. The method using here is proposed by Paul Viola and Michael J. Jones [10], known as the **cascaded Adaboost**. It has the advantages such as high accuracy rate, efficient, independent from the color of user, does not required the pre-knowledge of the facial image, and target independent. The cascaded Adaboost is not only designed for eye detection, but also for any other objects. It enables the scalability and compatibility of this layer as well as the system, because it only needs a little effort to change or enlarge the function of this layer.

3.1.1 Cascaded Adaboost algorithm

The basic idea of cascaded Adaboost is divided into two parts: **learning** and **classification**. Before the learning phase(Fig. 3.2), a large number of images are needed. It contains positive samples(with target) and negative samples(without target). Those samples are input into the training program, and it concludes the target features and some corresponding data. Then, in classification phase, the images with target(s) are input into the classification algorithm and it generates the output image with the detected location of the target.

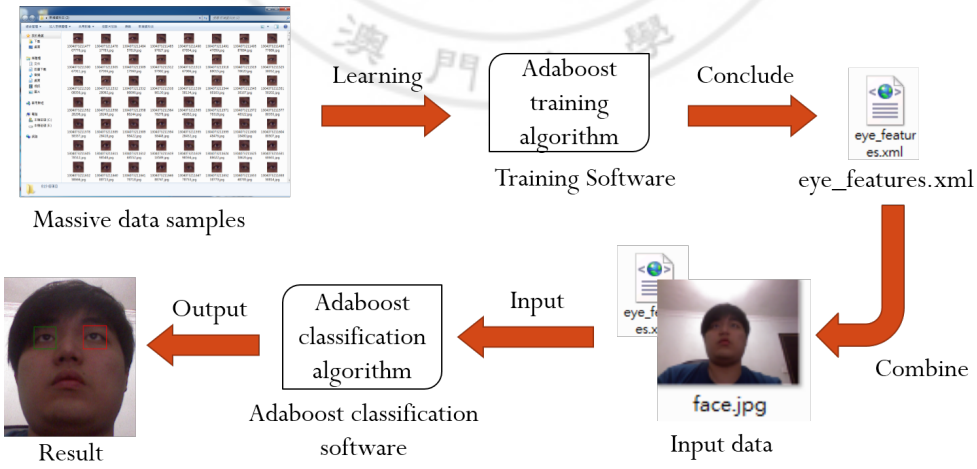


Figure 3.2: The basic flow of Adaboost algorithm.

A *strong classifier* is used in Adaboost algorithm to detect user's eyes. It comprises several weak classifiers and each weak classifier is formed with three parameters: **feature**, **threshold** and **polarity**. The classifier is so called "weak" because each of them is not expected to be very accurate. Therefore, several weak classifiers are combined linearly in order to complement each other, that is the strong classifier.

A feature is formed by one of the components shown in Fig. 2.1 with different sizes and positions. To enumerate all of them, according to the Fig. 3.3, the image is scanned with the component of feature from left to right, top to bottom. After each turn of scanning, the component is enlarged horizontally or vertically, and scan the image again until the the component is covered the whole image.

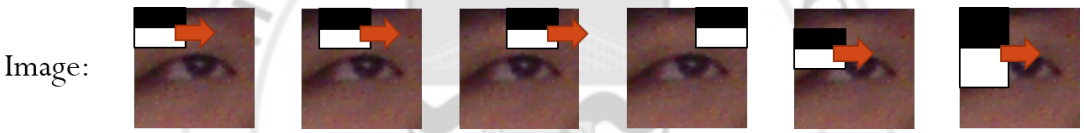


Figure 3.3: The process of an image is scanned by one of the features.

To calculate the feature value (Fig. 3.4), the corresponding feature is covered on the image. Since the inputs are color images form the environment, it is necessary to convert into gray-level. Every feature consists of white and black region, and the covered pixels are summed separately corresponding to different color. Finally, the feature value is obtained by calculating the differences between two summing up values.

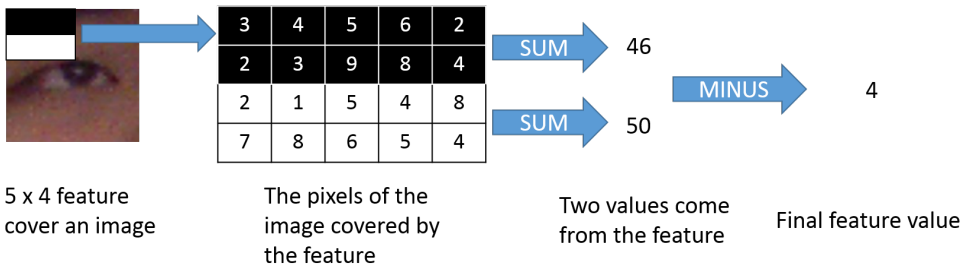


Figure 3.4: The process of calculating the feature value.

Algorithm 1 and Algorithm 2 show the pseudo-code to enumerate all the features. Note that each feature is represented by the points of its left-top corner and right-bottom corner.

