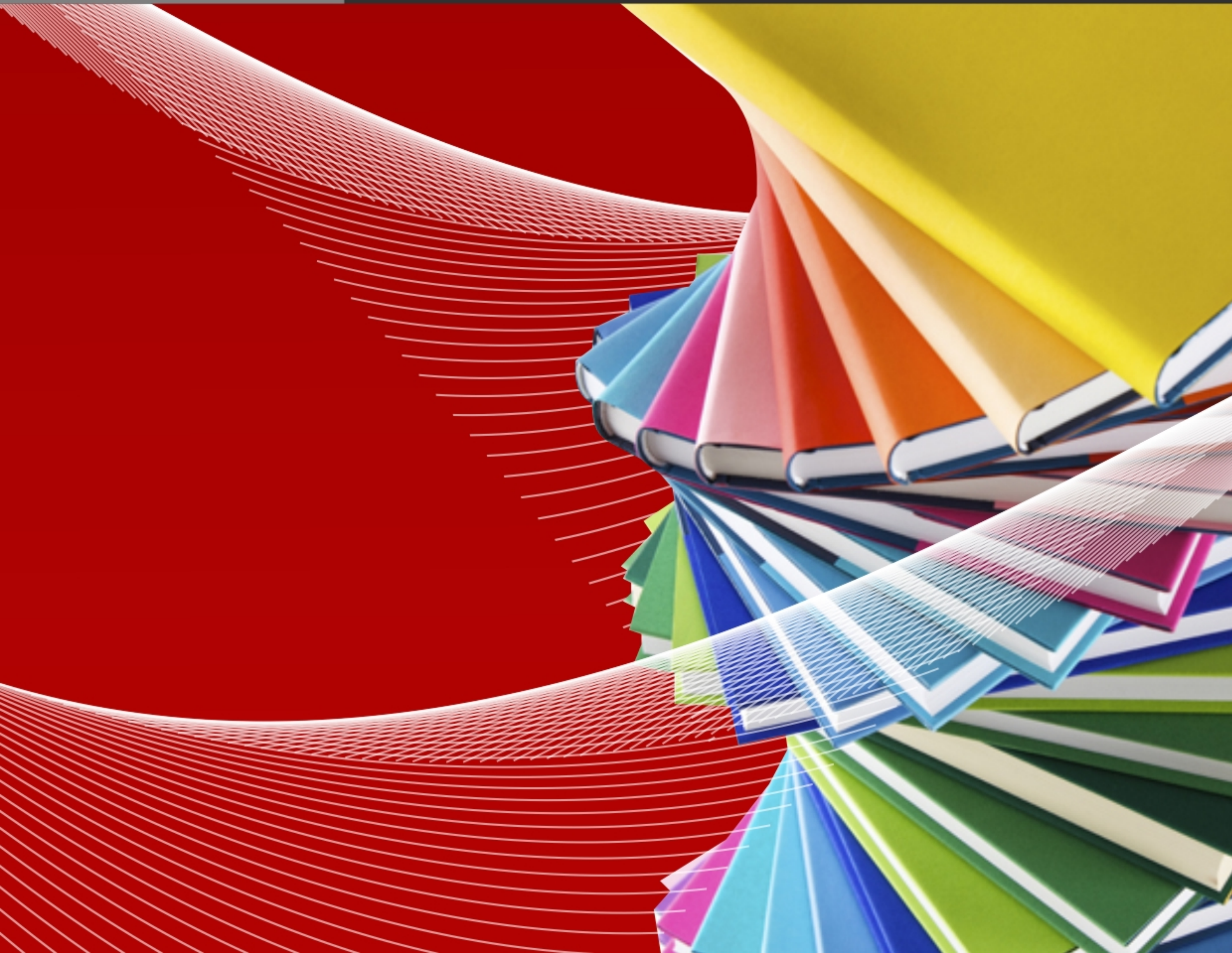
澳門大學
UNIVERSIDADE DE MACAU
UNIVERSITY OF MACAU

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

Wai Kin Ho, Student No: D-B0-2704-3

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 : Wai Kin Ho

Student No. : D-B0-2704-3

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, Meng Chu Cheong, for all the efforts and this thesis really would not have been completed without her contribution.

Abstract

Nowadays, although electronic devices are everywhere, 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

Listings

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 computers 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 reasons, 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 [14]. However, those equipments have some limitations and therefore can't fit for 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 phones, computers and most of the hand-held devices. Besides, our eyes are the most salient features of our face. We can get different information of others through eyes' contact, e.g. their thought, needs, cognitive processes, emotion, and even the interpersonal relations [12]. 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 structure enables our system to capture the environment as an input, locate the user's eyes and head, detect and analysis the user's motions, and helps user to accomplish simple operations of electronic device. In addition to the system structure, our work mainly focuses on locating the user's irises since it's the most difficult part throughout the structure. Once the irises are detected, the user's eyes 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 eyes rotation and support the turning-page function while user is reading the Power Point.

My work is mainly focus on the implementation of sensor input layer and the motion detection layer. In the sensor input layer, the sensor needs to be integrated into our system to capture the color image and the depth image. Second, the noise of color images is needed to be removed before any processing. In the motion detection layer, a quick and accurate iris detection method is needed to be implemented in order to detect the user's motion.

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 system features. Chapter 5 presents Experimental results are presented in Chapter 5. Chapter 6 includes conclusion and future the implement. Conclusion is presented in Chapter 7 and future works in Chapter 8.

Chapter 2

Related work

Currently, most of the electronic devices do not provide the motion capturing function. Although there are some stand-alone sensors are developed independently for that purpose, it is not common. In order to see the potential of our project, some researches are presented in the first section. In the second section, different iris detection methods are introduced and explain the reason why they are not the best candidate for our system.

2.1 Potential of this project

Although the motion capturing is not common nowadays, the new generation is coming soon.

In fact, there have been some motion capturing sensors for the electronic device. Except the Kinect, which will be introduced in Section. 3.1, ASUS also developed a similar devices, Xtion Pro [2]. It provides two lens for depth image capturing, one image color camera and two microphones. It also provides the SDK for developers to implement their system in C++ or C# on the Windows or Linux platform. However, the Kinect is a little bit better than Xtion Pro compare to the physical devices of both of them.

Besides, the project of the Structure Sensor [9] developed by Occipital company is successfully funded on 1st of November in 2013. The Structure Sensor provides the 3D depth image capturing function and it is able to attach to the mobile devices which is running on the IOS platform. It also provides the SDK for manipulate the sensor. With this sensor, our system structure can also be implemented in the mobile devices, such as iPhone, iPad.

On the Android platform, according to the latest report, Google is developing a new tablet with two cameras and a infrared depth sensor. The tablet is developed as part of Google's new project Tango program [4] and they claimed that they plan to produce 4000 prototypes and they are going to be released before the Google I/O at the end of June in this year.

We can expect that in the future, most of the electronic devices are able to capture the user's motion, and the traditional approaches of accessing the interface will be changed.

2.2 Current iris detection methods

Researchers have been done on different aspects of iris detection method. Basically the current iris detection methods are divided into four main parts: shape-based, appearance-based, feature-based, and hybrid methods [5].

A shape-based method proposed by Yuille et al [16] which is using a deformable eye model consists of two parabolas representing the eyelids and a circle for the iris and the model is fitted to the image through an update rule. However, this kind of method is computational complex due to the complexity of the template. Also, when the eye occluded by the eyelid, this method also have a difficulty to locate the iris.

Another method proposed by Kim and Ramakrishna, R.S. [6] is utilizing the edge of the iris. They proposed the longest line scanning and the occluded circular edge matching which is less complex than the method of eye model fitting proposed by Yuille et al. However, this method only works on near fontal-faces and the exact edge of iris is not easy to extract from a noisy image captured by low resolution camera.

Kothari and Mitchell [7] have proposed a method that is similar to us and our method is the improved version of it. They utilize the gradient vector field and extrapolated in a direction opposite to the gradient in order to locate the iris. However, the result shows that it's not accurate enough since the eyebrows, eyelids or glasses may influence quite a lot.

There are also some other different methods, for example, a learning-based method proposed by Reinders et al. [10], a voting scheme utilizes the isophote curvature proposed by Valenti and Gevers [13].

However, most of the methods do not mention the efficiency. Some of them need a very clear image or a near-frontal image. In order to capture the user's eye motion in real-time video stream. The method with efficient, working on low resolution images, and high accuracy rate is needed.

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**. Firstly, the **sensor input layer** captures the depth image and color image from the environment through sensor. After that, the **image analysis layer** locates the position of eyes and heads from the color image. Then, combining the depth image, the **motion detection layer** detects the user's motions according to the location of eyes and heads. Finally, the **application interface layer** triggers the corresponding pre-defined event and giving back the response to user according to the motions the system detected. In the following, the structure will be introduced layer by layer in detail.

3.1 Sensor input layer

The sensor input layer is used to capture the information from environment as an input to our system through sensor. Since camera is the most common device nowadays, it should be the best candidate. However, concerning about the speed of capturing user's motions, manipulate cameras are better. Although there are some new generation of devices with two cameras and infrared depth sensor is under developing, at present, the only suitable device for our project is Kinect for Windows [8].

