Design and Realization of the Gesture-Interaction System Based on Kinect

Jie Xu
Abstract

Design and Realization of the Gesture-Interaction System Based on Kinect

Jie Xu

In the past 20 years humans have mostly used a mouse to interact with computers. However, with the rapidly growing use of computers, a need for alternative means of interaction has emerged. With the advent of Kinect, a brand-new way of human-computer interaction has been introduced. It allows the use of gestures - the most natural body-language - to communicate with computers, helping us get rid of traditional constraints and providing an intuitive method for executing operations.

This thesis presents how to design and implement a program to help people interact with computers, without the traditional mouse, and with the support and help of a Kinect device (an XNA Game framework with Microsoft Kinect SDK v1.7).

For dynamic gesture recognition, the Hidden Markov Model (HMM) and Dynamic Time Warping (DTW), are suggested. The use of DTW is being motivated by experimental analysis. A dynamic-gesture-recognition program is developed, based on DTW, to help computers recognize customized gestures by users. The experiment also shows that DTW can have rather good performance.

As for further development, the use of the XNA Game 4.0 framework, which integrates the Kinect body tracking into DTW gesture recognition technologies, is introduced.

Finally, a functional test is conducted on the interaction system. In addition to summarizing the results, the thesis also discusses what can be improved in the future.

Keywords: Kinect, gestures, human-computer interaction, DTW, XNA.
Acknowledgement

First of all, I would like to give my sincerest gratitude to my tutor, Prof. Buyang Cao for his greatest supports and guidance. His expertise, suggestions and advice always give me the most benefits and motivate me excel myself.

Then, I would like to thank my classmate, Yuanqing Luo. It was him that helped me move forward and taught me everything he knew when I firstly stepped into the Kinect area and knew nothing about it.

I also thank my parents for their great sacrifice and silent support no matter when I was in Sweden or am in China. They are the first reason and propulsion I get where I am today.

Moreover, I must thank Tongji University and Uppsala University for giving me such perfect stage helping me growing up and learning knowledge. They not only teaches me the latest knowledge and technologies but also make me more and more independent and mature.

Last but not least, I need to thank my reviewer and examinor, Lars Oestreicher and Ivan Christoff. Thanks so much for their great patience and valuable suggestions which drive me make so much progress.
Content

I. Introduction ............................................................................................................................ 1
   1.1 Research Background ............................................................................................. 1
   1.2 Human-Computer Interaction ............................................................................. 1
   1.3 Thesis Structure ......................................................................................................... 2

II. Kinect and Related Knowledge ....................................................................................... 3
   2.1 Kinect Introduction .................................................................................................. 3
   2.2 Kinect Principles ........................................................................................................ 4
   2.3 Kinect Applications and Research Situation .................................................. 5
   2.4 Microsoft Kinect SDK .............................................................................................. 7
   2.5 Double Exponential Smoothing ......................................................................... 9
   2.6 Chapter Summary ................................................................................................. 11

III. Dynamic Gesture Recognition ................................ ...................................................... 12
   3.1 Algorithms on Dynamic Gestures Recognition ......................................... 12
   3.2 DTW vs. HMM ......................................................................................................... 13
   3.3 Gesture Recognition Based on DTW ............................................................. 14
   3.4 Experiment Design ................................................................................................ 17
   3.5 Result Analysis ........................................................................................................ 19
   3.6 Chapter Summary ................................................................................................. 23

IV. Design of the System ....................................................................................................... 24
   4.1 Requirements .......................................................................................................... 24
   4.2 Module Design ....................................................................................................... 24
   4.3 Chapter Summary ................................................................................................. 27

V. Realization and Test of the System ............................................................................. 28
   5.1 Hardware and Software Environment ........................................................... 28
   5.2 Establishment .......................................................................................................... 28
   5.3 Kinect Data Processing ........................................................................................ 28
      5.3.1 Kinect Initialization ................................................................................... 28
      5.3.2 Kinect Data Stream Processing ............................................................ 29
      5.3.3 User Tracking and Gestural Commands ........................................... 31
   5.4 XNA Processing ...................................................................................................... 32
      5.4.1 XNA Compendium .................................................................................... 32
      5.4.2 XNA Content ............................................................................................... 33
I. Introduction

1.1 Research Background

There used to be an advertisement on the internet that when you eat your breakfast, the wall you face shows the weather, news, and calendar, etc. You just wave your hand in the air without any touching, the wall changes the displaying contents as you wish. When you cook in the kitchen, an invitation flies into your screen and you just need to do some click actions with your not-that-clean hands without any tool and you can reply it to your friend. This is exactly the idea of a house of future. It describes a thing that computers seem like don’t exist around us, but actually they do and they are everywhere. This tells us that maybe we can interact with computers in a more natural and human-like way.

The traditional interaction tools, such as mouse and keyboard have dominated the market for more than 20 years. However, the appearance of this advertisement indicates that something new is beginning to happen in humans’ brains. The old, traditional ways can never satisfy our needs, and some better ways need to be brought out.

With the appearance of Kinect device, the magical life in the advertisement begins to be closer to us which aims to create a revolution in human-computer interaction.

1.2 Human-Computer Interaction

HCI (human-computer interaction) is a multi-field science which researches the relationship between users and computers. In a simple way, it can be regarded as a kind of user experience.

In the past decades, the ways human interacting with computers has changed a lot, from the case-like big machine, to personal computer and mobile computers. According to the research of Mark Weiser and John Seely Brown [1] from Xerox Corp, the history of interaction can be divided into four periods. They are big machine, personal computer, mobile computing and...
pervasive computing. Nowadays, we are in personal computer and mobile computing periods. In the computer period, we interact with them on graphics via mouse and keyboard. In the mobile period, we interact with them via touching. According to the research results, the pervasive computing period possibly begins from the twenties of 21st century which we are about to enter. In pervasive computing period, technologies no longer demand users to know the details of technologies and only need to care about what content they want to acquire and what events they want to realize. In daily life, humans can enjoy the convenient service provided by computer unconsciously via the most natural actions. In this period, humans’ voice and gestures begin to take the place of mouse and keyboard and humans’ instinctive behaviors are much more integrated into human-computer interaction. And fortunately, this is exactly the way the project we are about to discuss works on.

1.3 Thesis Structure

This thesis consists of these chapters,

First is introduction. It generally states the researching background and the history and future of its related area, human-computer interaction.

In the second chapter, it introduces the main hardware, Kinect, its working principles, its current state of researching and applications. Then it also specifies the history of SDK and the smoothing algorithm in it.

In the third chapter, it compares the performance of the two algorithms, HMM and DTW, for dynamic gestures recognition. And it emphasizes introducing the principles of DTW, designs and realizes a gestures recognition program based on DTW and analyze the experimental results.

In the fourth chapter, it generally presents the requirements of the interaction system and mainly states the framework of the system and designs of every module.

In the fifth chapter, it shows the whole developing process of the interaction system and the integration of Kinect and XNA game framework. Then it posts a complete test on the system.

In the last chapter, it summarizes the thesis and the whole project. It also discuss what improvements are worthy researching in future.
II. Kinect and Related Knowledge

2.1 Kinect Introduction

With the development of the science technology and pervasive computing, many sensing devices emerge in the market, such as Kinect, EyeToy etc. However, Kinect can be regarded as the most mature device which can capture and recognize body movement and voice.

In 2010, XBOX360 device was named Kinect officially by Microsoft. Kinect can be considered as a 3D camera which could capture real-time image, recognize depth information and body skeletons, etc. On the same year, this technology was developed onto the production stage. Then the Kinect device was swiftly spread in the market and became one of the most popular gaming device. By 2012, Microsoft launched the project of Kinect for Windows which solved the restriction that Kinect could only be customized on XBOX360 platform. With the later publishing of the Kinect for Windows SDK, it opened the prelude of software development based on Kinect. However, due to Microsoft’s authorization, this technology could only be used for education and scientific research.