Algorithm 1 Enumerate the features(1)

```
Enumerate all two horizontally stacked
for (v = 1; v ≤ ExamplesHeight; v++) do
    for (h = 2; h ≤ ExamplesWidth; h+ = 2) do
        for (i = v; i ≤ ExampleHeight; i++) do
            for (j = h; j ≤ ExampleWdith; j++) do
                Save(i − v, j − h, i, j)
            end for
        end for
    end for
end for

Enumerate all two vertically stacked features
for (v = 2; v ≤ ExamplesHeight; v+ = 2) do
    for (h = 1; h ≤ ExamplesWidth; h++) do
        for (i = v; i ≤ ExampleHeight; i++) do
            for (j = h; j ≤ ExampleWdith; j++) do
                Save(i − v, j − h, i, j)
            end for
        end for
    end for
end for

Enumerate all three horizontally stacked features
for (v = 1; v ≤ ExamplesHeight; v++) do
    for (h = 3; h ≤ ExamplesWidth; h+ = 3) do
        for (i = v; i ≤ ExampleHeight; i++) do
            for (j = h; j ≤ ExampleWdith; j++) do
                Save(i − v, j − h, i, j)
            end for
        end for
    end for
end for
```

Algorithm 2 Enumerate the features(2)

```
Enumerate all three vertically stacked features
for (v = 3; v ≤ ExamplesHeight; v+ = 3) do
    for (h = 1; h ≤ ExamplesWidth; h + +) do
        for (i = v; i ≤ ExampleHeight; i + +) do
            for (j = h; j ≤ ExampleWdith; j + +) do
                Save(i − v, j − h, i, j)
            end for
        end for
    end for
end for

Enumerate all four stacked features
for (s = 2; s ≤ ExamplesWidth; s+ = 2) do
    for (i = s; i ≤ ExamplesHeight; i + +) do
        for (j = s; j ≤ ExampleWidth; j + +) do
            Save(i − s, j − s, i, j)
        end for
    end for
end for
```

The total number of the features are generated by a sample with the size of 24×24 is 162,336. The number of different features is listed below:

- Two horizontally stacked features (Fig. 2.1(a)): $43,200$
- Two vertically stacked features (Fig. 2.1(b)): $43,200$
- Three horizontally stacked features (Fig. 2.1(c)): $27,600$
- Three vertically stacked features (Fig. 2.1(d)): $27,600$
- Four stacked features (Fig. 2.1(e)): $20,736$

In order to calculate such large amount of features, the "integral image" is used. It accelerates the whole process effectively and makes the cascaded Adaboost possible to be applied to detect the target in real-time video stream. Refer to Fig. 3.5, the point A on the integral image is the sum of the pixels to its left and above it. Then, the sum of the region D can be obtained by accessing four points on its corners.

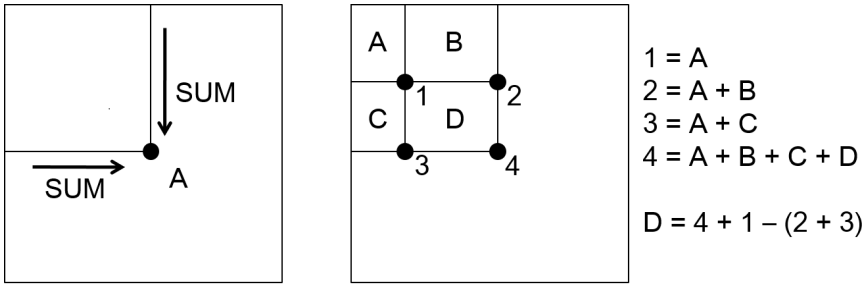


Figure 3.5: Integral image

The mathematical representation of the integral image is:

$$ii(x,y) = \sum_{x^{'} \leq x, y^{'} \leq y} i(x^{'}, y^{'}) \tag{3.1}$$

where ii is the integral image and i is the original image.

And it can be calculated by two steps:

$$s(x,y) = s(x,y-1) + i(x,y) \tag{3.2}$$

$$ii(x,y) = ii(x-1,y) + s(x,y) \tag{3.3}$$

The threshold of a weak classifier is a critical value to classify the target and non-target, and the polarity indicates that whether the feature value smaller or larger than threshold is target. To select the best threshold of the weak classifier, the error rate of each possible threshold is evaluated. The possible value of threshold is one of the feature values generated by applying the feature of the weak classifier to each input samples. The error rate of each threshold is the summation of the *weight* of each misclassify image. In order to simplify the explanation, the *weight* of each image is assumed to 1. Surely, the least amount of error rate indicates the best threshold. The description of the procedure is presented as follow:

- Assume that there are 6 input samples, A to F. The feature value of each sample is evaluated and listed in Table 3.1. The sign of " $+$ " indicates the positive samples and " $-$ " indicates negative. The table is sorted by the

Table 3.1: Error of each threshold and polarity.

Image	Sign	Feature values	Error for polarity "≤"	Error for polarity ">"
A	+	-10	2	4
B	+	10	1	5
C	-	52	2	4
D	+	65	1	5
E	-	73	1	4
F	-	90	3	3

feature values with an increasing order.

- Suppose that the threshold is -10, and the polarity is " $\leq$ ", which means that the image with a feature value smaller than or equal to the threshold is considered as target.
- Image B and D is misclassified since they are positive but classified as negative. Assume that the weight of each image is 1, then the error rate is 2 is this case.
- Note that each threshold has two error rates, since it has two kinds of polarity.
- The error rate of each threshold is also listed in Table 3.1. According to the table, the threshold and polarity with the smallest error rate can be selected.