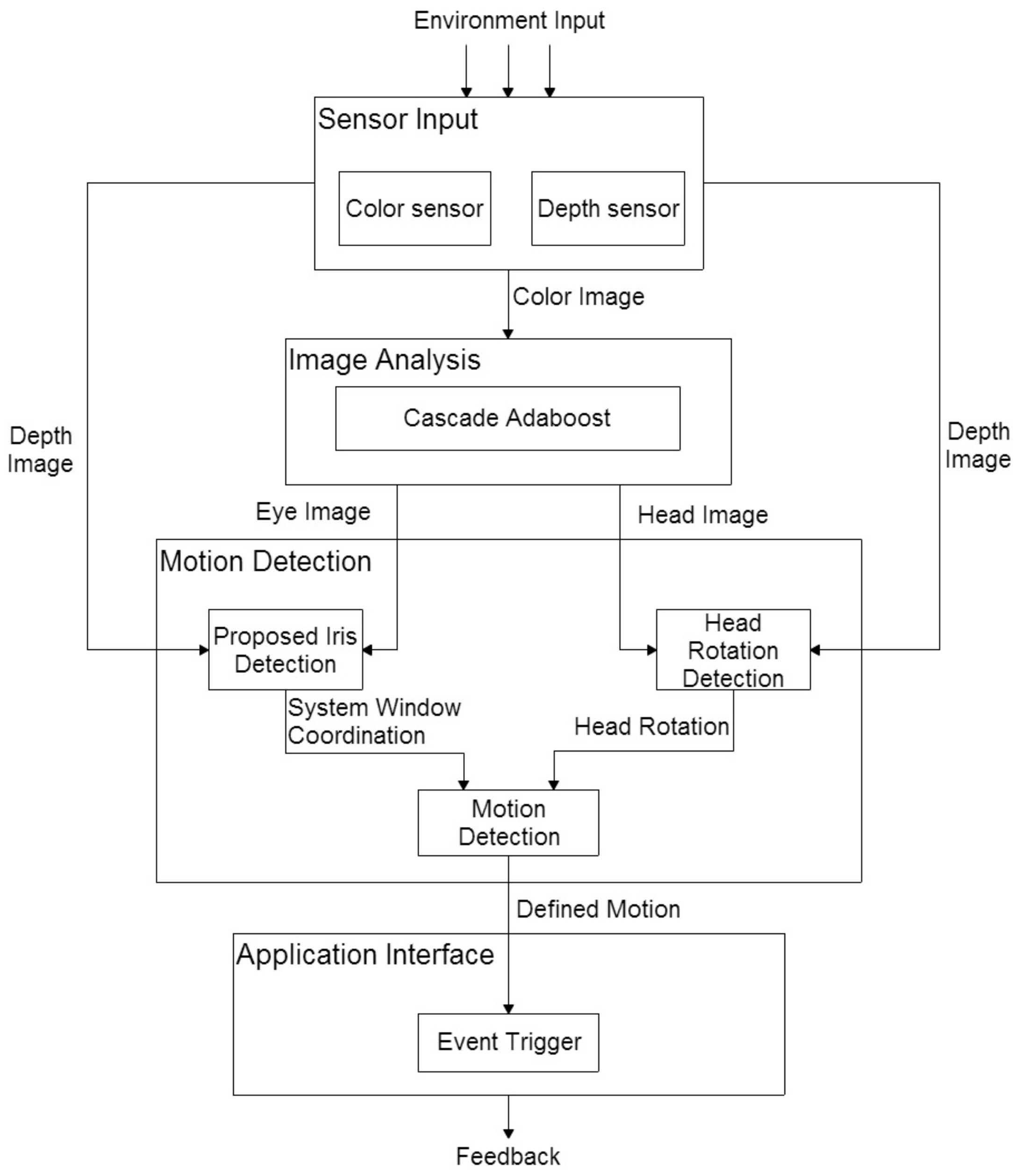


Figure 3.1: The proposed system structure

Kinect for Windows is a motion-capturing device developed by Microsoft. Originally, it's designed for playing video games with X-Box 360. It provides one color image camera and two lenses for depth image capturing. Also, Kinect provides a microphone array in order to capture the voice. Moreover, Microsoft also provides a set of robust SDK for Windows to manipulate. The SDK provides functions like skeleton tracking, gesture detection and voice recognition and enables developer to use C++, C#, or Visual Basic to develop application. The latest version of Kinect device is Kinect v2 and SDK v1.8.

The supported resolutions of the image camera of Kinect are 640×480 with fps 15, 640×480 with fps 30 and 1280×960 with fps 12. Also, it supports different format of color, like RGB, YUV and Bayer. In our implementation, the resolution of the camera is configured to the setting of 640×480 with fps 15 with YUV format.

Although Kinect for Windows is multi-functional, the only function it provides to our system is the depth image capturing. If any other device can provide or replace this function, our system is able to independent from Kinect for Windows totally.

After the color images are captured from the camera, the Gaussian filter with a size of 3 is applied to the image in order to remove the noise and amplify the contrast by using the histogram equalization. After this, it's ready to process the captured input in next layer.

3.2 Motion detection layer

After some processes of the second layer(image analysis layer), the system is able to locate the positions of user's eye. After that, user's motions can be detected by tracking their eyes and head. However, due to the time constrain, our project mainly focuses on tracking user's eyes with iris detection method, which also is the most difficult part of the system. In order to detect user's iris, the original method proposed by Fabian Timm and Erhardt Barth [11], hereafter referred to as "original method is used. There is one common characteristic of any nationality or ethnicity of people, that is, their eyeballs are all circular shape. Therefore, the basic idea is to find the center of a circular object and, geometrically, the center can be located by analyzing the gradient vector field of the image.

3.2.1 Iris detection method

According to the Fig. 3.2, the center of a circle has a property that the vector from the center c to any point at the edge x_i, denoted as d_i, is parallel to the gradient vector at the point x_i, denoted as g_i.

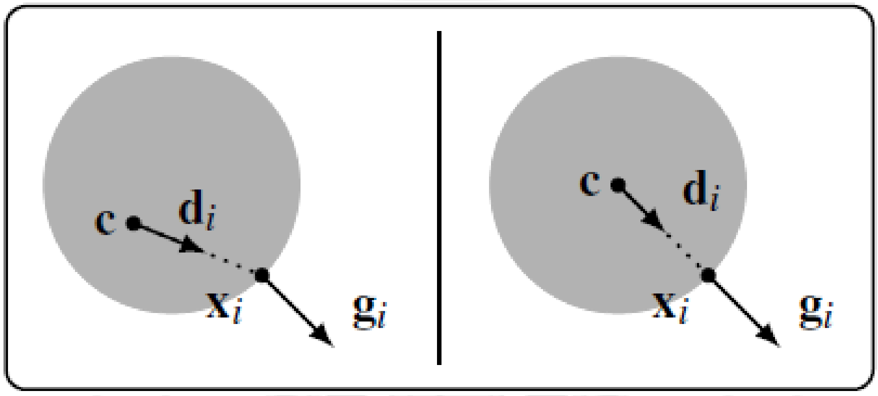


Figure 3.2: The vectors d_i and g_i corresponding to c

Recall the definition of inner product of two vectors,

$$\overrightarrow{A} \cdot \overrightarrow{B} = |A| \times |B| \times \cos(\alpha) = (x_1 \times x_2) + (y_1 \times y_2) \tag{3.1}$$

where $\overrightarrow{A} = (x_1, y_1)$, $\overrightarrow{B} = (x_2, y_2)$, α is the angle between $\overrightarrow{A}$ and $\overrightarrow{B}$.

The maximum value can be obtained when α is equal to zero, that is, $\overrightarrow{A}$ is parallel to $\overrightarrow{B}$. Note that the value of the inner product will vary according to the magnitude of the vector. Therefore, all the vectors before the calculation needed to be normalized.

The relationship between a possible center and the orientations of the image gradient field can be described as:

$$c^* = \arg_c \max\{\frac{1}{N}\sum_{i=1}^{N}(d_i^T g_i)^2\} \tag{3.2}$$

where

$$d_i = \frac{x_i - c}{\| x_i - c \|^2}, \forall i : \| g_i \|_2 = 1. \tag{3.3}$$

One important thing which do not mention in the original method is that, two vectors with opposite direction should be ignored, which can be done by only summing up the positive value of the dot product.

Furthermore, in addition to the original method, some extra works have been done. First, the original method involves all the gradient vectors of an image into the calculation. In fact, it can be dramatically reduced the complexity of the calculation by only involving the points on edge of the image. Recall that the gradient image can be obtained using the Sobel operator (Fig. 3.3), and rough edge can be detected by applying a threshold on the gradient image. Besides, only the most 500 intensive gradient vectors are reserved and involved into the calculation after extract the edge point in order to further reduce the complexity.

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

Figure 3.3: Sobel operator

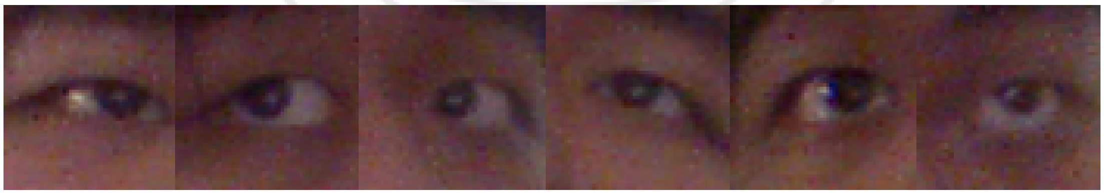


Figure 3.4: Examples of result of cascaded Adaboost. Iris most of the time is located in the middle of the result.