Kinect consists of RGB camera, CMOS depth camera, infrared transmitters, array microphones and base motor. Its horizontal angels of view is 57 degrees, the vertical angles of view is 43 degrees and the rotation angle range is $\pm 27$ degrees. It can detect at most six people, but currently could only tracks two active users which that a Kinect device could track two users at most at the same time.
As the image above shown, among the three cameras, the middle one is RGB camera, which can capture color images with a resolution of 640*480 pixels. The left one is infrared emitter and the right is the depth camera which can be used to capture depth images with the resolution of 320*240 pixel. The sampling frequency of Kinect image is 30 frames per second and theoretically effective depth distance is from 0.8 meters to 4 meters.

2.2 Kinect Principles

Infrared sensor and the depth camera are the main devices to obtain the depth image of the real world. Its core technology is called Light Coding [36]. Compared with the previous familiar technology, TOF (Time of Flight), the PS1080 chip is used in Light Coding to control the infrared light source. It uses continuous light rather than other pulse light illuminate and casts “stereoscopic codes” to replace the traditional two-dimensional cyclical image coding. The so-called three-dimensional coding records the real space using a light source. And the sensor camera in this technology is just an ordinary CMOS which can highly reduce the cost. This is also the reason why Kinect can quickly be popular throughout the market.

The light coding technology takes advantages of the principle of laser speckle rather than the geometric relationship of objects in space. When the surface is irradiated with a laser, the spatial scattering phenomenon will be formed and the light field near the surface will produce a random laser speckle (Laser Speckles). In optics, the laser speckle pattern formed by laser irradiation is highly random. So in one space, the speckle patterns produced by objects at different distances from the laser light source after the laser irradiation are never the same. This means that laser speckle has uniqueness. Therefore we labels spatial codes on every region of the space and make the system record the corresponding codes. Therefore when the system gets the speckle pattern of an object and matches it with the recorded codes, the position of object in space can be determined. When there is an object in space in a certain position, the position in the image will appear as a peak. The system superimposes these peaks and do interpolation calculation on them in order to achieve the three-dimensional reconstruction of the shape of the entire scene. As
described in patents of PrimeSense, at a certain distance, intercept a reference flat and record the speckle pattern of the reference flat \(^{[35]}\), which is the core of the optical Kinect coding technology.

![Image](image.png)

**Img.2.2. the left is laser speckles, the right is depth camera effects**

2.3 Kinect Applications and Research Situation

In 2008, Bill Gates had raised the concept of “natural user interface” \(^{[36]}\) and he predicted that the traditional keyboard and mouse would be replaced by touching, gestural, visual or voice ways in the next few years.

With Kinect becoming more and more mature, this new interactive way is gradually applied to the real world besides gaming. It allows users to interact more naturally with computer. For example:

(a) **PPT control**: Presentation is essential in company conferences and school education. Holding a mouse or controller to do flip, playback controls seems kind of inefficient or unnatural. While the skeletal tracking function of Kinect can help us identify the user’s posture at any time, we can let Kinect decide what should do next during the PPT presentation according to the definition of body gesture.

(b) **Application in surgical hospital**: in hospital operating room, the doctors need to use computer to watch real-time information of the patient. Because of the sterile gloves, he can’t touch mouse. He has to ask others help him do every operation on computer, which seems much inefficient. With the help of Kinect, the surgeon can easily observe the real-time information by waving hands. This method improves the efficiency and bring more convenience to surgery.

(c) **3D model reconstruction**: with the help of Kinect’s depth camera to obtain depth information, we can easily achieve capturing and
displaying three dimensional model after scanning the object.

(d) Virtual Dressing: connect the Kinect and a giant screen, through the control of the computer, the customers just need to wave their hands in front of the screen instead of change clothes in fitting rooms and will be able to see themselves on the screen wearing new clothes.

(e) Game Applications: XBOX360 has demonstrated the success of Kinect in the game field, waving hands in the air to cut the fruit "Fruit Ninja"; dancing without dance brackets "Dance Revolution"; playing tennis without rackets "Power Tennis"; and so on.

All above show that, the appearance of Kinect is beginning to change humans’ life in a new way which makes computer adapt to humans’ habits and lead interaction to a unprecedented area.

While the earlier version of Kinect only focused on XBOX360 platform and Kinect for Windows was released in 2012, the Kinect-based applications are still mostly limited to games and entertainment. With the new release of Kinect for Windows, the value of Kinect device is gradually reflected in health care, education, office etc. Nowadays the exploration of domestic and foreign developers on Kinect is still far from comprehensive and practical. And newly
researches are generally focused on some particular applications, such as PPT control, gestural games and so on. They only put most attention on some certain specific scenarios but has not yet integrated them into a complete system to let Kinect devices be completely able to replace the traditional keyboard and mouse.

So far, Kinect and related technologies have reached a quite mature degree. With the new release of Kinect SDK v1.7, the new actions of push, grab and release are added. The enthusiasm on this new field will definitely arrive at an unprecedented height before long. Just like Bob Heddle, the global director of Microsoft Kinect for Windows institution said, the natural way of computer communication supporting gestures and language are just like communicating with humans. We believe that this has important significance for the computer technology [2].

2.4 Microsoft Kinect SDK

Kinect SDK released by Microsoft supports audio and the base engine, is able to track 20 skeleton points of the whole body and can support connecting four Kinect devices at the same time. Moreover, it is easy to install and is best compatible with the Windows operation system and .Net platform. It can also be integrated into Visual Studio developing environment. But it still has its drawbacks. It is only for non-commercial use and can’t track some specific place of the body, like hands. And it only support Windows 7 system or newer.

In February 2012, Kinect for Windows SDK v1.0 was release and it accomplished,

- Support at most four devices connecting with computer simultaneously.
- Near Mode allows depth camera recognizing things ranging from 0.4 to 3.5 meters.
- Allow developers choose the tracking users.
- Improve audio recognition.

About the later SDK 1.5 and 1.6, they mainly focused on improving the performance on every respect.

However, on March 18, 2013, Kinect SDK v1.7 which was the most
important update so far, was released. First of the updates is Kinect Fusion. It uses depth camera to capture and construct complete 3D models by constant snapshot in real time. Second is that it is allowed to connect with Matlab and OpenCV. It can transform the formats of data to fit development in Matlab and OpenCV. Third and the most important to our project is Kinect Interaction. It is the beginning of the research of Microsoft in interaction. It makes Kinect enable to recognize Press, Grip and Release the three gestures. Though there are only three simple gestures, it begins to support developers do something about interaction, such as click, scroll, drag, and so on. And it also shows that Microsoft’s confidence of Kinect in human-computer interaction.

This project takes Kinect SDK as the developing kit.

![Img.2.5. real-time 3D reconstruction of Kinect Fusion](image1)

![Img.2.6. Kinect Interaction: Press, Grip, Release](image2)

![Img.2.7. the framework of Kinect Interaction](image3)
2.5 Double Exponential Smoothing

In some cases and projects based on old-version SDK, due to lacking of smoothing methods, there were always some leaps happening in the skeleton movement. When users did some long-distance movement, there was not much obvious influence. But when users just did some small adjustments, the experience would be fatally terrible. So smoothing algorithms become so much important in Kinect development, especially those which have the abilities to predict the tracking data. The Kinect system needs to track the position data in every frame in real time. Without these predicting algorithms, the data the computer is calculating and the data the screen is displaying would have little de-synchronization which could bring horrible impact on user experience. Fortunately, in Kinect SDK v1.7, Microsoft helps us accomplish the double exponential smoothing.

In current Kinect device, it already has the ability to measure velocity and acceleration. So compared with Kalman Filter algorithm, Double Exponential Smoothing is a better choice which can provide a faster performance for data smoothing. Double Exponential Smoothing is derived from Simple Exponential Smoothing, which was firstly proposed by Holt.