After the threshold and polarity is selected, the error rate of each weak classifier due to the selected parameters is also obtained. Then, the best classifier can be found by comparing the error rate between each classifier.

All the weights of the samples were assumed to be 1 before. In fact, at the very beginning of the algorithm, the weight of each sample is assigned to $1/m$ if the sample is positive, or $1/n$ otherwise, where m, n is the total amount of positive samples and negative samples correspondingly. Before each turn of the training, all the weights are normalized by divided each weight to the total sum of weights. After the training, the weights are updated by the function which is presented in the following:

$$w_{t+1,i} = w_{t,i}\beta_t^{1-e_i} \tag{3.4}$$

where $e_i = 0$ if the sample corresponding to the weight is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$. ϵ_t is the summation of the weights of the misclassified samples. Note that the weight of the correct classification of sample isn't updated since $\beta_t^0 = 1$. The update function tries to amplify the error of the weak classifier generated in this turn in order to find a weak classifier which can be complemented with the former classifiers.

The mathematical representation of the error rate of each threshold is:

$$e_T = min(S^+ + (T^- - S^-), S^- + (T^+ - S^+)) \tag{3.5}$$

where S^+ and S^- is the summation of weight of positive and negative images with feature values larger than the threshold T correspondingly. T^+ and T^- are the total sum of weight of positive and negative images correspondingly. $S^+ + (T^- - S^-)$ is the error rate of the polarity "$\leq$" and $S^- + (T^+ - S^+)$ is the error rate of the polarity ">". Therefore, this formula also indicates the polarity of each threshold.

After several turns of training, weak classifiers are combined together linearly to form a strong classifier in order to detect the object more correctly and efficiently. The details of the training algorithm is presented as follows:

- Given sample images $(x_1, y_1), ..., (x_n, y_n)$ where $y_i = 0$ for negative samples, and $y_i = 1$ for positive samples.
- Assign the weights of each samples i:

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

  where m, l are the number of negative and positive samples correspondingly.

- For $t = 1, ..., T$:

– Normalize each weight of the samples i, $w_{t,i} \leftarrow \frac{w_{t,i}}{Z_t}$, where Z_t is the total sum of the weights.

– Select the weak classifier, denoted as $h_t(x)$, with the minimum weighted error:

$$\epsilon_t = \min_{f,\theta,p} \sum_i w_i |h(x_i, f, \theta, p) - y_i|$$

where f, θ, p are feature, threshold and polarity correspondingly.

– Update the weights of misclassified sample images i:

$$w_{t+1,i} = w_{t,i} \frac{\epsilon_t}{1-\epsilon_t}$$

- Combine the weak classifiers linearly and form the strong classifier $C(x)$:

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

where $\alpha_t = \log \frac{1-\epsilon_t}{\epsilon_t}$

In classification phase, the image is scanned in different scales in order to detect the object in different sizes. The cascaded Adaboost detects an object through a voting process. An object is a positive result when it is detected as positive by more than half of the weak classifiers in the strong classifier. In order to speed up the classification process, the strong classifiers are cascaded together in order to eliminate the non-target object at earlier time. The algorithm of training the cascaded Adaboost is as follows [9]:

- Given the values of the maximum false-positive rate per layer f, the minimum detection rate per layer d, and the overall false-positive rate F
- P = set of positive samples; N = set of negative samples
- $F_0 = 1.0; D_0 = 1.0; i = 0$
- while $F_0 > F$

- $i \leftarrow i + 1$
- while $F_i > f \times F_{i-1}$
  - $n_i \leftarrow n_i + 1$
  - Train a classifier $h_i(x)$ which is consist of n_i features with the samples in P and N
  - Cascade the $h_i(x)$ with the cascaded classifier
  - Evaluate the cascaded classifier on the samples detected as positive to determine F_i and D_i
  - While $D_i < d \times D_{i-1}$, decrease threshold for $h_i(x)$. This might affects F_i
- If $F_i > F$
  - Evaluate the cascaded classifier on the set N.
  - N = false detects samples .

3.1.2 Post-processing of cascaded Adaboost

In order to improve the accuracy of the detection, the post-processing method is proposed. Although the original method performs satisfactory, either the false-positive or false-negative rate is still a room for improvement. In fact, the false-positive detection rate can be lowered by eliminating some detected regions that couldn't be eyes. Since the false-positive detection is usually caused by our mouse and nose (Fig. 3.6), it can be assumed that the highest detected region, denoted as r, is one of the user's eyes. Then, the second highest region, denoted as r^*, is found. If these two regions, r and r^*, are too closed to each other or the angle is over certain degree, then the r^* is eliminated from the image, and do the process again until there is no more r^* is found. Otherwise, r and r^* are output as the result of the detection. Some qualitative results are shown in Fig. 3.7 and the flow chart of the algorithm is presents in Fig. 3.8.

Figure 3.6: Wrong detection causes by nose.

Figure 3.7: The comparison between wrong detection without post-processing and correct detection with post-processing.