Second, the local maximum of the inner product is sufficient or even better to indicate the center of a circular object. Due to the observation of the result from cascaded Adaboost (Fig. 3.4), most of the time the most important information is concentrated in the middle of the image. Moreover, some of the unnecessary information, such as eyebrow, glasses, is on the border of the image. A local maximum can be found quickly by an iterative method initiated from the center point, denoted as p, of the image. Each time the inner product is calculated within the neighbor of p, and select the maximum as the new p and continue the process until no point with larger inner product can be found. The whole algorithm is shown in Fig. 3.5.

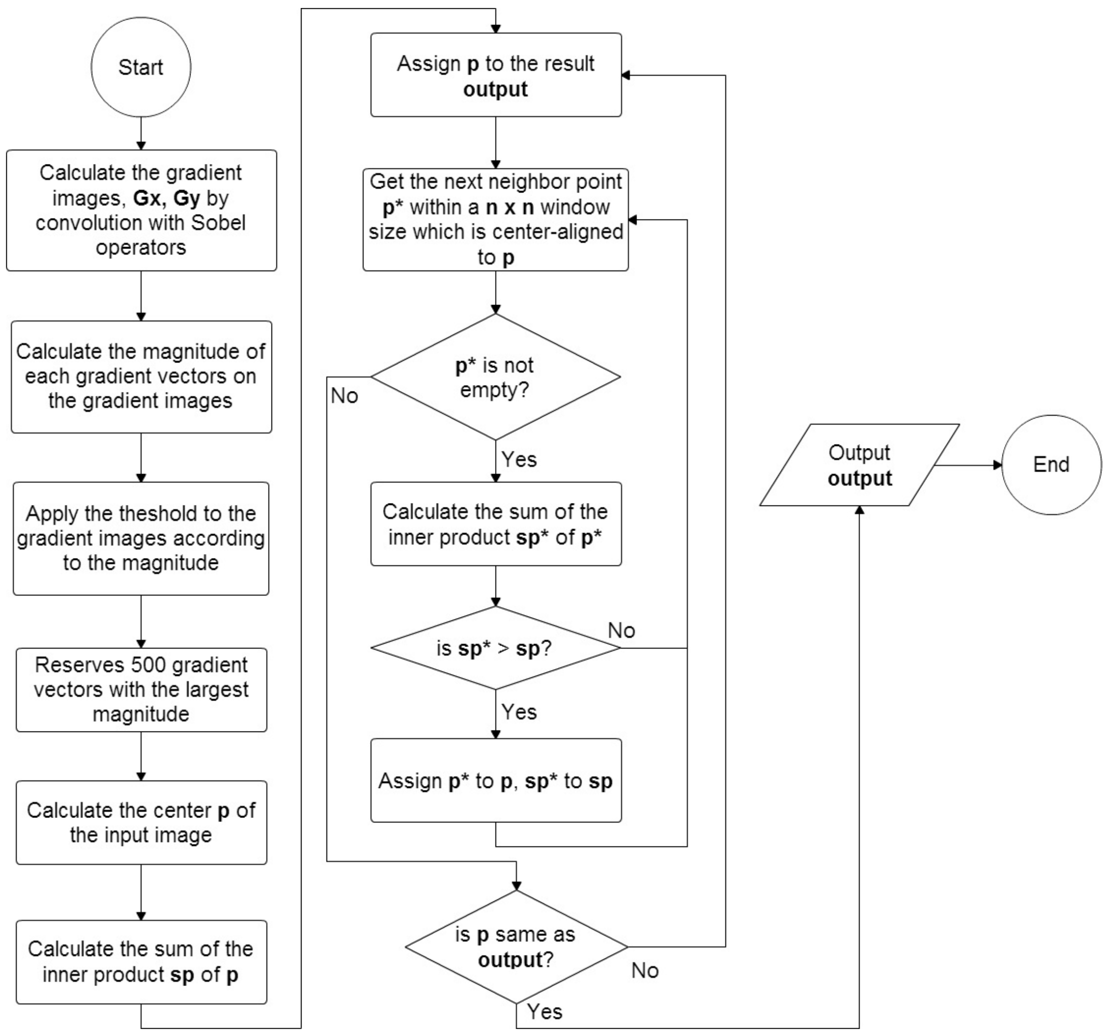

Start
Calculate the gradient images, Gx, Gy by convolution with Sobel operators
Calculate the magnitude of each gradient vectors on the gradient images
Apply the theshold to the gradient images according to the magnitude
Reserves 500 gradient vectors with the largest magnitude
Calculate the center p of the input image
Calculate the sum of the inner product sp of p
Assign p to the result output
Get the next neighbor point p* within a n x n window size which is center-aligned to p
p* is not empty?
No
Yes
Calculate the sum of the inner product sp* of p*
is sp* > sp?
No
Yes
Assign p* to p, sp* to sp
is p same as output?
No
Yes
Output output
End

Figure 3.5: Algorithm of the proposed iris-detection method.

There are two parameters in this method: **threshold** and **window size**. The **threshold** is applied to the gradient image in order to extract the edge points. The **window size** indicates, during calculating the sum of inner product of p, the size of the region which is center-aligned to the point p, and the pixels in that region are considered as the neighbor of the point p. The selected values of these two parameters are presented in the Section 6.1 with the experimental results.

After located irises, the motion of user's eyes can be detected. A simple method is to ask user for staring at some points (e.g. four corners of the screen) in order to calibration the initial position of irises. Due to the problem of slight head movement, it can be re-corrected by tracking the center point between two eyes and the distance between eyes and Kinect. The idea of calibration is discussed in the next section.

3.2.2 Calibration method

The basic idea of calibration is to make use of the Gaussian distribution. The Gaussian distribution, also known as normal distribution, is a very common probability distribution. The curve of the function is shown in Fig. 3.6, also known as bell-shaped curve, and the formula is shown as follow:

$$f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.4}$$

where μ is the mean and σ is the standard deviation.

During the calibration, first, the user is asked for looking at some points on the screen. Meanwhile, the system captures the positions of user's irises and records all the captured data. The system captures 50 data points of each of the user's iris, and theoretically, it needs to take about 3 second with 15 fps of the camera. After collecting enough points of the user's irises, the system calculates the mode and the variance of the data points. Note that the mode is used instead of mean since the positions of the user's irises might be unstable due to the wrong detection of the iris, some unconscious motions of user, eyes blinking, or noise. The mean is affected a lot in those cases, but mode isn't. Therefore, the mode is selected as the parameter of Gaussian distribution instead of mean. In order to have a more accurate result of calibration, the variance should be less than a certain value. If the variance is too high, it indicates that the result are affected by some issues. In that case, the calibration for that iris needs to redo again until the variance o f the data is low enough. After the variation and mode is calculated, the Gaussian distribution model can be built for each event.

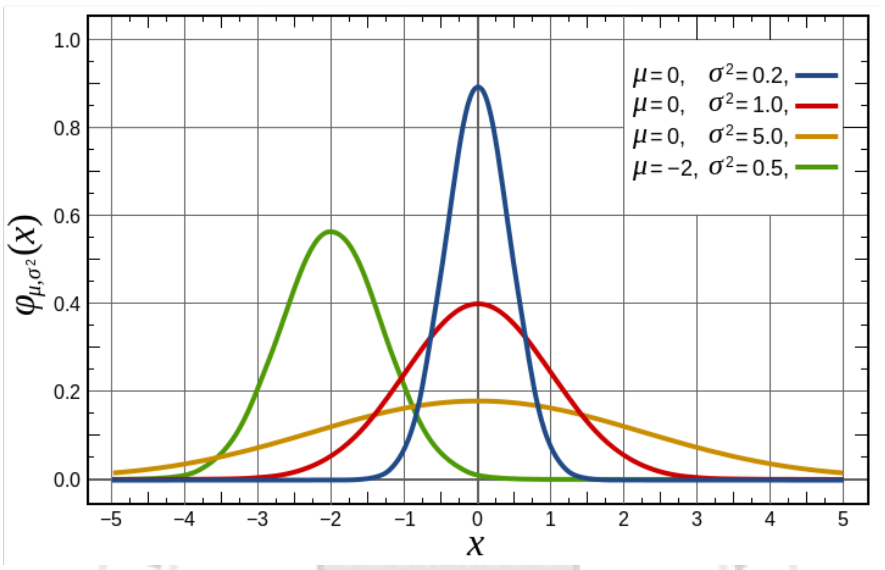


Figure 3.6: Normal distribution [15]

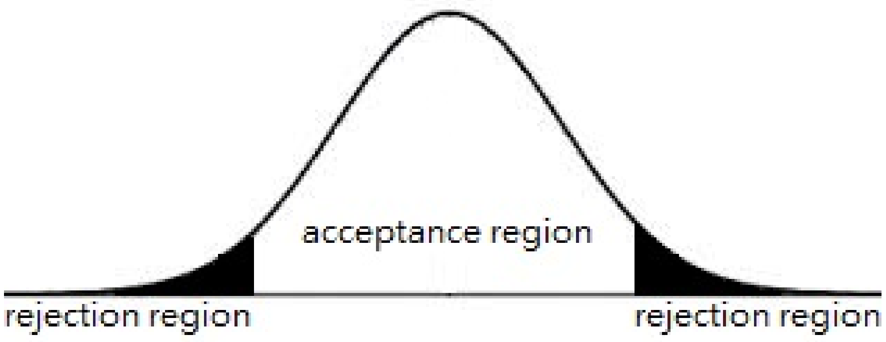


Figure 3.7: Two tail test.