Exponential smoothing algorithm is an important and effective way dealing with temporal sequence data which can not only smooth data, but also predict future changes. This technology utilizes all the historical data and related information to weight new data based on the principle of later data being more important than earlier one. It can greatly decrease the possibility of abnormal data happening with the pre-condition that the preciseness of data doesn’t be decreased much. In the exponential smoothing, it computes in a recursive way which can make it a very efficient algorithm.

Firstly, simple exponential smoothing was proposed by Robert Goodell Brown in 1956 [20]. The formulae are following,

\[ s_1 = x_0 \]  \hspace{1cm} (3.1)

\[ s_t = \alpha x_{t-1} + (1 - \alpha)s_{t-1} = s_{t-1} + \alpha(x_{t-1} - s_{t-1}), \quad t > 1 \]  \hspace{1cm} (3.2)

And, \( \alpha \) is smoothing factor, \( 0 < \alpha < 1 \). Smoothing result \( s_t \) is the weighted average of last observation \( x_{t-1} \) and the last smoothing result \( s_{t-1} \). The closer \( \alpha \) is to 1, the larger the weight of the last smoothing result is and the less the
smoothing effect is. And vice versa. The selecting of the value of $\alpha$ doesn’t have a standard.

However, when there is a trend in the data, such as direction, acceleration, simple exponential smoothing couldn’t perform well. But double exponential smoothing can ideally deal with this situation. It has two different smoothing variables to process the original sequences.

In double exponential smoothing, the original sequence is $\{x_t\}$, the beginning time is $t = 0$. At every time node $t$, $\{s_t\}$ is the sequence after smoothing, $\{b_t\}$ is the best evaluation of the trend, $\{f_{t+m}\}$ is the output result representing the predicted result for the time node $t + m$, given that $m > 0$.

The formulae are following,

$$s_1 = x_1$$  \hspace{1cm} (3.3)

$$b_1 = x_1 - x_0$$  \hspace{1cm} (3.4)

When $t > 1$

$$s_t = \alpha x_t + (1 - \alpha)(s_{t-1} + b_{t-1})$$  \hspace{1cm} (3.5)

$$b_t = \beta(s_t - s_{t-1}) + (1 - \beta)b_{t-1}$$  \hspace{1cm} (3.6)

Formula (3.5) is the smoothing calculation based on time and formula (3.6) is the smoothing calculation based on trend, given that $0 < \alpha, \beta < 1$.

The last smoothing result is,

$$f_{t+m} = s_t + mb_t$$  \hspace{1cm} (3.7)

$m$ is the following time slots that are predicted.

Double exponential smoothing plus the last trend value $b_{t-1}$ and the last smoothing value $s_{t-1}$ to amend the delay effect of the simple smoothing. And it uses the difference of the two adjacent smoothing values representing the trend. With the existence of random interference, formula (3.6) is used to smooth the trend. These can help ensure the feasibility and correctness double exponential smoothing.

Current Kinect devices can measure the direction, velocity and acceleration of users’ movement, so it is easy to get the trend vector $\overrightarrow{b_t}$ after getting the skeleton vector $\overrightarrow{s_t}$ and calculating it with its direction, velocity and acceleration attributes. Then double exponential smoothing can be as followed.

$$\overrightarrow{s_t} = \alpha \overrightarrow{x_t} + (1 - \alpha)(\overrightarrow{s_{t-1}} + \overrightarrow{b_{t-1}})$$  \hspace{1cm} (3.8)

$$\overrightarrow{b_t} = \beta(\overrightarrow{s_t} - \overrightarrow{s_{t-1}}) + (1 - \beta)\overrightarrow{b_{t-1}}$$  \hspace{1cm} (3.9)
It can provide us a much good way eliminating the jitter effect in movement. There is no standard method setting the smoothing factor $\alpha$ and trend smoothing factor $\beta$.

2.6 Chapter Summary

In this chapter, firstly it introduces the Kinect device and the principles how it works. Then it introduces the current state of the research and applications of Kinect and states the history of Kinect SDK and the updates of the latest one. At last, it specifies the double exponential smoothing newly added in Kinect SDK v1.7.
III. Dynamic Gesture Recognition

3.1 Algorithms on Dynamic Gestures Recognition

In Microsoft Kinect for Windows SDK, there are only three gestures can be recognized as operations, which is quite insufficient. Consequently, it is much necessary to implements the function that users can add more customized gestures to offer better user experience and expansibility.

Gestures can be divided into two categories, dynamic and static. When recognizing static gestures, we always accomplish it by these procedures: segmentation of gesture regions, extraction of eigenvalues and classification of gestures according to eigenvalues. The most common methods are Bayesian Network, SVM (Support Vector Machine), etc. In this project, considering that Microsoft has provided three static gestures for us, I put higher priority to dynamic gestures recognition.

Compared with static gestures, the most significant feature of dynamic gestures is the temporal and spatial changes, so in general, we model a dynamic gesture into a moving path in the coordinate system. As to a dynamic gesture, even though one user does the same one for several times, the scopes and velocities are always different which would result in temporal and spatial difference. So it is the most important to finding a way to precisely locate the starting and ending points and match the points on the track. For dynamic gesture recognition, the most common methods are Neural Network, Hidden Markov Model and Dynamic Time Warping.

Neural Network: it is a scientific calculation method which is enlightened by bio-neural network. The network is consisted of many “neuron” points and each neuron point is an output function. Between every two points there is a weighted value to represent the memory of the bio-nerves. Neural network is a way to simulate the logical representation in nature.

Hidden Markov Model: it is a method to describe the Markov Model with unknown variables. In HMM, the information of each state point is available while its output which is dependent from itself is available. And each state
point has a possible probability distribution of its output. So we build statistic models for observable outputs and use massive computation to get the possible distribution situations of unobservable state points. HMM can effectively recognize complex gestures, but it needs massive computation and plentiful training examples.

Dynamic Time Warping: compared with HMM, DTW is much simpler and it needs less computation and training samples. However, its recognition rate for complex gestures is lower.

3.2 DTW vs. HMM

As for dynamic gestures recognition, DTW and HMM are definitely the most widely used algorithms.

Before we formally make a decision on which algorithm is more suitable to be used in our project, I get help from an open-source project, “Speech Recognition System Based on Matlab”[34], to do a test experiments on the two algorithms in Matlab. In that system, I use simple voice dataset to do massive experiments to evaluate the respective performance of the two algorithms on sample matching. Though the dataset is voice instead of gestures, it is acceptable considering that different data types wouldn’t make any differences on the results. After numerous experiments, I record the results and analyze the relationship between recognition rate and numbers of training samples in Matlab.

The image below is the statistic result in Matlab.
I record the results respectively when the number of training samples grows from 0 to 50. According to image above, we can conclude that DTW can provide 90% of recognition rate after only 4 training samples and after 27 training samples DTW can provide 98% of recognition rate. However, HMM needs 45 training samples to provide the same-level performance.

Considering that interaction gestures should be simple and it would be a trouble for users to do much training, so DTW is definitely the better choice in this project.

3.3 Gesture Recognition Based on DTW

Dynamic time Warping is a technology on measuring the similarity between two temporal sequences which vary in time. In the analysis, the two temporal sequences differ from each other in temporal range and velocity. DTW can effectively resolve these restrictions and precisely estimate the similarity.

A dynamic gesture can be regarded as a series of hand movement with temporal sequences. For a single gesture, almost no one can do it in the same time and the same moving path. As a consequence, the gesture features recognized at the same time point can be different and the start and end time
points for same feature can also be different every time. Therefore, we can’t calculate the distance of the two sequences in the ordinary ways, such as K-means to recognize the dynamic gestures.