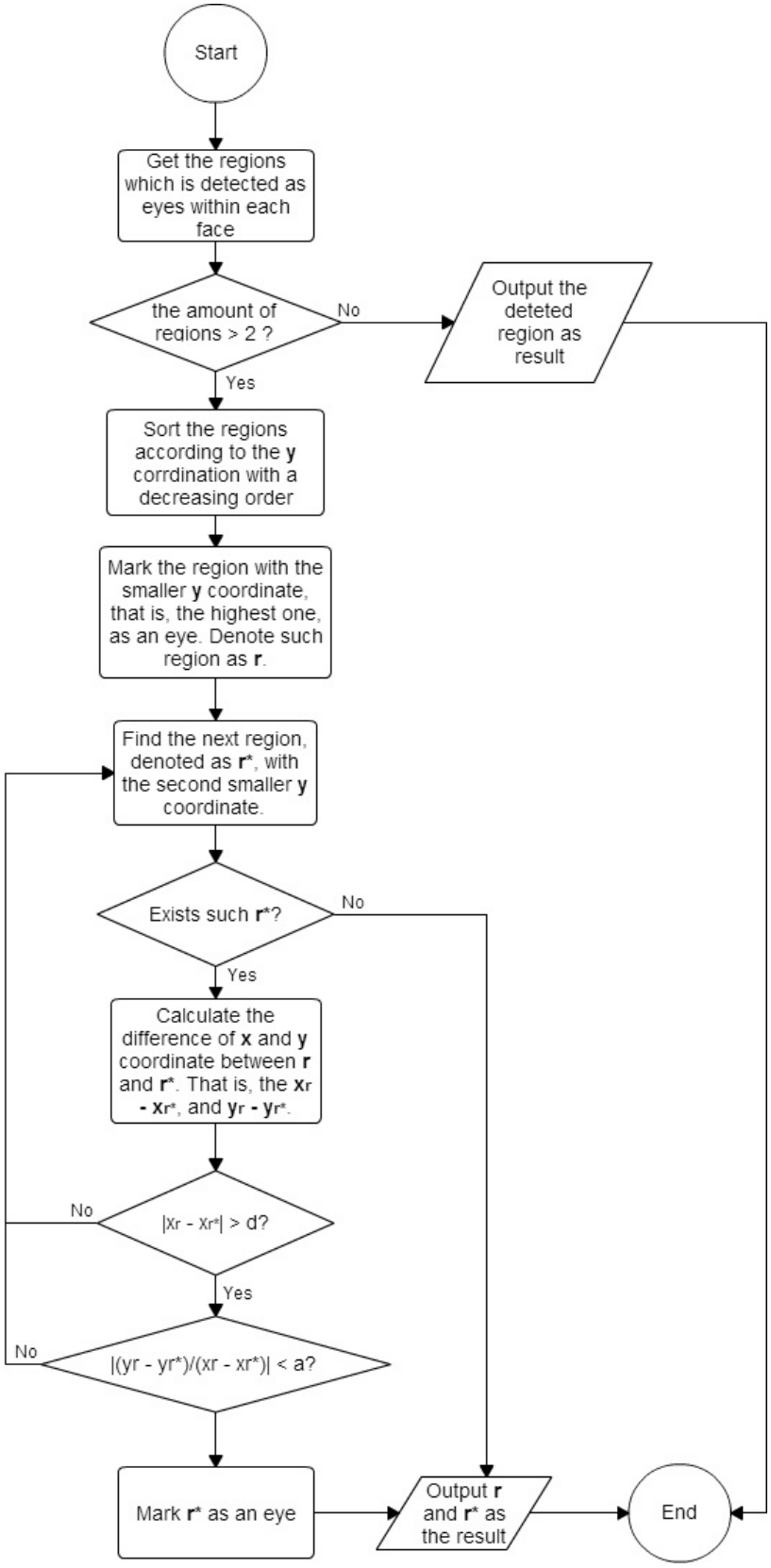


Figure 3.8: Flow chart of post-processing.

3.2 Application interface layer

The application interface layer lies between user to system and system to computer. Some events for manipulating the computer are defined in this layer. The developer can implement his own event handler to specify the function when the event is triggered. Such design enables system be expanded easily to support various motions in the future. The former layer sends the detected information, for example, the position of user's gaze, to the application interface layer. Then, it gives the feedback to user corresponding to the event handler if the motion of user fires the trigger of any event. For example, if the system detects that user stared at an icon of document for few seconds, it opens the file as a response. This layer also enables the system being compatible to different application. For example, stereoscopic display can be implemented by defining an event handler to show a scene in different angle according to user's gaze.

Chapter 4

System features

Our system help user to accomplish some simple tasks with computer through eyes or head rotation. It automatically tracks the motion of user's eyes and head. Furthermore, it is independent from any device, platform and language. It does not require any specific except two cameras or a camera with a infrared sensor. It is easy to expand the system in order to support various motion manipulations. It can also be applied into different application, e.g. stereoscopic display, after some small modification.