In order to calculate the probability, two-tailed test are used (Fig. 3.7). The probability of a variable x with two-tailed test can be calculated by the follows:

$$\begin{aligned} p = & \int_{-\infty}^{-x} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx + \int_{x}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \\ = & \quad 2\int_{x} \infty \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \end{aligned} \tag{3.5}$$

It calculates the surface under the interval $(-\infty, -x]$ and $[x, \infty)$ of Gaussian function. Since the Gaussian distribution is symmetric to its mean, only half of the surface needs to be calculated, and the remaining parts can be done by doubling the result.

To calculate the integral, we can use the trapezoidal rule. It is a numerical method for approximate an integral. It divides a continue function into some trapezoids, calculates their surfaces and sums them up. For a domain divided into N equal interval, where $a = x_1 < x_2 < K < x_{N+1} = b$, the formula of trapezoidal rule is shown as follows:

$$\begin{aligned} \int_a^b f(x)dx \approx \frac{h}{2} \sum_{k=1}^{N} (f(x_{k+1}) + f(x_k)) \\ = \frac{b-a}{2N} (f(x_1) + 2f(x_2) + 2f(x_3) + ... + 2f(x_N) + f(x_{N+1})) \end{aligned} \tag{3.6}$$

However, note that the interval of integral of formula 3.5 is from x up to ∞, such b can't be obtained in that case. Therefore, a $\triangle x$ is specified as the increment instead of $(b-a)/2N$, and the calculation stops when the $f(x_i)$ is smaller enough.

After the calibration is done, the model is built. Then, we can detect the user's motions by input the position of user's irises into the model and calculate the likelihood to trigger the event.

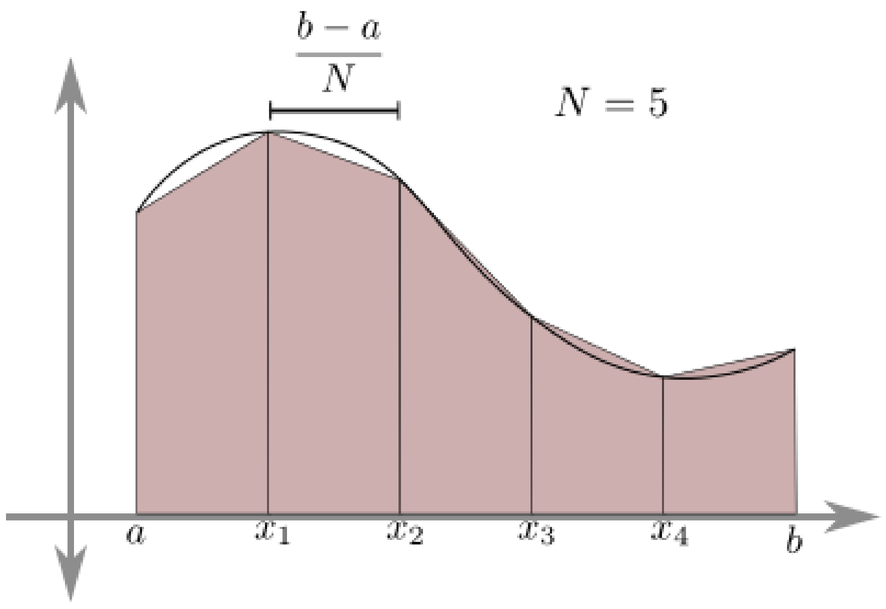


Figure 3.8: Discrete Integration [3]

Chapter 4

System features

Our system can help user to accomplish some simple tasks with computer through eyes rotation 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

In this section, the implementation of some important parts of our system is presented. Recall that this report focuses on two layers: sensor input layer and motion analysis layer, some special techniques such as data flow and data structure in these two layers are highlighted. Some codes of the crucial algorithm in this section can be referred to the Appendix.

5.1 Sensor input layer

This layer mainly focuses on how to capture the input environment. Since the configuration can be easily found in the Kinect for Windows documentation, the main focus in this section is the data flow of this layer.

Note that the images captured from Kinect for Windows is fetched frame by frame. The Kinect for Windows SDK provides some methods for developer to register some handler functions to the sensor corresponding to each input, that is, color image and depth image input frame. The handler function is invoked each time each frame of image is captured. With the parameters which are passed by the handler, the corresponding input can be fetched into the system.

To propagate the image from this layer to other layers, same working principle is adapted. Since the depth image captured has not yet implemented into our first prototype, there are two functions that are implemented for other layers to register the handlers: the gray image input handler function and the color image input handler function. Since all of our algorithms are worked with gray-level image, the conversion between color and gray level image is done in the this layer, as well as the Gaussian filtering and the histogram equalization. However, in order to display the results in color, it needs to access the color images. Therefore, two input handlers are opened for different purpose. Furthermore, under this design, the structure of implementation of other layers becomes clear. All the image-processing can be implemented in the gray image handler, and left all the implementation of displaying output in the color image handler.

Besides the data flow, one thing about the data structure is needed to be highlighted. Since EmguCV library provides most of the functions of image-processing to our system, the images are needed to store in the structure the library can process. The structure to store the images which are captured from Kinect for Windows is an array of byte, and the structure using in EmguCV library is Emgu.CV.Image. In order to transform the format of the captured images, an intermediate format, that is, the System.Drawing.Bitmap, is used. The codes of the conversion between them is listed in Appendix, List. 1.

5.2 Motion detection layer

This layer tracks the user's eyes motions. It is composed of two parts: calibration method and iris detection method. About the data flow of the motion detection layer. Since there are only two events are implemented in our first prototype, this layer sends three kinds of flag as output to indicate different state of the users eyes, that is, user is looking at the right side, left side, or neither side.

In the first section, some technical issues of the calibration method are discussed. After that, the implementation of iris detection is presents in the second section.

5.2.1 Calibration

In this part, the Gaussian distribution is used to build up a probabilistic model. To calculate the probability of each event, the trapezoidal rule with a little modification is proposed.

During the calibration, some data points are collected. Each data points contain four data values, that is, the x and y coordinate for each eye. Also, a set of data points needed to be collected for each event. Therefore, the data volume is up to $4 \times amountofdata \times amountofevent$. To store such a data, an array with the *EyesData* class is implemented. The *EyeData* class consists of four integer variables, corresponding to the coordinate for each eye. The data points of each event are interleaved stored, that is, the data for the first event are stored in the position of $0 + k \times amountofevents$, for the second event are stored in the position of $1 + k \times amountofevents$, and so on. Therefore, only a one-dimensional array is employed to store the data instead of an n-dimensional array for n events.

One more thing about the calculation of the probability, note that if the variable x is smaller than the mean, the interval of the integral should be from $-\infty$ to x, otherwise, it should be from x to ∞. Therefore, we need to change the sign of the increment according the variable x.

Some related codes about the calculation of mode, variance, and probability is presented in the Appendix, List. 5.

5.2.2 Iris detection method

The basic idea of the proposed iris detection makes use of the gradient vector field. It can be obtained by the Sobel operator. One thing should be toke care is that since the representation of the coordinate in traditional image-processing and mathematic is different, the Sobel operator shown in Fig. generates the gradient vectors with an inverse direction to the vector d_i (please refer to Fig. 3.3). In order to correct this, the rows and columns are turned upside down and laterally reversal. The modified Sobel operator is shown in below:

Listing 5.1: Code of Sobel operator

```
private static float[,] GY = new float[3, 3] {
    {-1, 0, 1},
    {-2, 0, 2},
    {-1, 0, 1}
};

private static float[,] GX = new float[3, 3] {
    {-1, -2, -1},
    { 0,  0,  0},
    { 1,  2,  1},
};
```

Besides, the efficiency of the program can be improved by some code optimization. First, the access of the variables from the ***Emgu.CV.Image*** is too slow. To traversal an image stored with the **Image** format, it is double times slower than traversing an image stored with an array format. It's better to convert the format into an array. However, instead of convert the whole image to an array, the better way is to only store the necessary data in a one-dimensional array. Recall that the proposed method only selects the gradient vectors which are on the edge into the calculation. There are two kinds of data required in the algorithm: vector and point position. The data of vector including the gradient vectors and the vectors d_i, and the point position including the location of the points on which the gradient vectors are selected. This information is needed in order to calculate d_i. Note that these two data are obtained after applying a threshold to the gradient image. Therefore, during the thresholding, instead of putting zeros in the gradient image, the candidate of gradient vectors are inserted into a one-dimensional array. After this optimization, the gradient images only needed to be traversal once, and the remaining part of the iris detection method is totally free from the *Emgu.CV.Image*.

By the way, more about the data structure, there are two classes are implemented: the *Vector* class and the *Position* class. The *Vector* class, just as its name implies, is using for storing the vector. It includes two data members with type of double, i.e. x and y, corresponding to the direction of the vector. It also concludes two methods: *Magnitude()* and *Normalize()* for calculate the magnitude of the vector and normalization. Note that the value of magnitude doesn't stored in the class since most of the time the algorithm only requires the direction of the vector, but thresholding. The *Position* class is using for store location of a point. It only includes two integer variables, also x and y. This class is more general than *Vector* and it is invoked throughout all the layer, for example, to store the points of user' irises, or the location of each gradient vectors.