In DTW algorithm, the all eigenvalues of a sample sequence can be denoted as $X = \{x_1, x_2, x_3, \ldots, x_M\}$, and the all eigenvalues of sequence to be recognized is $Y = \{y_1, y_2, y_3, \ldots, y_N\}$. In general, $M$ and $N$ are different positive integers. Giving the definition that the distance between one element $x$ in $X$ and one element $y$ in $Y$ as $c = c(x, y)$ and all the distances constitute the distance sequence $C = \{c_1, c_2, c_3, \ldots, c_L\}$, given $L$ is the length of warping of sequences $X$ and $Y$.

Each distance function $c$ calculates the distance in a pair of $x$ and $y$ and the smaller result is, the more similar $x$ and $y$ are. After calculating the distance of every pair $(x_m, y_n)$, given that $m \in [1, M], n \in [1, N]$, then we can get a $M \times N$ distance matrix. Consequently we can solve this problem by finding a path on the matrix in which the sum of all the distances through the path is smallest that represents the highest similarity.

We define a sequence $P = \{p_1, p_2, p_3, \ldots, p_L\}$ is a path on the matrix, given that $p_l = c(x_i, y_l)$, and in gesture recognition, each sequence $P$ should satisfy several requirements below,

- **Continuity**: the distance of each point in the matrix should be calculated to ensure the accuracy of the gestures matching.

- **Monotonicity**: points in the gesture sequence are ordered in time, so each point on the path should abide by the temporal orders, which means $x_1 \leq x_2 \leq \cdots \leq x_L, y_1 \leq y_2 \leq \cdots \leq y_L$.

- **Length of steps**: when stepping forward along the path, the unit changes along the direction of $M$ and $N$ can’t be more than 1 respectively, which means $(p_l - p_{l-1}) \in \{(1,0), (0,1), (1,1)\}$, given that $l \in [2, L]$.

These three restrictions guarantee that all elements can be calculated and compared un-repeatedly. And the direction of the path always keeps monotonic and never turns backward.
The image below shows the warping restrictions.

The image below shows a valid distance path.

As for a valid path of sequences $X$ and $Y$, the total distance is,

$$C_p(X, Y) = \sum_{t=1}^{L} c(x_t, y_t)$$ (4.1)

Among all the distances, DTW distance, $DTW(X, Y)$, is the one with the smallest value,

$$DTW(X, Y) = C_{p_{\text{min}}}(X, Y) = \min\{C_p(X, Y) | p \text{ is a valid warping path}\}$$ (4.2)

As we mentioned above, we can define that $D(X, Y)$ is a $M \times N$ matrix whose unit is filled with the distance of elements $x$ in $X$ and $y$ in $Y$. This matrix should satisfy the following formula:

$$D(m, 1) = \sum_{k=1}^{m} c(x_k, y_1), \quad m \in [1, M]$$ (4.3)
\[ D(1, n) = \sum_{k=1}^{n} c(x_1, y_k), \quad n \in [1, N] \]  

(4.4)

Finally, we can get the DTW distance as:

\[ D(m, n) = \min\{D(m-1, n-1), D(m-1, n), D(m, n-1)\} + c(x_m, y_n) \]  

(4.5)

\( D(m-1, n-1), D(m-1, n) \) and \( D(m, n-1) \) represent the valid precedent time point.

According to the formula (4.5), searching begins from point (1,1), recurs and compares all the valid distance values to get the best path. We can also conclude that each time it only calculates one unit in row or column of the matrix, therefore the time complexity of the algorithm is \( O(MN) \). The pseudo-codes are followed [24],

```c
int DTWDistance(x: array [1..m], y: array [1..n])
{
    DTW := array [0..m, 0..n]
    for i := 1 to m
        DTW[i, 0] := \infty
    for i := 1 to n
        DTW[0, i] := \infty
    DTW[0, 0] := 0
    for i := 1 to m
        for j := 1 to n
            distance := d(s[i], t[j])
            DTW[i, j] := distance + minimum(DTW[i-1, j], DTW[i, j-1], DTW[i-1, j-1])
    return DTW[m, n]
}
```

3.4 Experiment Design

DTW can be considered as a sample-matching algorithm, so before applying it to recognize gestures, the system needs to train samples and memorize them first. Then the system can calculate the similarity between gestures and samples. So the experiment is divided into two phases: sample training and gesture recognition.
The training phase generally has 4 parts, gathering video, processing gestures, recording coordinates and saving into samples library. We only track eight skeleton points, left hand, left wrist, left elbow, left shoulder, right hand, right wrist, right elbow and right shoulder. Because everyone is different from each other in body, we can’t directly use users’ skeleton data to do the training and recognition. We need to normalize the skeleton data first.

When users do gestures, the most stable skeleton among those eight skeleton points are left and right shoulders. So we determine the mid-point of the line connecting left and right shoulders as the base point.

To each skeleton point \( S \), we can normalize it by the formula below,

\[
\overline{S}_{\text{norm}} = \frac{\vec{S} - \vec{C}}{|\vec{L} - \vec{R}|}
\]

\( \vec{S} \) is the vector of the skeleton \( S \) before normalization; \( \overline{S}_{\text{norm}} \) is the vector of the skeleton \( S \) after normalization; \( \vec{C} \) is the vector of base point; \( \vec{L} \) is the vector of left shoulder; \( \vec{R} \) is the vector of right shoulder. By this method we can normalize the skeleton points onto the coordinate system with the base point, which is the mid-point of line connecting left and right shoulders.

At last, we record the coordinates of each tracking skeleton points in every frame and save them into the sample library.
On the other hand, the gesture recognition phase has these 5 parts, gathering video, processing gestures, samples in library, DTW recognition and output. We use DTW algorithm to match the input gesture with the samples in the library and calculate the similarity, then display the output.

By means of the same process in training part, after normalizing input gestures, we match them with samples in library, calculate the distances and record the DTW one (which value is smallest). If the value of DTW distance is below a threshold value, we consider it as recognizing successfully. Otherwise, it means that the input fails to match with any sample and it is an unknown gesture.

```csharp
for (int i = 0; i < sequences.Count; i++)
{
    var example = (ArrayList) sequences[i];
    if (Dist((double[]) seq[seq.Count - 1], (double[]) example[example.Count - 1]) < Threshold)
    {
        double d = DTW(seq, example) / example.Count;
        if (d < minimumDist)  {
            minimumDist = d;
            classification = (string)labels[i];
        }
    }
}
return (minimumDist < globalThreshold ? classification : "UNKNOWN");
```

3.5 Result Analysis

I refer to an open-source project “KinectDTW” [30] and according to my own requirements in which I amend, improve and re-design some functions under the environment of Kinect SDK v1.7. Finally I implement a program which can do DTW training and recognition.

In the program, there is a list showing all the available gestures. When choosing one gesture and clicking “Start Training”, user will have one period of time of 30 frames to perform a gesture and then the program will record it. User has two chances to make this training. If user do the gesture for the second, the former one will be covered. Otherwise, the program will still keep the first one. After completing all the training, the user can click the “Save” to
save all the samples into the library as a txt file.

In the process of recognition, the user can click “Load Model File” to choose the samples library. Then when the user performs a gesture, if it is matched with a sample in the library, the program will display the gesture sample’s name on the window. Otherwise, the program will show the result of “UNKNOWN”.

In this experiment, I have trained eight gestures, “Left hand wave”, “Right hand wave”, “Left hand push up”, “Right hand push up”, “Two hands zoom in”, “Two hands zoom out”, “Circle wave” and “Both hands eight shape”.

The image below is the training process of the gesture “Circle wave”. I am drawing an anticlockwise circle with my right hand.