Chapter 5

Implementation

5.1 Image analysis layer

In order to locate user's eyes, it is necessary to implement the training program and the classification algorithm of cascacded Adaboost. But in fact, the OpenCV [6] provides the functions of cascaded Adaboost. Some built-in classifiers for face and eye are already existed in the library and each classifier is defined in an XML file. Therefore we can directly invoke the library to train our classifier or to detect the target. Some parameters of the function provided by OpenCV are needed to specify: **scale factor** and **minimum neighbor**. The **scale factor** refers to the ratio of scaling up the image gradually during the classification. The **minimum neighbor** is, in case of multiple times of detection of the same object, and in order to lower the false-positive rate, the function automatically groups up the detected rectangles and rejects the group which the number of rectangles which is smaller than the **minimum neighbor**. The result of cascaded Adaboost is shown in Fig. 5.1. It shows false-positive error rate, false-negative error rate, and accuracy corresponding to each **minimum neighbor** with a scale factor 1.2. According to the figure, with larger **minimum neighbor**, the false-positive rate falls but the false-negative rate rises. The best result is 82% for accuracy with 42 of **minimum neighbor**.

The selected value of **scale factor** and **minimum neighbor** in the cascaded Adaboost with our proposed post-processing is shown in Fig. 5.2. It presents the

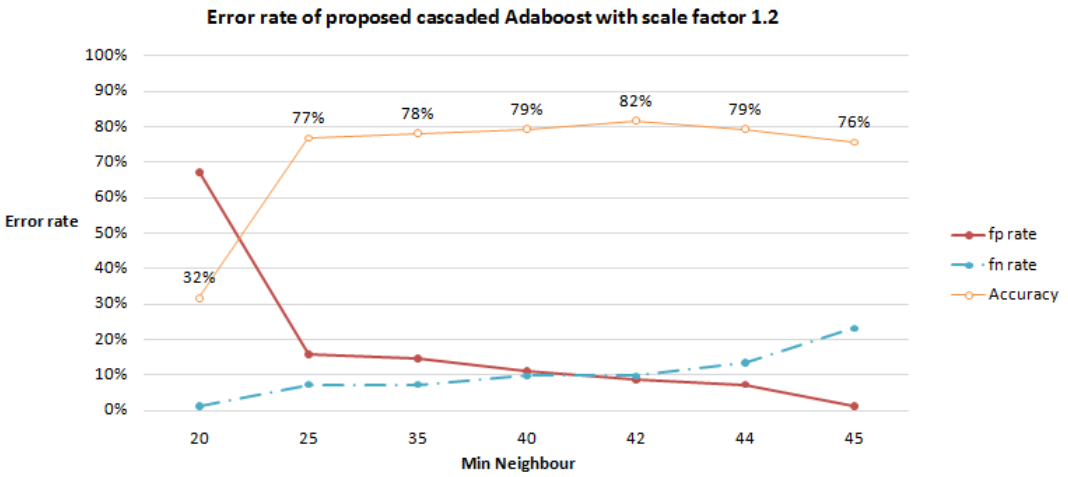


Figure 5.1: False-positive, false-negative and accuracy rate with scale factor 1.2

accuracy and processing time corresponding to each parameters. The relative difference between the processing time and accuracy also has been shown, which is the yellow bar in the figure. Furthermore, in order to compare each parameter easily, the relative difference has been quantified and shown in the Fig. 5.3. Note that the numbers shown in Fig. 5.3 are only for comparing between each parameters and do not indicate any other information. According to the Fig. 5.2, although the lower scale factor and minimum neighbor indicates higher accuracy, it also indicates higher processing time. To balance these two factors, a parameter which has the largest value of the relative difference is selected, that is, the longest of the yellow bar. According to Fig. 5.3, the best result is indicated by the scale factor of 1.2 and the minimum neighbor of 10.

Besides, the speed of classification of cascading Adaboost can be improved by constrain the window size of the detection. The experimental results show that the cascaded Adaboost can detect the user's eyes around a distance 40 to 70 cm from the sensor to the user with a minimum of window size of target is 30×30 and the maximum 35×35.

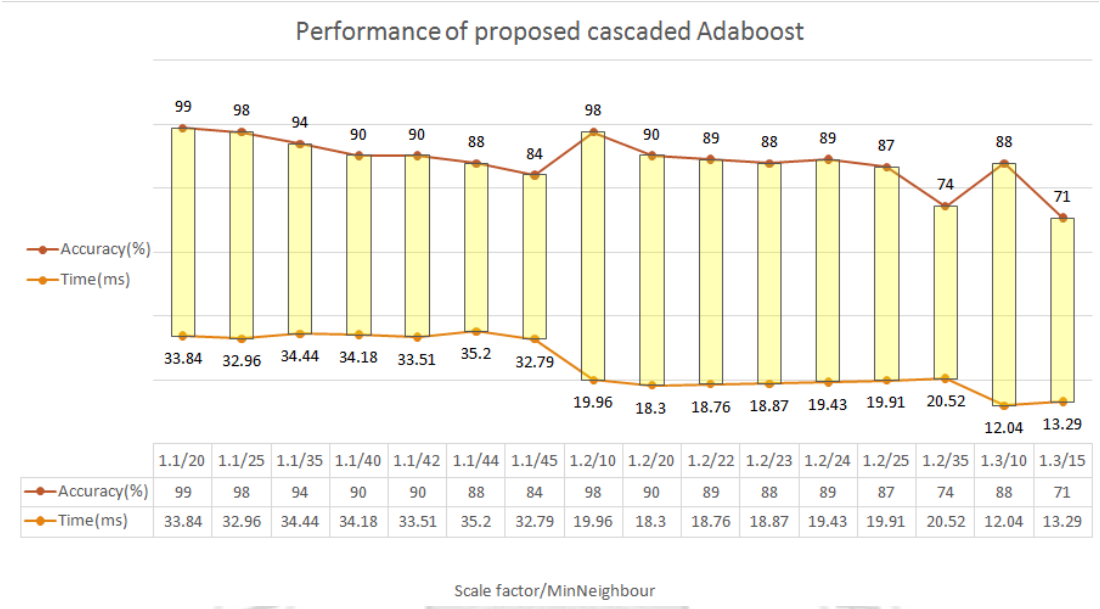


Figure 5.2: The accuracy and the processing time of the cascaded Adaboost with the proposed post-processing.