More details about the thresholding, firstly, recall that the proposed method only reserves 500 the most intensive gradient vectors. In order to quickly filter out these 500 vectors, some operations of heap is implemented. A heap is a specialized tree-based data structure. It has the advantages such as it can locate the minimum or maximum value from data set quickly by reserving the minimum information about which, and the insertion is relatively low cost. Also, it can be implemented with an array, which is compatible with our data structure. Note that to find out the 500 most intensive vectors, the only information needed is the current minimum value of the intensive. It can be done by comparing the current value to the minimum and replacing the minimum value whenever the current value is larger than it. However, the heap may break the data into fragments if the amount of elements is smaller than the capacity of heap. Some advantages of the array are lost due to fragmentation since the whole array needed to be traversed each time in order to check whether it still has the data in behind. To avoid the fragmentation, when the number of elements is under the capacity, the new elements are inserted into the array normally without any heap operation. Whenever it reaches the capacity, the *BuildHeap* function is invoked to convert the array into a heap. After that, all the new elements are inserted into the array with the heap operation.

Secondly, as mentioned before, in most of the time magnitude is not required, but thresholding. Moreover, both the vector and the position are required to be stored. Instead of create a new class for these data, a derived class from the *Position* class, the *ExPosition* class, is implemented. Besides the data member it includes originally, it also includes an extra double variable, magnitude, and a vector v. Since the *ExPosition* is derived from Position, the results which is stored in an array with type of *ExPosition* can be directly assigned to the array of *Position* by upcasting. It keeps the *Position* class more general and more efficient.

Another code optimization is make used of multi-threading. One advantage of the proposed iris detection method is that the algorithm can be highly parallelized. There are four parts which can be parallelized in the program. First, the whole process can be parallelized since each point on the image can be processed individually. Second, the calculation of the magnitude in the process of thresholding can be parallelized. However, the process of selecting the most intensive vectors can't be parallelized. Third, the process of calculating the inner product can be parallelized, note that it doesn't include the summing part. At last, the whole process of calculating the d_i can be parallelized.

In Appendix, List. 2 shows the data structure of the iris detection. List. 3 shows the heap operation. List. 4 shows the process of thresholding.

Chapter 6

Experimental results

In this session, the results are discussed into two parts. In the first part, the results of the proposed iris detection method are shown. It compares the processing time and the accuracy rate between the original method and the proposed method.

To compare the processing time, different scale of images are input into the program and the processing time is recorded. Besides, in order to simplify the work of the comparison, the original method is implemented regardless the prior knowledge and post-processing .

In order to compare to the original method, the BioID database [1] is used to evaluate the accuracy. The accuracy of the method is measured by **normalized error** proposed by Jesorsky et al. It is defined as:

$$e \leq \frac{1}{d}\max(e_l, e_r) \tag{6.1}$$

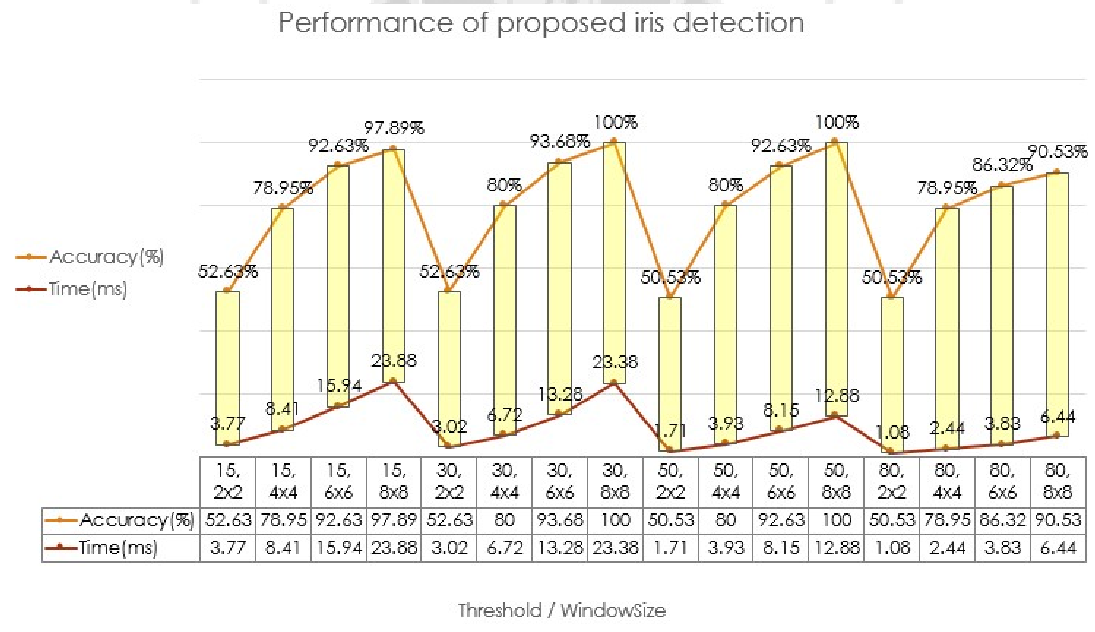
where e_l, e_r are the Euclidean distances between the estimated and the correct left and right eye centers, and d is the distance between the correct eye centers.

In the second part, it presents the results of the comparison between the proposed method and the state of art (including the original method).

6.1 Result of iris detection method

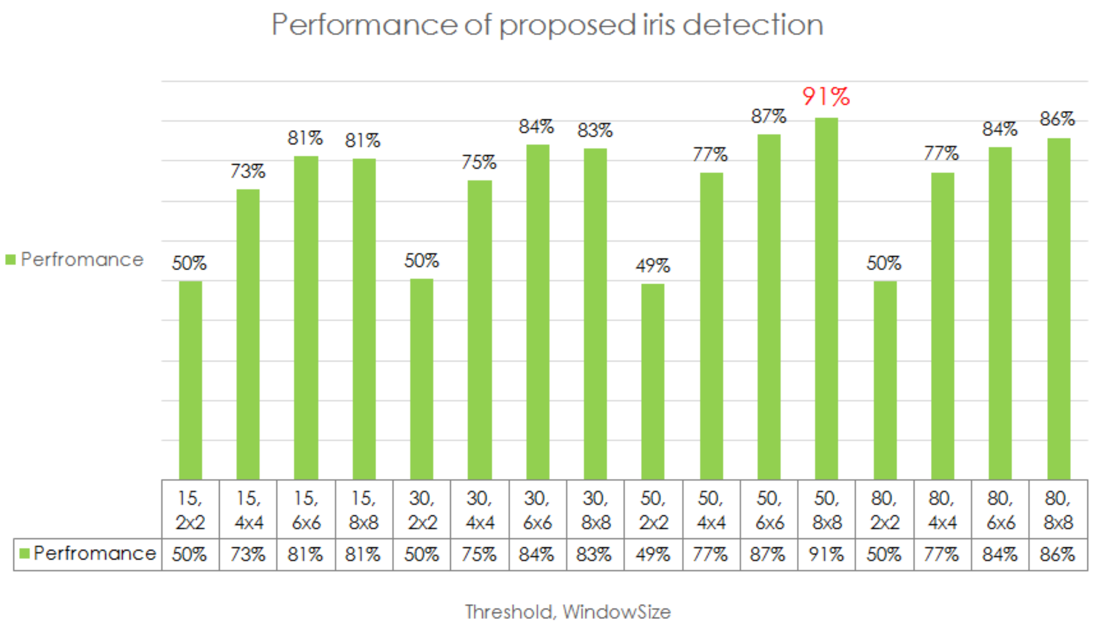
In this section, it presents the performance of the proposed iris detection method and the experimental results for choosing the parameters.

There are two parameters need to nailed down: **threshold** and **window size**. Some experiences have been done to compare the processing time and accuracy corresponding to each parameters, which is shown in Fig. 6.1. The yellow bars are the relative difference between the accuracy and processing time. In order to compare each parameters, the relative different is quantified and shown in Fig. 6.2. Note that the data of Fig. 6.2 is only a valued-representation of the yellow bar of Fig. 6.1. The values of data only for comparing between each parameters and do not indicate any other information. According to the figure, threshold can affected very much in performance since the higher threshold can eliminate more amount of gradient vectors. On the other side, the window size affect a lot the accuracy rate. The best result is indicated by the threshold of 50 and the window size of 8×8.



	15, 2x2	15, 4x4	15, 6x6	15, 8x8	30, 2x2	30, 4x4	30, 6x6	30, 8x8	50, 2x2	50, 4x4	50, 6x6	50, 8x8	80, 2x2	80, 4x4	80, 6x6	80, 8x8
Accuracy(%)	52.63	78.95	92.63	97.89	52.63	80	93.68	100	50.53	80	92.63	100	50.53	78.95	86.32	90.53
Time(ms)	3.77	8.41	15.94	23.88	3.02	6.72	13.28	23.38	1.71	3.93	8.15	12.88	1.08	2.44	3.83	6.44

Figure 6.1: The accuracy and processing time of proposed iris detection method.



	15, 2x2	15, 4x4	15, 6x6	15, 8x8	30, 2x2	30, 4x4	30, 6x6	30, 8x8	50, 2x2	50, 4x4	50, 6x6	50, 8x8	80, 2x2	80, 4x4	80, 6x6	80, 8x8
🟩 Perfromance	50%	73%	81%	81%	50%	75%	84%	83%	49%	77%	87%	91%	50%	77%	84%	86%

Figure 6.2: The performance of proposed iris detection method.