![Training process with gesture “circle wave”](image)
Here are the results of every gesture,

**Img.4.8. Gesture is not matched**

**Img.4.9. Results of “Right hand wave” and “Left hand wave”**

**Img.4.10. Results of “Right hand push up” and “Left hand push up”**
After finishing all the training, my friend (whose physique is different from me) and I both do 50 times tests on each gestures, totally $8 \times 50 \times 2 = 800$ times. The statistics shows as followed,

<table>
<thead>
<tr>
<th>Gestures</th>
<th>Correct</th>
<th>No Response</th>
<th>Wrong</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left hand wave</td>
<td>97%</td>
<td>3%</td>
<td>0%</td>
</tr>
<tr>
<td>Right hand wave</td>
<td>96%</td>
<td>4%</td>
<td>0%</td>
</tr>
<tr>
<td>Left hand push up</td>
<td>98%</td>
<td>2%</td>
<td>0%</td>
</tr>
<tr>
<td>Right hand push up</td>
<td>98%</td>
<td>2%</td>
<td>0%</td>
</tr>
<tr>
<td>Two hands zoom in</td>
<td>94%</td>
<td>6%</td>
<td>0%</td>
</tr>
<tr>
<td>Two hands zoom out</td>
<td>93%</td>
<td>6%</td>
<td>1%</td>
</tr>
<tr>
<td>Circle wave</td>
<td>92%</td>
<td>6%</td>
<td>2%</td>
</tr>
<tr>
<td>Both hands eight shape</td>
<td>79%</td>
<td>15%</td>
<td>6%</td>
</tr>
</tbody>
</table>

Fig.4.13. Statistic result of DTW recognition

From the results, we can see that it has really high recognition rate for the simple dynamic gestures, such as “wave”, “push up”, etc. And compared with
other algorithms, DTW is much simpler and easier to achieve and it doesn’t need much sample training. So it can provide better scalability in future and users can add new customized gestures more easily. However, when the gestures are getting more and more complex, the performance of DTW is getting worse and worse. For example, when I do the gesture of “Both hands eight shape”, the recognition rate decreases to 79% and the false recognition rate increases to 6%. During the experiment, I also notice that if I don’t perform gestures standardly, especially “Both hands eight shape”, it would occasionally be recognized falsely as “Two hands zoom in”. As a result, we can see that when a gesture is complex, it is possible to be recognized falsely as another relatively similar gesture. To sum up, the performance of the experiment is much acceptable in general, but we can’t ignore the fact that the sample library is small and the amount of samples is also simple. When the library is getting larger and the interference among gestures is getting stronger, the false-recognition rate will probably increase. This is an issue that developers and users must take into consideration.

According to the result, it can be concluded that DTW algorithm is simple and efficient, can provide the interaction system good scalability and support adding new customized gestures. Though it performs not so well when gestures are complex, considering that most users liking using simple gestures for interaction, DTW is definitely a feasible and effective method in gesture recognition.

3.6 Chapter Summary

In this chapter, firstly it briefly introduces some algorithms in the field of gesture recognition and compares the performance of HMM and DTW in Matlab. Then it mainly illustrates the principles, advantages, and application of DTW. At last, an experiment is designed and implemented to measure and analyze the performance of DTW.
IV. Design of the System

4.1 Requirements

The establishment of the interaction system is enlightened by the application of Kinect for Windows in office area. I need to figure out what it can bring to market and whether people would like it before designing and developing it.

The interaction system needs to track a user, locate his right hand and associate it with the mouse and keyboard in the computer to make sure that user can control the computer immediately. The system is able to catch the positional change of users’ skeletons, the gestures of left and right hands and the customized dynamic gestures in real time. In the meantime, the system will recognize and response to the activities immediately while sensing them.

With the support of Kinect device, the system can let users control the mouse and operate some keyboard events in long distance via no touching and no medium. This could help users get rid of space and tool restrictions to interact with computers only via gestures.

I posted a survey on the internet and took a questionnaire in campus and there were 3 options participants could take. A was “like and support the system and want to buy it”. B was “not so interested in it and it depends whether to buy it or not”. C was “it is meaningless and won’t buy it”. Totally 106 answers were received and A, B and C occupied about 70%, 20% and 10% respectively. According to the result we can conclude that the system will have a good potential and it is worthy of our efforts.

4.2 Module Design

It is likely to be friendlier that users can also see what the Kinect is tracking when they interact with computers. It is one of the reasons that XNA Game framework is applied to the project. XNA framework supports combining all the functions and displays the video data on the screen to let developers and users know whether their orders are operated correctly and timely. And in
another hand, game is a market with huge potentials. So XNA becomes an excellent platform to have the chance extending this interaction system to the game market. Besides what has been mentioned above, compared with DirectX, which is another mature game framework, XNA is perfectly compatible with .Net platform and Visual Studio IDE which provide great convenience and efficiency for C# developers. In sum, XNA could be a very good choice as the developing framework.

The image 5.1 shows that the system is composed of 5 modules, Kinect processing, user identifying, DTW recognition, XNA processing and display on screen. Firstly, the Kinect gets and processes the chosen user’s data and passes it to XNA Game. Then in XNA processing part, it integrates the data with DTW algorithm to realize the gestural recognition and execute the corresponding computer events. At last, the video data is displayed and the gestures and computer events are drawn as texts on the screen simultaneously.

The detailed specifications of each module are following,  

(a) Kinect Processing

This module implements the initiation and configuration of the Kinect device. It also catches RGB, depth and skeleton data and accomplishes
mapping the different coordinate systems of them. Besides, it realizes some core functions of tracking, recognizing, calculation of users’ skeleton points, etc.

(b) User Identifying

With the support of *Kinect for Windows SDK*, Kinect device can support six users but could only track at most two users at the same time. So far the interaction system is proposed to be one-controller mode, thus it is necessary for the module to decide which one of the two tracked users is the only authorized active controller.

(c) DTW Recognition

This module has been specified in the last chapter. In the module, it discards the graphical interface and merges the core functions into the interaction system to make XNA able to recognize users’ gestures.

(d) XNA Processing

In XNA processing module, the system should determine whether Kinect is running normally or not firstly. If yes, the contents, such as fonts, images, etc. are loaded into the system. In addition, the system provides the function for configuration. Another core key of this
module is that it combines the previous three modules together and processes and executes them simultaneously. Moreover, once a gesture is successfully recognized, the system will execute the corresponding event automatically.

(e) Display on Screen

![Diagram of Display on Screen](Image)

In this module, if the Kinect device doesn’t run normally, it will show the error messages to users. Otherwise, it displays the real-time video captured by Kinect and the mouse image on the screen. And once gestures are recognized and the corresponding computer events are triggered successfully, it also draws the texts of the gestures on the screen to let users know whether their commands are operated properly.

4.3 Chapter Summary

In this chapter, it states the requirements and the market potential of this interaction system. Then it specifies the main framework and each module of the interaction system.
V. Realization and Test of the System

5.1 Hardware and Software Environment

(a) Operation system: Windows 7 x64.
(b) Development environment: Visual Studio 2012, XNA Game Studio 4.0.
(c) Development kit: Kinect for Windows SDK s1.7, Microsoft XNA Studio 4.0.
(d) Development language: C#, XML.
(e) Hardware: Intel Core i5, 4G RAM.

5.2 Establishment

This interaction project is based on Kinect for Windows SDK 1.7 and XNA Framework 4.0, so Kinect for Windows SDK 1.7, Kinect for Windows Developer Toolkit 1.7 and XNA Game Studio 4.0 should be installed before we start developing.

In Visual Studio, we create a Windows Game (4.0) project and import Microsoft.Kinect.dll, Microsoft.Kinect.Toolkit.dll, Microsoft.Toolkit.Interaction.dll into reference at first.

As to the process of the development, I will emphatically specify two parts, Kinect processing part related to Kinect device and XNA processing part related to XNA framework.