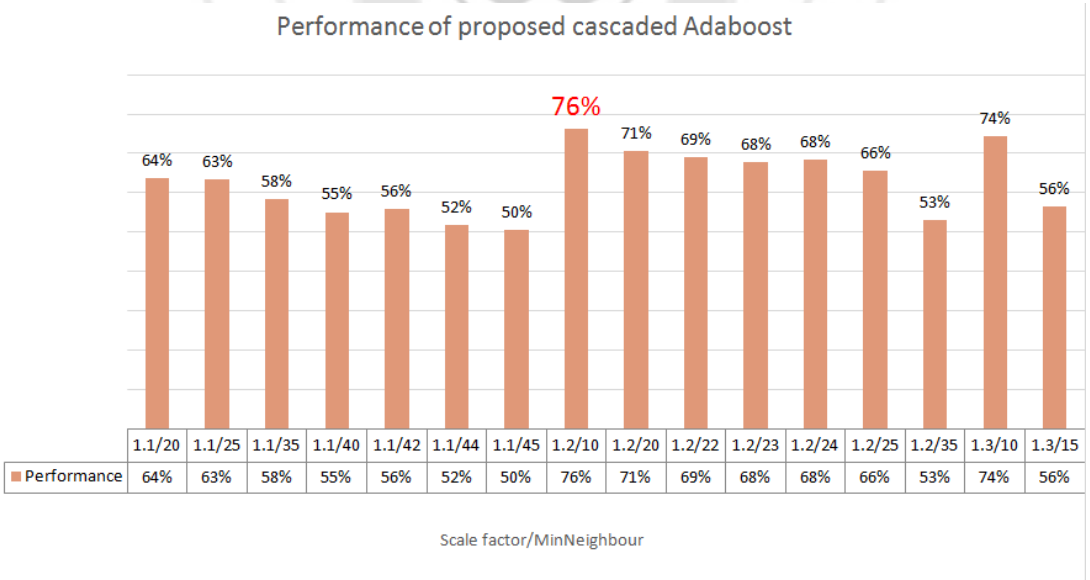


Figure 5.3: The performance of the cascaded Adaboost with the proposed post-processing.

5.2 Application interface layer

The main mission of application interface layer is to define a method for system to give response to user. There are some event handlers can be defined by this layer in order to specify the approach for manipulating the computer.

Another issue the application interface layer takes charge of is, it integrates all the other layers. All the details about the data flow between each layer are totally hidden in this layer. It only opens for some methods for developer to register the event handler and enables the feature of system which can be applied into different application by registering different event handlers and the developer doesn't need to take care about the data flow of each layer.

Furthermore, at the current state, the interface of the first prototype of our system is also integrated into this layer temporally, as well as the implementation of the function of eyes event handler.

In the first prototype of our system, there are two events are defined: looking at left side and right side. In order to define the trigger of these two events base on the user's eyes, two phases are implemented.

The first phase is the *calibration phase*. In this phase, user is asked for looking at the both left side and right side in order to obtain the initial position of user's irises (Fig. 5.4). During the calibration, two circles are shown on either sides in order to instruct user. Also, there is a message shows to the user about the current state of the calibration. Since it may fails in calibration, the system will ask the user to re-calibration again immediately and therefore only the side which is failed need to be redone. More details of the concept and implementation of calibration can be referred to my teammate report.

The second phase is the triggering phase. After the calibration, the motion detection layer is able to detect the user's motions and sends the results to this layer. After receiving the probability of each model of event, the event handler will be triggered by this layer. A flag parameter indicates which event is trigger, that is, left, right, or neither, is passed into the event handler.

To implement the function of checking if the user is looking at the border for three second, a timer is created to count the time. When one of the events is triggered, the timer is started the counting. After it counts a certain time, the

Figure 5.4: The process of calibration.

eyes event handler is invoked. The timer is reset when the same event is no more triggered or different events are triggered. However, since there may have some unstable results due to noise, it's about 0.55 second of buffer time is used in order to smooth the unstable state of the results. The timer isn't reset until the same event no more triggered for 0.55 second.

The eyes event handler makes the system can be applied into different application easily, the function of eyes event handler can be customized by developers. The function implemented in our system is to simulate the keyboard events of the left-arrow and right-arrow button in order to turn pages of the Power Point document.

Besides, in order to display the video stream in the WPF application, it is necessary to convert the Emgu.CV.Image to System.Window.Media.Imaging.BitmapSource. There is one thing should be noted. There is a memory leaking in the conversion and therefore the objects are created required to be freed manually. The codes of the conversion is shown in List. 5.1.