Image Size	Without multi-threading			With multi-threading		
	best	avg	worst	best	avg	worst
0	0	0	0	0	0	0
20x20	1	4	4	1	4	8
40x40	2	13	11	2	11	16
60x60	7	17	16	4	13	13
80x80	9	23	15	4	15	17
100x100	13	33	30	5	18	31
120x120	14	40	25	6	19	20

Table 6.1: The processing time of the iris detection method with and without multi-threading

Fig. 6.3 and Table. 6.1 shows that the processing time of the proposed iris detection method with and without the multi-threading. Just as mentioned in Chapter. 5, one advantage of the proposed iris detection is can be highly parallelized. In order to show this, several images are input into two programs of the same iris detection method, one is optimized with multi-threading, another isn't, and the processing time is recorded. The results are generated by inputting 1502 face images into the program, and each face image contains two eyes. Every eyes on the face images are located by some rectangle regions with different size. In fact, it is equivalent to input different size of eye images. Therefore, the results are actually the processing time of two times of iris detection. Note that the accuracy of the algorithm isn't indicated in this result.

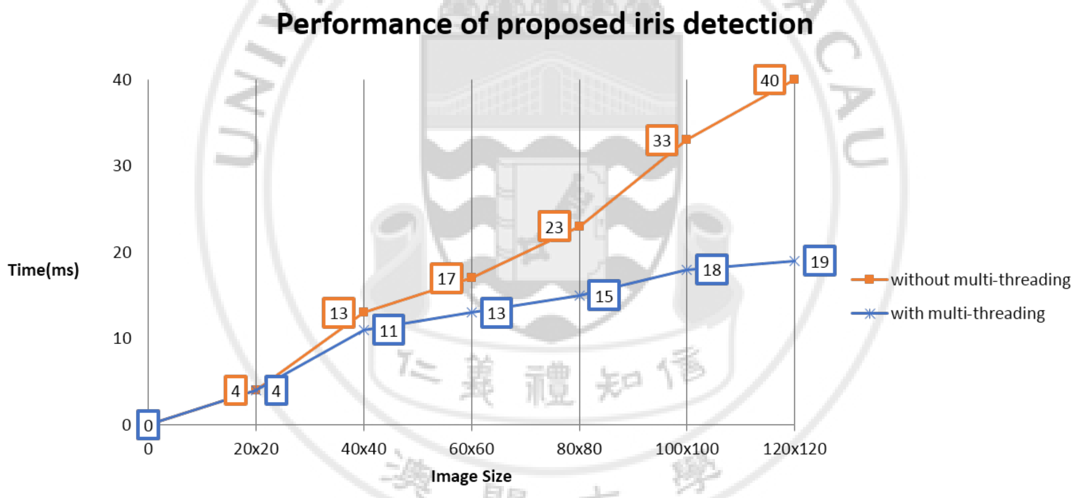


Figure 6.3: Improvement with multi-threading.

According to the figure, although the results generated by the images with smaller scale ($20 \times 20, 40 \times 40, 60 \times 60$) don't perform big different, it almost or even performs double times quicker with the images with large scale ($100 \times 100, 120 \times 120$).

After that, the results of comparison between the original method and the proposed method according to time and accuracy are shown. Table. 6.2 shows that the processing time with different size of the input images. Fig. 6.4 shows that the comparison of the average processing time between the original method and the proposed method. The processing time is generated by same method

Image Size	Original			Improved		
	best	avg	worst	best	avg	worst
0	0	0	0	0	0	0
20x20	15	56	48	1	4	8
40x40	104	375	335	2	11	16
60x60	253	850	554	4	13	13
80x80	447	1573	935	4	15	17

Table 6.2: The processing time of the original and proposed iris detection method

mentioned above. By theory, the original method has a running time of $O(n^3)$ respect to x, however, our method, only takes $O(n)$. According to the figure, the proposed method has a better and better performance of processing time when the size goes larger and larger.

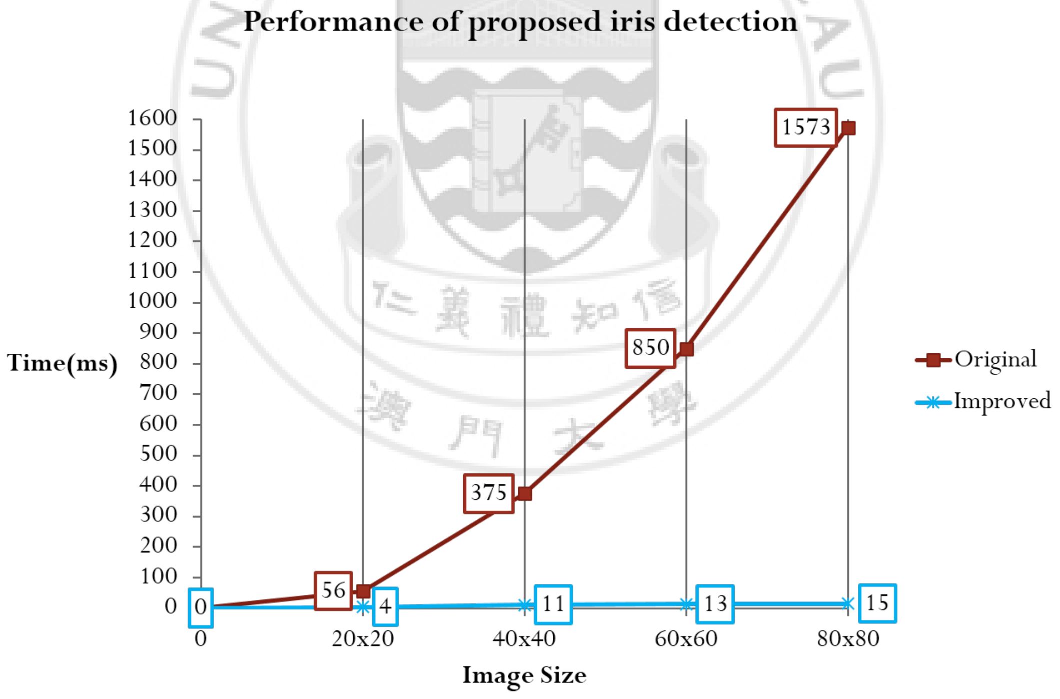


Figure 6.4: The comparison of processing time between proposed iris detection method and the original method.

Besides, Fig. 6.5 and Table. 6.3 shows the accuracy of the proposed method and compares to the original method. To extract the image of user's eyes, a size of 40 × 40 rectangle is drawn at the point of the iris. In order to simulate the result of cascaded Adaboost, a random offset with ±20% to its size is added to the located point of the rectangle. The results are generated with the threshold of 50 and the window size of 8 × 8. Moreover, the gaussian filter is disabled. Although the proposed method does not perform as good as the original in the normalized error of 0.05, it does in most of the time.

Normalize error rate	Original	Improved
0.05	82.50%	74.23%
0.1	93.40%	94.48%
0.15	95.20%	97.44%
0.2	96.40%	98.82%
0.25	98.00%	99.51%

Table 6.3: Normalize error rate of the iris detection method

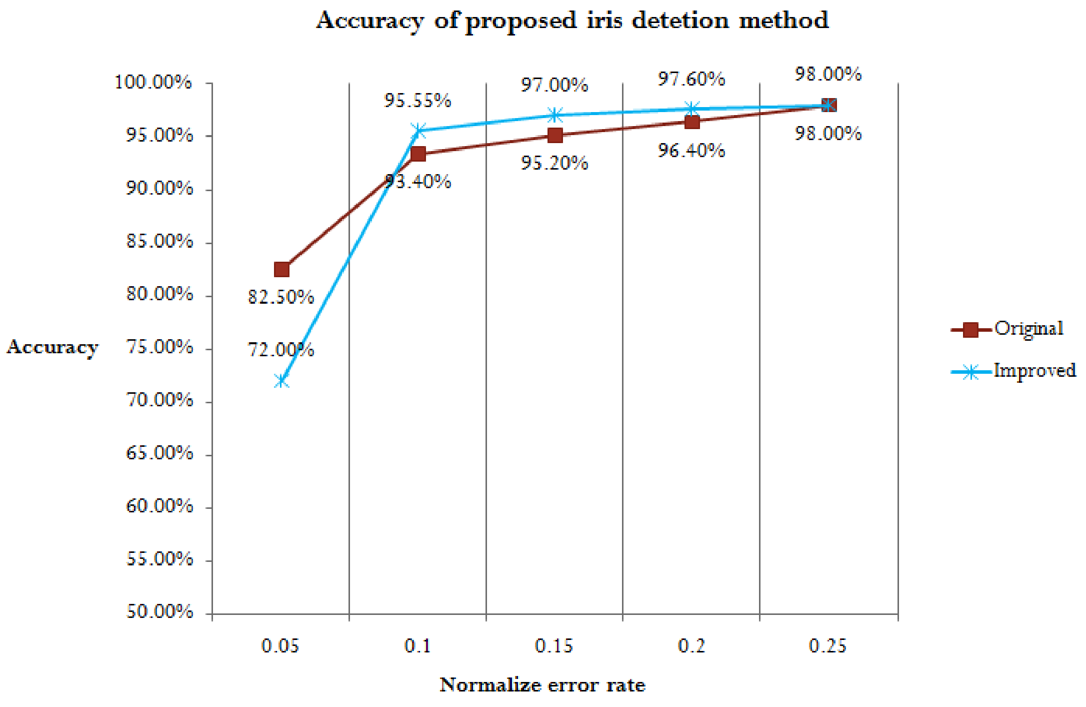


Figure 6.5: The comparison of accuracy between proposed iris detection method and the original method.