5.3 Kinect Data Processing

5.3.1 Kinect Initialization

Before Kinect starts to track, developers need to set and initial the configuration of Kinect device. In this stage, we need to decide what functions we want to switch on and assign the format of RGB and depth video.

```csharp
colFormat = ColorImageFormat.RgbResolution640x480Fps30;
depFormat = DepthImageFormat.Resolution640x480Fps30;
```

This sets the working modes of RGB and depth cameras which both capture...
the data stream with the resolution of 640*480px. For one computer can connect with 4 Kinect devices simultaneously, the system needs to use \texttt{KinectSensor.KinectSensors[i]} to assign which one is selected one.

Developers need to make a decision what data the Kinect device tracks at first. In the interaction system, it demands all RGB, depth and skeleton data. I also set tracking range as near mode (\texttt{EnableTrackingInNearRange=true}) and tracking mode as seated mode (\texttt{TrackingMode=SkeletonTrackingMode.Seated}) which means that we can use the Kinect device when we are close or seated in a better performance.

\begin{verbatim}
this.kinectSensor.ColorStream.Enable(colFormat);
this.kinectSensor.DepthStream.Enable(depFormat);
this.kinectSensor.SkeletonStream.Enable(smoothParams);
\end{verbatim}

The variable, \texttt{smoothParams}, represents the parameter of smoothing algorithm. If a \texttt{TransformSmoothParameters} parameter is passed in, Kinect will utilize Double Exponential Smoothing algorithm to smooth users’ movement. There are 5 parameters composing the \texttt{TransformSmoothParameters}.

- **Correction**: It represents the correcting value. It can be a float number within the range from 0 to 1. If the value is larger, the correctness of predicting data is more obvious. It is equivalent to the \((1 − β)\) in Double Exponential Smoothing.
- **JitterRadius**: Its unit is meter. It can adjust skeleton point’s jitter radius within a threshold if it is too large.
- **MaxDeviationRadius**: Unless skeleton point’s displacement exceeds this value, it will be regarded as a jitter stead of a normal movement.
- **Prediction**: It describes how many frames are predicted for skeleton moving. It is equivalent to the \(m\) in Double Exponential Smoothing.
- **Smoothing**: It can be a float number within the range from 0 to 1. If the value is larger, the smoothing effect is more obvious. It is equivalent to the \((1 − α)\) in Double Exponential Smoothing.

5.3.2 Kinect Data Stream Processing

RGB, depth and skeleton data have to be stored in arrays in corresponding formats, \texttt{Byte[]}, \texttt{DepthImagePixel[]}, \texttt{Skeleton[]}, respectively.

In RGB data, each pixel is composed of four integral values, R, G, B, Alpha.
In RGB color system, the value of each of the four integers ranges from 0 to 255 which explains the reason we use byte format to store RBG data. In Kinect color data stream, the order of the four values of a pixel is BGRA while it is RGBA in Texture2D in XNA. So an adjustment is required.

```csharp
for (int i = 0; i < eachPixel.Length; i += 4) {
    eachPixel[i] = colorArray[i + 2];
    eachPixel[i + 1] = colorArray[i + 1];
    eachPixel[i + 2] = colorArray[i];
    eachPixel[i + 3] = (Byte)255;
}
```

When processing depth and skeleton data, timestamps should be added into data in every frame to avoid disorder. And when processing skeleton data, acceleration data will be measured and used in Double Exponential Smoothing algorithm to optimize smoothing and prediction.

```csharp
Microsoft.Kinect.Vector4 accelerometer = this.kinectSensor.AccelerometerGetCurrentReading();
interactionStream.ProcessSkeleton(skeletonArray, accelerometer, skeletonFrame.Timestamp);
```

Due to the working principles of Kinect device, RGB, depth and skeleton data come from different cameras which means that their coordinate systems are also different. If we these data in the calculations directly, we will never get correct results. Extra mapping schema is essential for these data. Because we only requires displaying users’ RGB video in the project, it just needs to maps RGB data with skeleton data.

```csharp
ColorImagePoint colorPoint =
    this.kinectSensor.CoordinateMapper.MapSkeletonPointToColorPoint(
        skeletonPoint, colFormat);
return new Point((int)colorPoint.X, (int)colorPoint.Y);
```
5.3.3 User Tracking and Gestural Commands

Kinect can detect six users and track two active users but generally there is only one user controlling the computer, so we should make Kinect choose the only one controller user. Taking the situation into account that there are two active users, we can demand the system to make the first user be tracked, or the left or the right one of the two active users be the only controller. Besides, if we know the unique ID of each user, we can assign someone to be the only one controller by the unique ID.

In this project, I assume that the user is right handed and assign the right hand as the mouse. The system should calculate the position of the user’s right hand in real time. In addition, a parameter, scale, whose range is from 0.01 to 0.1, is introduced to help users adjust the speed of hand moving. The smaller the scale value is, the faster the mouse moves, and vice versa. After confirming that the user is the selected one and the current tracked skeleton point is exactly the one we want as the mouse, the system begins calculating the coordinates of the skeleton point on the computer screen according to the screen resolution. Because the mouse movement is on a 2D flat, only X-axis and Y-axis of the coordinate system are necessary.

```csharp
//default: the first one
return skeletonList.FirstOrDefault<Skeleton>();
//the left one
return (from Skeleton ske in skeletonList orderby ske.Position.X ascending select ske).FirstOrDefault();
//the right one
return (from Skeleton ske in skeletonList where ske != GetLeftUser() orderby ske.Position.X descending select ske).FirstOrDefault();
//choose user by ID
return (from Skeleton ske in skeletonList where ske.TrackingId == id select ske).FirstOrDefault();
```

```csharp
cursorPosition.X = (joint.Position.X + scale) * (screenWidth / (2 * scale));
cursorPosition.Y = (scale - joint.Position.Y) * (screenHeight / (2 * scale));
```

To increase the user experience, the system can support adjusting the angle of the base of Kinect device. It allows users to use UP or DOWN keys to uplift or drop cameras which gives users more comfortable and convenient
XNA is a game developing environment provided by Microsoft which can support running on several platforms. It is based on .Net framework and it has a much large class library. XNA can support games on Windows, Windows Phone, Xbox platforms, so the advent of XNA offers developers a much more powerful tool to make multi-platform games.

The developing process in XNA is divided by these several functions.

- **Initialize()**

  It is used to initialize the objects and tools related to the game, such as graphics device. In this function, I define the attributes of the window which is used to display the RGB video data, such as size, position, etc.:  
  ```csharp
  if (keyBoard.IsKeyDown(Keys.Up))
  {
    camAngle += 3;
    if (camAngle >= 39)
      camAngle = 39;
    this.kinectSensor.ElevationAngle = camAngle;
  }
  if (keyBoard.IsKeyDown(Keys.Down))
  {
    camAngle -= 3;
    if (camAngle <= -39)
      camAngle = -39;
    this.kinectSensor.ElevationAngle = camAngle;
  }
  ```

5.4 XNA Processing

5.4.1 XNA Compendium

XNA is a game developing environment provided by Microsoft which can support running on several platforms. It is based on .Net framework and it has a much large class library. XNA can support games on Windows, Windows Phone, Xbox platforms, so the advent of XNA offers developers a much more powerful tool to make multi-platform games.

The developing process in XNA is divided by these several functions.

- **Initialize()**

  It is used to initialize the objects and tools related to the game, such as graphics device. In this function, I define the attributes of the window which is used to display the RGB video data, such as size, position, etc.

  ```csharp
  User32.SetWindowPos((uint)this.Window.Handle, 0, 10, 10, screenWidth + 10, screenHeight + 30, 0);
  ```

However, if we want to use `SetWindowPos()` in XNA, we need to do the `P/Invoke` firstly. In C#, developers can't directly call the methods which are dependent on the low layer of the operation system, so some extra reference is
required. This is called P/Invoke. DllImport is one of the most important methods to refer to native methods. The user32.dll is the one containing most basic functions about user-interface.

```c
[DllImport("user32.dll")]
public static extern void SetWindowPos(uint Hwnd, uint Level, int X, int Y, int W, int H, uint Flags);
```

- **LoadContent()**
  After initialization, all the game contents, such as images, audio, fonts and so on, are loaded into the system by this function to make them accessible for the game.