Listing 5.1: Code of conversion

```
[System.Runtime.InteropServices.DllImport("gdi32.dll")]
static extern int DeleteObject(IntPtr o);

public BitmapSource GetBitmapSource(Image<Bgr, Byte> image)
{
    IntPtr intPtr = image.Bitmap.GetHbitmap();

    BitmapSource bitmapSource =
        System.Windows.Interop.Imaging.CreateBitmapSourceFromHBitmap
    (
        intPtr,
        IntPtr.Zero,
        Int32Rect.Empty,
        BitmapSizeOptions.FromEmptyOptions()
    );

    DeleteObject(intPtr);

    return bitmapSource;
}
```

Chapter 6

Experimental results

In this session, the results are discussed into two parts. In the first part, the performance of cascaded Adaboost is shown. It compares the cascaded Adaboost with and without the proposed post-processing by evaluating the results generated from 82 head images with different angle and distance which are captured from Kinect sensor.

In the second part, it presents the first prototype of our system.

6.1 Result of cascaded Adaboost

In this section, we compare the results of cascaded Adaboost with and without our proposed post-processing Table 6.1. The comparison of the accuracy rate is shown in Fig. 6.1. The quantitative results without our proposed post-processing are shown in the Table 6.2 . The best result is 90% which is generated by the parameters 1.1 of scale factor and 35 of minimum neighbor. However, it needs extra processing time to obtain such a result. If the processing time is taken into the consideration, the best result is only 82%, which is generated by the parameters 1.2 of scale fact and 24 of minimum neighbor. Therefore, it's still a room for improvement. With the proposed post-processing method, which quantitative results are shown in Table . The proposed method improve the false-positive rate very much. The best result is 99% which is generated by the parameters 1.1 of scale fact and 20 of minimum neighbor. Take the processing

time in consideration, it still has 98% of accuracy rate, which is generated by the parameters 1.2 of scale factor and 10 of minimum neighbor. It is interesting that it turns the worst into the best. Since the worst result is caused by over-sensitive detection, after eliminated the false-positive detection, it becomes the best.

Besides, although the processing time of parameters with 1.1 of scale fact and 20 of minimum neighbor is a little bit long, it is not unendurable. In fact, a computer with Intel i5 760 CPU and 4GB ram, it can perform very smoothly with those parameters.

Finally, there are some results of the cascaded Adaboost eye detection shown in Fig. 6.2. The results show the detection performs efficient and effective, since no matter what direction the user face to, or even one eye is coved by hand, the cascaded Adaboost detection can detect the user eyes successfully.

Scale factor/ Min neighbour	False positive	False negative	Time	FP rate	FN rate	Accuracy
1.1/20	37	1	33.84	45%	1%	54%
1.1/25	15	1	32.96	18%	1%	80%
1.1/35	1	7	34.44	1%	9%	90%
1.1/40	1	8	34.18	1%	10%	89%
1.1/42	1	9	33.51	1%	11%	88%
1.1/44	1	11	35.2	1%	13%	85%
1.1/45	0	11	32.79	0%	13%	87%
1.2/10	55	1	19.96	67%	1%	32%
1.2/20	13	6	18.3	16%	7%	77%
1.2/22	12	6	18.76	15%	7%	78%
1.2/23	9	8	18.87	11%	10%	79%
1.2/24	7	8	19.43	9%	10%	82%
1.2/25	6	11	19.91	7%	13%	79%
1.2/35	1	19	20.52	1%	23%	76%

Table 6.1: Experimental results of the method without the post-processing

Scale factor/ Min neighbour	False positive	False negative	Time	FP rate	FN rate	Accuracy
1.1/20	0	1	33.84	0%	1%	99%
1.1/25	0	2	32.96	0%	2%	98%
1.1/35	0	5	34.44	0%	6%	94%
1.1/40	0	8	34.18	0%	10%	90%
1.1/42	0	8	33.51	0%	10%	90%
1.1/44	0	10	35.2	0%	12%	88%
1.1/45	0	13	32.79	0%	16%	84%
1.2/10	1	1	19.96	1%	1%	98%
1.2/20	2	6	18.3	2%	7%	90%
1.2/22	1	8	18.76	1%	10%	89%
1.2/23	1	9	18.87	1%	11%	88%
1.2/24	0	9	19.43	0%	11%	89%
1.2/25	0	11	19.91	0%	13%	87%
1.2/35	0	21	20.52	0%	26%	74%
1.3/10	0	10	12.04	0%	12%	88%
1.3/15	1	23	13.29	1%	28%	71%