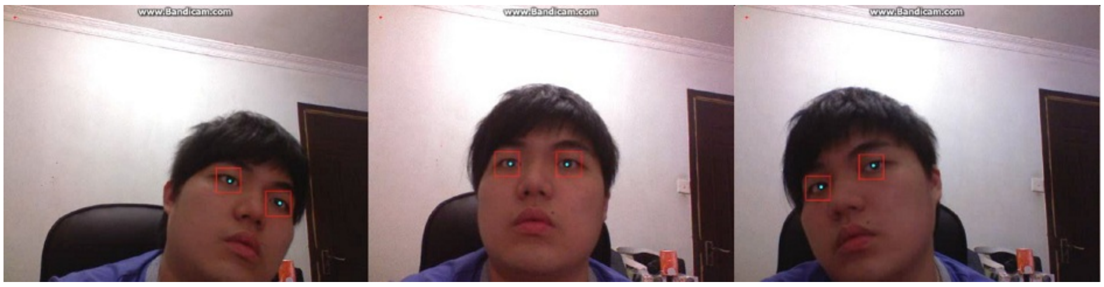
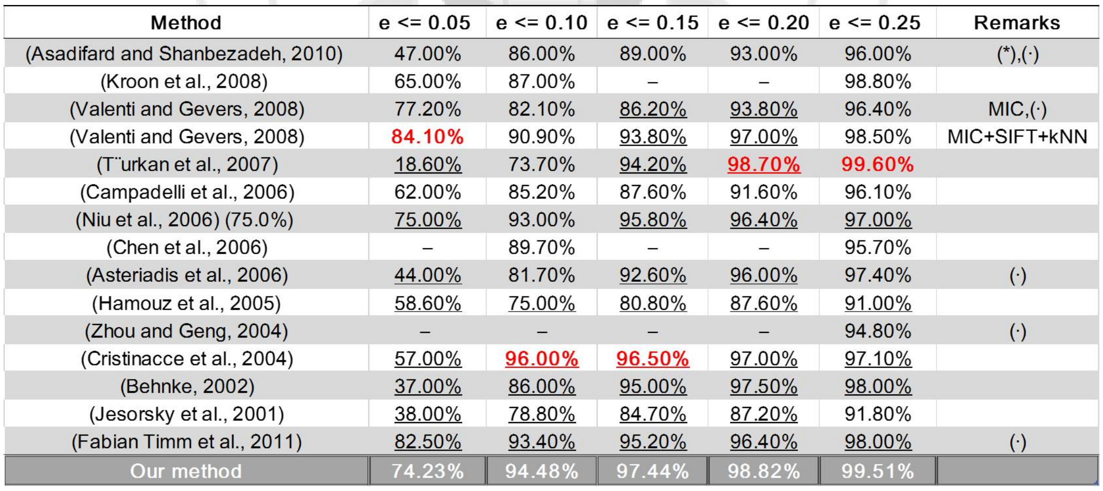


Figure 6.6: The result of iris detection method.

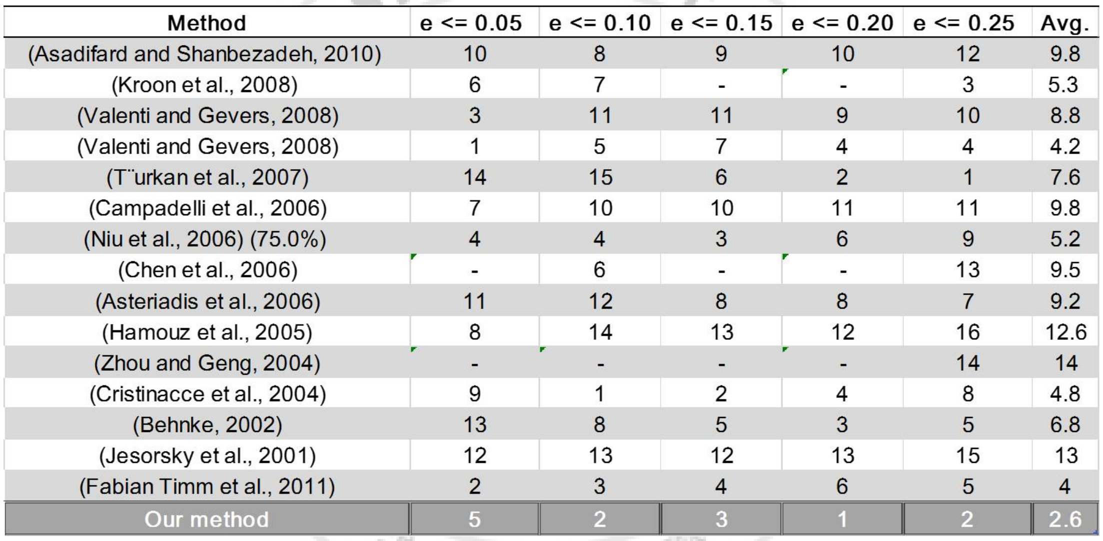
6.2 Compare to the state of art

Table 6.4 shows the performance of each method currently and the Table. 6.5 shows the rank of each method. The underline indicates that have been accurately measured from author's graphs. (∗) Images with closed eyes and glasses were omitted. (•) Methods that don't involve any kind of learning or model scheme. Since some authors didn't provide any graphical evaluation of the performance, e.g. by using a WEC curve, intermediate values couldn't be estimated - these missing values are denoted by "–".



Method	e <= 0.05	e <= 0.10	e <= 0.15	e <= 0.20	e <= 0.25	Remarks
(Asadifard and Shanbezadeh, 2010)	47.00%	86.00%	89.00%	93.00%	96.00%	(*),(·)
(Kroon et al., 2008)	65.00%	87.00%	–	–	98.80%	
(Valenti and Gevers, 2008)	77.20%	82.10%	86.20%	93.80%	96.40%	MIC,(·)
(Valenti and Gevers, 2008)	**84.10%**	90.90%	93.80%	97.00%	98.50%	MIC+SIFT+kNN
(T¨urkan et al., 2007)	18.60%	73.70%	94.20%	**98.70%**	**99.60%**	
(Campadelli et al., 2006)	62.00%	85.20%	87.60%	91.60%	96.10%	
(Niu et al., 2006) (75.0%)	75.00%	93.00%	95.80%	96.40%	97.00%	
(Chen et al., 2006)	–	89.70%	–	–	95.70%	
(Asteriadis et al., 2006)	44.00%	81.70%	92.60%	96.00%	97.40%	(·)
(Hamouz et al., 2005)	58.60%	75.00%	80.80%	87.60%	91.00%	
(Zhou and Geng, 2004)	–	–	–	–	94.80%	(·)
(Cristinacce et al., 2004)	57.00%	**96.00%**	**96.50%**	97.00%	97.10%	
(Behnke, 2002)	37.00%	86.00%	95.00%	97.50%	98.00%	
(Jesorsky et al., 2001)	38.00%	78.80%	84.70%	87.20%	91.80%	
(Fabian Timm et al., 2011)	82.50%	93.40%	95.20%	96.40%	98.00%	(·)
Our method	74.23%	94.48%	97.44%	98.82%	99.51%	

Table 6.4: The comparison of state of art.



Method	e <= 0.05	e <= 0.10	e <= 0.15	e <= 0.20	e <= 0.25	Avg.
(Asadifard and Shanbezadeh, 2010)	10	8	9	10	12	9.8
(Kroon et al., 2008)	6	7	-	-	3	5.3
(Valenti and Gevers, 2008)	3	11	11	9	10	8.8
(Valenti and Gevers, 2008)	1	5	7	4	4	4.2
(T¨urkan et al., 2007)	14	15	6	2	1	7.6
(Campadelli et al., 2006)	7	10	10	11	11	9.8
(Niu et al., 2006) (75.0%)	4	4	3	6	9	5.2
(Chen et al., 2006)	-	6	-	-	13	9.5
(Asteriadis et al., 2006)	11	12	8	8	7	9.2
(Hamouz et al., 2005)	8	14	13	12	16	12.6
(Zhou and Geng, 2004)	-	-	-	-	14	14
(Cristinacce et al., 2004)	9	1	2	4	8	4.8
(Behnke, 2002)	13	8	5	3	5	6.8
(Jesorsky et al., 2001)	12	13	12	13	15	13
(Fabian Timm et al., 2011)	2	3	4	6	5	4
Our method	5	2	3	1	2	2.6

Table 6.5: The rank of Table 1.

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.

Moreover, in the motion detection layer, the proposed iris detection method can quickly locate the user's irises accurately. Compare to the original method or state of art, the proposed method improved a lot in processing time and the accuracy is still very promising.

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.