- **Update() & Draw()**
  One computer game is composed of numerous small cycles. A cycle is composed of a series of functions and methods. These functions and methods are called repeatedly until the game is over. A cycle in XNA game has two main parts generally, **update()** and **draw()**. In this project, **update()** executes all the logical business related to Kinect and DTW. And **draw()** is only responsible for drawing data on the window.

- **UnloadContent()**
  It is only called when the game is over. When it is called, it unloads all the contents and does some extra operations which are required in the end.

![Graph](image.png)

*Img.6.2. These 5 methods compose a cycle of a XNA Game*

5.4.2 **XNA Content**

In the **Content** folder, it stores the files which are required in the game, such as images, audio, fonts, textures, etc. The reason for using the content is to make game run more efficiently and quickly. Without this **Content**, the program has to read files in raw formats at first and when these files need to be loaded, the program must judge whether their formats are correct and transfer
them to the readable ones. This will definitely lower the efficiency of the program. But the content mechanism eliminates the defect. When game is in build time, files are loaded in advance and the program transforms them to a series of managed codes and serialize them in an executable file. When game is in run time, the program can read the serialized content data directly which can greatly improve loading efficiency.

In this project, it uses two types of files, image and font. After adding images into the content and setting the values of Content Importer and Content Processor attributes as Texture – XNA Framework, we can call images via their Asset Name attributes.

As to font, there are two ways to support it, SpriteFont and SpriteFontTexture.

SpriteFont is actually a XML configuration file. It sets the style, size, etc. of font in XML language. The number of characters SpriteFont can support depends on the local library of the computer.

```xml
<?xml version="1.0" encoding="utf-8"?>
  <Asset Type="Graphics:FontDescription">
    <FontName>Microsoft YaHei</FontName>
    <Size>18</Size>
    <Spacing>0</Spacing>
    <UseKerning>true</UseKerning>
    <Style>Regular</Style>
    <CharacterRegions>
      <CharacterRegion>
        <Start>&#32;</Start>
        <End>&#126;</End>
      </CharacterRegion>
    </CharacterRegions>
  </Asset>
</XnaContent>
```

In the project, I choose the other way, SpriteFontTexture, to make the font, which aims to turn characters into a bmp image. It doesn't need the support of the local font library. So every character can be displayed correctly even though it doesn't exist in the computer. I use an open-source tool, ttf2bmp, transforming the TrueType font to a bmp image. After adding it into the content, we need to change the value of Content ProcessorSprite to Font
Texture – XNA Framework. But the defect of this method is that it can’t support some other characters well, such as Chinese.

After initialization, I add a txt file to record users’ habitual moving scale and Kinect camera angle in LoadContent() function. If this record file doesn’t exist, the system will create a new one and write current data into it. And the record file will be saved in UnloadContent().

- **Update()**

  The logical codes of the system is mostly executed in Update() part. In this part, the interval of every game cycle is, $(\text{float})\text{gameTime.ElapsedGameTime.TotalMilliseconds}$. I add a title on the window, scrolling the system instructions with the interval being 500 millisecond.

  ```csharp
  scrollTime = scrollTime - updateTime;
  if (scrollTime < 0) {
    scrollTime = 500;
    message = message.Substring(1, message.Length - 1) + message.Substring(0, 1);
  }
  ```

  In the system, I also add some shortcuts to make user experience better. The left key and right key mean increasing and decreasing moving scale, the up key and down key mean lifting up and down the camera, the esc key means exiting and the space key means pausing the program. As to up key and down key, one second interval is inserted between every two clicks in order to protect camera from moving too frequently.

  After confirming the TrackingId is correct, the system should calculate the
coordinates of user’s right hand immediately and draw the mouse image on the position of the right hand. When these steps are done, remember to set the actual mouse on that position. When user moves his hand, the mouse image should move following the hand. However, the coordinate system of user’s hand is from Kinect and the coordinate system of mouse image is from computer screen, so it is indispensable to do a mapping between these two different coordinate system to avoid dislocation.

Mouse.SetPosition((int)cursorPosition.X, (int)cursorPosition.Y);

...... ......
skeletonPoint = joint.Position;
newDynamicPoint = PointSkeletonToColor(skeletonPoint);
//PointSkeletonToColor: coordinate system transformation of different Kinect cameras
float frameWidth = (float)this.kinectSensor.ColorStream.FrameWidth;
float frameHeight = (float)this.kinectSensor.ColorStream.FrameHeight;
cursorDynamicPosition.X = newDynamicPoint.X * (devWidth / frameWidth);
cursorDynamicPosition.Y = newDynamicPoint.Y * (devHeight / frameHeight);

As to the associations of gestures and computer events, the system checks the results from the Kinect data processing module and DTW recognition module. If a result is matched with the rules we have defined, it makes computer response to it.

The rules are following,
Press Right Hand -> right click
Grip Right Hand -> hold left click
Release Right Hand -> release left click
Press Left Hand -> double click
Grip Left Hand -> hold wheel
Release Left Hand -> release wheel
Left Hand Push Up -> Alt + F4
(The rules are used to test the performance of the system, I don’t apply all gestures that I have trained to the computer events.)

In XNA game, if I want to call the mouse and keyboard events, I need to declare them at first. The method is also the P/Invoke[^2].

```csharp
[DllImport("user32.dll")]
public static extern void mouse_event(uint GetSkeletonByIdFlags, uint dx, uint dy,
    uint dwData, int dwExtraInfo);
[DllImport("user32.dll")]
public static extern void keybd_event(byte bvk, byte bScan, uint dwFlags,
    UIntPtr dwExtraInfo);
```

[^2]: "user32.dll"
In this project, Draw() function is only used to the draw graphic for the system. In XNA, the most important and most common tool for drawing is SpriteBatch.

In the interaction system, the draw function has two parts. One is that when Kinect device is not connected or the SDK is not installed, it should have reminders displayed on the screen.

The other is that when the interaction system runs normally, it draws the RGB video on the game window and draws the mouse image on the position

```csharp
// Enumeration of mouse events
[Flags]
public enum MouseEventFlags { 
    LEFTDOWN = 0x00000002, 
    LEFTUP = 0x00000004, 
    MIDDLEDOWN = 0x00000020, 
    MIDDLEUP = 0x00000040, 
    MOVE = 0x00000001, 
    ABSOLUTE = 0x00000000, 
    RIGHTDOWN = 0x00000008, 
    RIGHTUP = 0x00000010, 
    WHEEL = 0x00000800, 
    XDOWN = 0x00000080, 
    XUP = 0x00000100
}

// Enumeration of keyboard events
void PressKey(byte keyCode)
{
    const int KEYEVENTF_EXTENDEDKEY = 0x1;
    const int KEYEVENTF_KEYUP = 0x2;
    // click keyboard button
    User32.keybd_event(keyCode, (byte)0x45, KEYEVENTF_EXTENDEDKEY, (UIntPtr)0);
    // release keyboard button
    User32.keybd_event(keyCode, (byte)0x45, KEYEVENTF_EXTENDEDKEY | KEYEVENTF_KEYUP, (UIntPtr)0);
}
```

- **Draw()**

  In this project, Draw() function is only used to the draw graphic for the system. In XNA, the most important and most common tool for drawing is SpriteBatch.