Table 6.2: Experimental results of the proposed post-processing

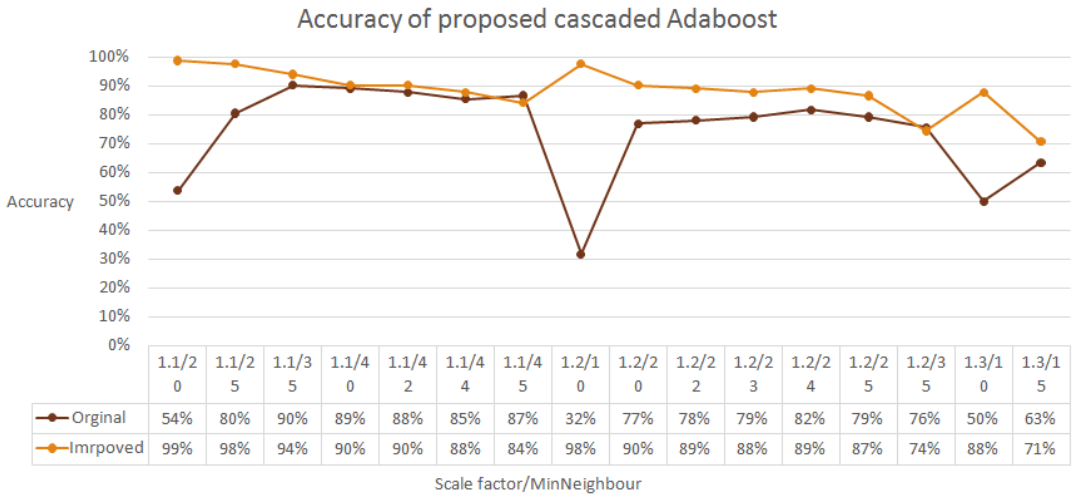


Figure 6.1: The comparison of the cascaded Adaboost with and without the post-processing.

Figure 6.2: Results of the cascaded Adaboost detection.

6.2 Prototype of the interface

The system is running on the computer with the Intel Core i5-760, 4 GB RAM. About the sensor device, our system uses the Kinect v1 in the implementation. The resolution of the camera is set to 640 × 480 with fps 15 with YUV format.

The system is a WPF application which is implemented in C♯. The libraries it uses including the Kinect for Windows SDK v1.8 and the EmguCV v2.9. The EmguCV library is same as the OpenCV library, however, in C♯. The functions the Kinect of Windows SDK provide to our system are the color image capturing and the depth image capturing, and the functions of the basic image processing such as convolution, histogram equalization, and gray image conversion are provided by EmguCV library. It also provides the function of cascaded Adaboost.

Fig. 6.3 is the captured screen of our first prototype of the interface. This interface can help user to turn pages while the user are reading the Power Point. The circles on the left and right side indicates the focus point of the user's eyes. Whenever the user stares at the either side for three seconds, the system will automatically help user to turn to the next or the previous page.

Although it hasn't combined the depth image into the motion detection layer to fix the problem of head movement, the result is still promising. The system can correctly detect the user's motions whenever the user stares at the border of the screen.

Figure 6.3: Captured screen of the first prototype. The circle indicates the focus point of the user's eyes. It would not appear when user is looking at the center of the screen.

Chapter 7

Conclusions

This paper presents a "hands-free" human-computer interface by capturing the motions of user's head and eyes in order to control the computer. It includes the design of the system structure and the methodology to implement each layer. The system structure enables the system being implemented in different platform, application and can be easily expanded in order to support various motion capturing.

Besides, the cascaded Adaboost in the image analysis layer with the proposed post-processing method performs very well. The results of the proposed method show a higher accuracy rate compare to the original method. Moreover, the first prototype of our system is also presented in this paper, which can automatically locate user eye's position, turning page of the Power Point document for user without using hands.

In the future, we hope that it could be truth that the "hands-free" interface can further replace the traditional interface and can help more people to control the computer.

Chapter 8

Future work

Currently, we just implemented our system with Kinect. In the future, we'd like our system can be also implemented on any other devices. Therefore, we are going to implement it with only one frontal camera. Besides, in order to fulfill our system and to be more comprehensive, we will try to combine the speech recognition and text-to-speech function into our system. We'd like our system can be implemented in different platform, especially in mobile device. Furthermore, we'd like to expand our improved iris detection algorithm to more application, for example, the naked-eye 3D.

References

[1] Stephen C. Crampton and Margrit Betke. Finger counter: A human-computer interface. *7th ERCIM Workshop "User Interfaces for All," UI4ALL 2002*, pages 195 – 196, 2002. 6

[2] S. Janaqi D. Sidibe, P. Montesinos. A simple and efficient eye detection method in collor images, 2006. 5

[3] Masayuki Asano Hironobu Takano and Kiyomi Nakamura. Real-time eye detection method robust to facial pose variations using gradient directional features and particle filter, 2013. 5

[4] J. Lombardi and M. Betke. A camera-based eyebrow tracker for hands-free computer control via a binary switch. *7th ERCIM Workshop "User Interfaces for All," UI4ALL 2002*, pages 199 – 200, 2002. 6

[5] Shun Ido M. Hassaballah, Kenji Murakami. An automatic eye detection method for gray intensity facial images, 2011. 5

[6] OpenCV. Cascade classification. `http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html`. 22

[7] Lotte NS Andreasen Struijk. Tongue-computer interface for disabled people. *International Journal on Disability and Human Development*, 5:223 – 226, 2011. 6

[8] Geoffrey Underwood. Cognitive processes in eye guidance. *Oxford University Press*, 2005. 2

[9] vaish.ra...@gmail.com. Face recognition in video stream. https://code.google.com/p/face-recognition-in-video-stream/. 16

[10] Paul Viola and Michael Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57:137 – 154, 2004. 9

[11] WebAIM. Motor disabilities - assistive technologies. http://webaim.org/articles/motor/assistive. 1

[12] Wikipedia. Adaboost. http://en.wikipedia.org/wiki/AdaBoost. 4