Appendix

Listing 1: Code of transforming the format of the captured images

```
public Image<Gray, Byte> ConvertImageFormat(byte[] colorPixels, int
    width, int height)
{
    Bitmap bitmap = CreateBitmap(colorPixels, width, height);

    Image<Bgr, Byte> image = new Image<Bgr, Byte>(bitmap);
}

public Bitmap CreateBitmap(byte[] pixels, int width, int height)
{
    Bitmap bitmap = new Bitmap(width, height,
        Imaging.PixelFormat.Format32bppRgb);
    Imaging.BitmapData bmapdata = bitmap.LockBits(new
        Drawing.Rectangle(0, 0, width, height),
        Imaging.ImageLockMode.WriteOnly, bitmap.PixelFormat);
    IntPtr ptr = bmapdata.Scan0;
    InteropServices.Marshal.Copy(pixels, 0, ptr, pixels.Length);
    bitmap.UnlockBits(bmapdata);

    return bitmap;
}
```

Listing 2: Data structure of iris detection

```
// The data structure of the iris detection method
public class Position
{
    public int x;
    public int y;

    public Position() { }

    public Position(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}

public class Vector
{
    public double x;
    public double y;

    public Vector() { }

    public Vector(double x, double y)
    {
        this.x = x;
        this.y = y;
    }

    public Vector Normalize()
    {
        double magnitude = Magnitude();

        x = x / magnitude;
        y = y / magnitude;
        return this;
    }

    public double Magnitude()
    {
        return Vector.Magnitude(x, y);
    }

    public static double Magnitude(double x, double y)
    {
        return Math.Sqrt(x * x + y * y);
    }
}

public class ExPosition : Position
{
    public double magnitude;
    public Vector v;

    public ExPosition(Position pos, double magnitude, double x, double
        y)
    {
        this.x = pos.x;
        this.y = pos.y;
        this.magnitude = magnitude;
        this.v = new Vector(x, y);
    }
}
```

Listing 3: Heap operations

```
//The heap operations:
public void AddElement(Position element, double magnitude, double x,
    double y)
{
    // if the amount of the elements doesn’t reach the capacity of the
        array
    if (size < elements.Length)
    {
        elements[size++] = new ExPosition(element, magnitude, x, y);

        // build heap once it reaches the capacity
        if (size == elements.Length) BuildHeap();
    }
    else
    {
        // insert into heap when it exceeds the capacity
        InsertHeap(new ExPosition(element, magnitude, x, y));
    }
}

private void BuildHeap()
{
    for (int i = (elements.Length - 1) / 2; i >= 0; i--)
    {
        PercolateDown(i);
    }
}

private void PercolateDown(int pos)
{
    int child = pos * 2 + 1;
    ExPosition element = (ExPosition)elements[pos];

    while (child < elements.Length)
    {
        if (child + 1 < elements.Length && ((ExPosition)elements[child
            + 1]).magnitude < ((ExPosition)elements[child]).magnitude)
        {
            child++;
        }

        if (((ExPosition)elements[child]).magnitude < element.magnitude)
        {
            elements[pos] = elements[child];
            pos = child;
            child = pos * 2 + 1;
        }
        else
        {
            break;
        }
    }

    elements[pos] = element;
}

private void InsertHeap(ExPosition element)
{
    if (element.magnitude > ((ExPosition)elements[0]).magnitude)
    {
        elements[0] = element;
        PercolateDown(0);
    }
}
```

Listing 4: Thresholding

```
//Codes for thresholding:
Object synLock = new Object();
public void Thresholding(Image<Gray, float> gx, Image<Gray, float> gy,
    int threshold){
    Parallel.For(0, gx.Rows, i =>
    {
        Parallel.For(0, gx.Cols, j =>
        {
            double x = gx[i, j].Intensity;
            double y = gy[i, j].Intensity;
            double magnitude = Vector.Magnitude(x, y);
            if (magnitude > threshold)
            {
                Position pos = new Position(i, j);

                lock (syncLock)
                {
                    AddElement (pos, magnitude,x, y);
                }
            }
        });
    });
}
```

Listing 5: Calculation of mode, variance, and probability

```
// Find the mode
public EyesData calMode(EyesData[] data, int state)
{
    Dictionary<int, int> leftXcounts = new Dictionary<int, int>();
    Dictionary<int, int> leftYcounts = new Dictionary<int, int>();
    Dictionary<int, int> rightXcounts = new Dictionary<int, int>();
    Dictionary<int, int> rightYcounts = new Dictionary<int, int>();

    // Count the data
    for (int i = state; i < data.Length; i += sizes.Length)
    {
        SuccCounts(leftXcounts, data[i].leftX);
        SuccCounts(leftYcounts, data[i].leftY);
        SuccCounts(rightXcounts, data[i].rightX);
        SuccCounts(rightYcounts, data[i].rightY);
    }

    // Find the max count
    int leftXMaxCount = leftXcounts.Max(g => g.Value);
    int leftYMaxCount = leftYcounts.Max(g => g.Value);
    int rightXMaxCount = rightXcounts.Max(g => g.Value);
    int rightYMaxCount = rightYcounts.Max(g => g.Value);

    // Find the value according to the max count
    int leftXMode = leftXcounts.First(g => g.Value ==
        leftXMaxCount).Key;
    int leftYMode = leftYcounts.First(g => g.Value ==
        leftYMaxCount).Key;
    int rightXMode = rightXcounts.First(g => g.Value ==
        rightXMaxCount).Key;
    int rightYMode = rightYcounts.First(g => g.Value ==
        rightYMaxCount).Key;

    return new EyesData(leftXMode, leftYMode, rightXMode, rightYMode);
}

private void SuccCounts(Dictionary<int, int> counts, int element)
{
    if (counts.ContainsKey(element))
    {
        counts[element]++;
    }
    else
    {
        counts[element] = 1;
    }
}

// Calculate the variance
private double[] calVar(EyesData[] data, EyesData mean, int state)
{
    double leftXVar = 0;
    double leftYVar = 0;
    double rightXVar = 0;
    double rightYVar = 0;

    // Var(x) = E((x - mean)^2)
    for (int i = state; i < data.Length; i += sizes.Length)
    {
        leftXVar += Math.Pow((data[i].leftX - mean.leftX), 2);
        leftYVar += Math.Pow((data[i].leftY - mean.leftY), 2);
        rightXVar += Math.Pow((data[i].rightX - mean.rightX), 2);
        rightYVar += Math.Pow((data[i].rightY - mean.rightY), 2);
    }

    return new double[] { leftXVar / data.Length, leftYVar /
       data.Length, rightXVar / data.Length, rightYVar / data.Length };
}
```

```
// Calculate the probability according to the model
public double calProbability(int x, int mean, double var, double delta)
{
    double dx = x;
    double dy = 0;

    // factor of Gaussian distribution
    double factor = 1 / Math.Sqrt(var * 2 * Math.PI);

    // Gaussian distribution
    Func<double, double> normal = v => factor * Math.Exp(-(Math.Pow(v -
        mean, 2)) / (2 * var));

    // initiate variables
    double a = normal(dx);
    double b = 0;
    double height = delta / 2;
    double result = 0;

    // change the sign of the delta according to the variable x
    int sign = x > mean ? 1 : -1;

    // trapezoidal rule
    do
    {
        dx += sign * delta;
        b = normal(dx);
        dy = height * (b + a);
        a = b;
        result += dy;
    } while (a > 0.001);

    // two-tailed test
    return result * 2;
}
```

References

[1] BioID AG. Bioid face database. http://www.bioid.com/index.php?q=downloads/software/bioid-face-database.html. 23

[2] ASUS. Xtion pro. http://www.asus.com/Multimedia/Xtion_PRO/. 3

[3] powered by PmWiki Enlighten theme originally by styleshout, adapted by David Gilbert. Discrete integration. http://calculus.seas.upenn.edu/?n=Main.DiscreteIntegration. vi, 15

[4] Google. Atap project tango. https://www.google.com/atap/projecttango/. 4

[5] D.W. Hansen and Qiang Ji. In the eye of the beholder: A survey of models for eyes and gaze. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(3):478 – 500, 2010. 4

[6] Kyung Nam Kim and R.S. Ramakrishna. Vision-based eye-gaze tracking for human computer interface. *Systems, Man, and Cybernetics, 1999. IEEE SMC '99 Conference Proceedings. 1999 IEEE International Conference on*, 2:324 – 329, 1999. 4

[7] R. Kothari and J.L. Mitchell. Detection of eye locations in unconstrained visual images. *Image Processing, 1996. Proceedings., International Conference on*, 3:519 – 522, 1996. 4

[8] Microsoft. Kinect for windows. http://www.microsoft.com/en-us/kinectforwindows/. 6

[9] Occipital. Structure sensor. http://structure.io/. 3

[10] M.J.T. Reinders, R.W.C. Koch, and J.J. Gerbrands. Locating facial features in image sequences using neural networks. *Automatic Face and Gesture Recognition, 1996., Proceedings of the Second International Conference on*, pages 230 – 235, 1996. 5

[11] Fabian Timm and Erhardt Barth. Accurate eye centre localisation by means of gradients. *Proceedings of the Int. Conference on Computer Theory and Applications (VISAPP)*, 1:125 – 130, 2011. 8

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

[13] R. Valenti and T. Gevers. Accurate eye center location and tracking using isophote curvature. *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1 – 8, 2008. 5

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

[15] wikipedia. Normal distribution. `http://en.wikipedia.org/wiki/Normal_distribution`. vi, 13

[16] A.L. Yuille, D.S. Cohen, and P.W. Hallinan. Feature extraction from faces using deformable templates. *Computer Vision and Pattern Recognition, 1989. Proceedings CVPR '89., IEEE Computer Society Conference on*, pages 104 – 109, 1989. 4