  In the interaction system, the draw function has two parts. One is that when Kinect device is not connected or the SDK is not installed, it should have reminders displayed on the screen.

```
spriteBatch.Draw(errorBackgroundImage, new Rectangle(0,0,setWidth,screenHeight), Color.White);
spriteBatch.DrawString(mainFont, errorMsg1, new Vector2(200, 50), Color.Red);
```
of the right hand, namely the actual mouse. When the mouse image shows, it means that the Kinect device has chosen the right user and tracked him successfully. When a gesture is recognized, it should draw the message of the gesture on the window to state that this order has been taken and demand the computer response to it. For example, when the user grabs his right hand, there will be a message “LEFT CLICK DOWN” on the window and the computer do clicking the left button of the mouse in the meantime. All of these activities should be ensured in real time.

```csharp
public void DrawFore(ref SpriteBatch spriteBatch3) {
    if (display)
        spriteBatch3.Draw(texture, rectangle, null, color, 0, center,
            SpriteEffects.None, 0); // 0 is foreground, 1 is background

    spriteBatch.Draw(kinSensor.kinectVideo, new Rectangle(0, 0, screenWidth,
        screenHeight), Color.Wheat);
    cursorImage.DrawFore(ref spriteBatch);

    ... ...
    if (gestureText.Contains("PRESS RIGHT")) {
        spriteBatch.DrawString(textFont, "RIGHT CLICK", new Vector2(230, 10),
            Color.Salmon, 0, new Vector2(0, 0), 1f, SpriteEffects.None, 0);
    }
    ... ...
}
```

Moreover, the system displays the moving scale value on the window all the time to let user know what scale they like most. This can make the system more convenient for them to adjust the scale value to fit their habits later.

```csharp
spriteBatch.DrawString(textFont, "Move Scale: " + moveScale.ToString("0.00"), new Vector2(10, 450), Color.AliceBlue, 0, new Vector2(0, 0), 0.5f, SpriteEffects.None, 0);
```

In this project, it supposes that the computer is controlled by one user; but with the development of technologies, it is not impossible to happen that more than one user can control one computer together. With the consideration of this, I add the collision model of XNA game into the system. As a result, when several users control a computer in the meantime, it won’t happen that users’ hands appear on the same position to avoid some unexpected operations or errors. And this function can also provide better scalability and performance when some games run on the system.

```csharp
boundSphere = new BoundingSphere(new Vector3(position.X + rectangle.Width / 2,
    position.Y + rectangle.Height / 2, 0), rectangle.Width / 2);
```
5.5 Testing

5.5.1 Testing Requirements

In the interaction system, I define these rules for the associations of gestures and computer events.

- Press Right Hand -> Right Click
- Grip Right Hand -> Hold Left Click
- Release Right Hand -> Release Left Click
- Press Left Hand -> Double Click
- Grip Left Hand -> Hold Wheel
- Release Left Hand -> Release Wheel
- Left Hand Push Up -> Alt + F4

(a) Press Right Hand: when the user does the press gesture with his right hand, the right button of the mouse will be clicked.

(b) Grip and Release Right Hand: when the user does the grip gesture with his right hand, the left button of the mouse will be hold until the hand releases

(c) Press Left Hand: when the user does the press gesture with his left hand, the left button of the mouse will be double-clicked.

(d) Grip and Release Left Hand: when the user does the grip gesture with his left hand, the wheel of the mouse will be hold until the hand releases.

(e) Left Hand Push Up: when the user push his left hand (arm) up, the Alt key and the F4 key will be clicked together to close the current program.

5.5.2 System Demos

When the Kinect device is not connected normally with the computer or the Kinect SDK 1.7 hasn’t been installed, the system will remind the user of these.
When the Kinect device and the system run normally, every rule is tested and the demos are following,

(a) Press Right Hand -> Right Click

Img.6.6. demo of right click
(b) Grip Right Hand -> Hold Left Click & Release Right Hand -> Release Left Click

![Image 6.7. demo of holding left click and dragging](image)

(c) Press Left Hand -> Double Click

![Image 6.8. demo of double click](image)

(d) Grip Left Hand -> Hold Wheel & Release Left Hand -> Release Wheel

![Image 6.9. demo of holding wheel and scrolling](image)
(e) Left Hand Push Up -> Alt + F4

![Image of a left hand pushing up](image)

**Img.6.10. demo of closing the current window**

In this testing, I have done numerous tests on these pre-defined gestures and these activities can mostly be recognized and executed successfully. I do 50 times tests on each rule and statistically the recognition rate of each gestures can be more than 85% or even 90% which in my opinion has reached the requirements of the beta version.

5.6 Chapter Summary

In this chapter, it states the realization part of the interaction system in details. It introduces the developing environment, the developing process and the details of logical codes. At last, the test is carried out to evaluate the effects of the system.
VI. Conclusion and Prospect

6.1 Conclusion

Human-computer interaction is always in the state of fast development and it always aims to give human better experience, more convenient operation and better customized service. It is exactly one of the most important issues in the area that this thesis is researching on. By means of Kinect device tracking users’ key skeleton points and associating them with the computer events, it can help users interact with computers directly via body gestures without any touching or using any other tool. And I also believe that this more human-like interaction way can bring human more natural and marvelous experience.

In this thesis, I refers to many researches and achievements about Kinect, studies the latest technologies brought by Kinect SDK and integrates them with the gestures recognition algorithm which can help promise the feasibility and effectiveness of the project.

In this project, these tasks have been accomplished in general,

(a) It deeply studies the working theory, developing process and methods of Kinect and the API of Microsoft Kinect SDK. Then it processes the Kinect data scream, catches the certain gestures, such as grip, release and press and defines them as mouse and keyboard events which can basically cover the most common activities.

(b) It designs the training and recognizing program based on DTW algorithm in order to let users add and modify customized gestures interacting with computers in the simplest ways. Users can decide what gestures they want and how the computer responses to the gestures by themselves. It also posts an experiment on the program and analyzes the results.

(c) It integrates the static gestures that the SDK provides and the dynamic gestures that the DTW algorithm provides into the XNA framework. In XNA, it associates the gestures with computer events and releases the
interaction system as a XNA game with a window displaying the current state in real time.

6.2 Prospects and Improvements

Although the interaction system has achieved the expected requirements, it can’t be denied that there are lots of flaws.

Kinect device not only supports skeleton tracking, but also allows to recognize audios. So I think it can be a good idea to add the function of audio recognition to help disabled people use this system.

As to mouse moving, its preciseness and agility are far from the current traditional mouse, so some algorithms are eagerly required to improve the performance. This is definitely the most important direction we need to move forward right now.

As to the DTW algorithm used in gesture recognition, it reaches a relatively ideal expectation, but DTW has a drawback that it can only support simple gestures well. When gestures are getting more complicated, its performance is becoming worse than HMM. So in the later improvements, HMM can be considered adding into the system to allow users to interact with computer by some complicated gestures.

Besides dynamic gestures, the static gestures of the system are still limited within the ones provided by Kinect SDK. Therefore, the studies and researches on the algorithms which can support recognizing customized static gestures should be carried out, such as SVM, K-Means and Bayesian Network.

6.3 Chapter Summary

In this chapter, it summarizes the reasons why I want to develop this interaction system and what has been done and achieved in this project. Apart from those, it also discusses the current flaws of the system and what improvements should be researched and accomplished in the future about this project.
References

[3] Aaron Reed, XNA 4.0 Tutorial, O’REILLY.
Engineering and Informatics, 2011.07.


[33] http://pinvoke.net/default.aspx/user32/keybd_event.html (Date accessed at 2013.11)
[34] http://www.codeforge.cn/article/150074 (Date accessed at 2013.11)
[37] Guangyou Xu, Linmi Tao, Yuanchun Shi, Xiang Zhang, Human-Computer Interaction in Pervasive Computing, Department of Science and Technology of Tingshua University, July 2007.
[40] Chao He, Zhangfang Hu, Yan Wang, A Dynamic Gesture Recognition Method based on Improved DTW, National University of Defense Technology, June 2013.
[41] Heqing Qian, Augmented Reality in Education System based on Kinect and Gesture Recognition, Shanghai Jiaotong University, Dec 2011.