

Writing In Air

A-Level Coursework by Shenggang Hu
d'Overbroeck's College

2014

Contents

1	Definition, Investigation and Analysis	3
1.1	Definition - nature of the problem to be investigated	3
1.1.1	About the Organisation	3
1.1.2	About the End User	3
1.1.3	Kinect Sensor	4
1.1.4	Areas of Development	4
1.1.5	About the Data	5
1.2	Investigation and Analysis	5
1.2.1	System Analysis and Communication	5
1.2.2	Requirement Specification	18
1.2.3	Client's confirmation to the Requirement Specification . .	19
2	Design	20
2.1	Nature of the solution	20
2.1.1	Objectives and Limitations	20
2.1.2	Stroke Categorisation	20
2.1.3	Window design	22
2.1.4	Design of database	23
2.1.5	Client's confirmation to the Design Specification	25
2.2	Algorithms	26
2.2.1	Main Program	26
2.2.2	Result search	27
2.2.3	Data Insertion	28
2.3	Test Strategy	29
2.3.1	Initialisation	29
2.3.2	Distance Calibration	29
2.3.3	Stroke Recognition	29
2.3.4	Gestures	31
2.3.5	Database	31
3	Software Development and Testing	32
3.1	Alpha Testing	32
3.1.1	Kinect Tracking Parameters	32
3.1.2	Scaling Input to the Screen	32

3.1.3	Input Selection	33
3.1.4	Stroke Pattern Recognition	36
3.1.5	Stroke Recognition	39
3.1.6	Wiping Gesture Recognition	41
3.2	Source Code	43
3.2.1	Main Window	43
3.3	Testing	69
3.4	Acceptance Testing	74
4	Evaluation	76
4.1	Meeting the objectives	76
4.1.1	User Requirements	76
4.1.2	Hardware Requirements	76
4.2	Software Requirements	77
4.3	Desirable extensions	79

Chapter 1

Definition, Investigation and Analysis

1.1 Definition - nature of the problem to be investigated

1.1.1 About the Organisation

The company is a 'life-style company more specialising in interesting projects for clients that range right across the software spectrum.

Among them have skills in programming mostly in C, C++, Perl and Java though more recently they've been writing code in Python. In recent years they've also been involved in writing code for dynamic web sites which has meant HTML, CSS, MySQL and PHP. One of us also has many years experience of electronic design ranging from early ICL computer systems to embedded control systems.

They've designed apps for the iPhone and Android, they've built embedded systems for alarm companies, hand held electronic gadgets for a local games company and more recently built complete web sites using a combination of WordPress and the standard web technologies mentioned earlier.

1.1.2 About the End User

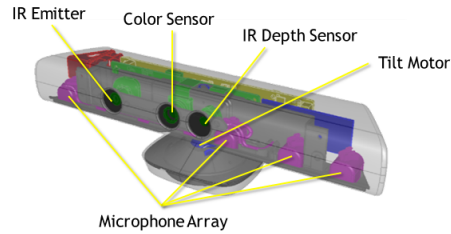
The end user will be members of the organisation. They are technically sophisticated and able to correctly setup the settings for the program.

The user will need a Kinect gaming unit in order for the application to work. The application allows user to input Chinese Character without knowing its pronunciation.

1.1.3 Kinect Sensor

A Kinect Sensor will have the following components installed:

- *An RGB camera* that stores three-channel data in a 1280 x 960 resolution at 12 frames per second, or a 640 x 480 resolution at 30 frames per second. This makes capturing a color image or video possible.
- *An infrared (IR) emitter and an IR depth sensor*. The emitter emits infrared light beams and the depth sensor reads the IR beams reflected back to the sensor. The reflected beams are converted into depth information measuring the distance between an object and the sensor. This makes capturing a depth image possible.
- *A multi-array microphone* that contains four microphones for capturing sound. Because there are four microphones, it is possible to record audio from a specific direction, as well as find the location of the sound source and the direction of the audio wave.
- *A three-axis accelerometer* configured for a 2G range, where G is the acceleration due to gravity. It is possible to use the accelerometer to determine the current orientation of the sensor.



In this program, only the IR emitter and the IR depth sensor are used.

The **Kinect for Windows SDK** is required for using a Kinect on PCs, note that the driver is included in the package.

1.1.4 Areas of Development

Two major parts of the program are:

- Kinect-User Interface
- Database

The Kinect-User Interface will allow interaction between the user and the Kinect. The gestures of the user will be recognised and response will be given according to the gesture.

The database will store all the Chinese characters and their stroke representations. Input from the user will be sent to the database and results will be given.

1.1.5 About the Data

The computer is able to obtain three forms of video streams from Kinect. Color stream, depth stream and skeleton stream. In this program, only the depth and skeleton data are used. The depth frame gives the position of the joints in the space and the skeleton frame tracks all the skeleton in front of the Kinect.

1.2 Investigation and Analysis

1.2.1 System Analysis and Communication

Email 1:

From: John Peter Thomas [john.peter.thomas@gmail.com]

Sent: 11 November 2013 08:57

To: Sheng Hu

Subject: Computing Project

Hi Jack

I've had an email from your teacher who happens to be a good friend of mine. Most years we work with his better students to do interesting and often quite complex projects, some of which either form the basis of projects for our clients whilst others are used to demonstrate interesting proof of concepts. Alan has suggested that you would be capable and interested in taking on a project for the Kinect gaming unit. He has in mind a system whereby you identify strokes which are then combined into chinese characters so as to allow you to 'write in the air'. This would be a difficult and very challenging project but Alan tells me that you are a quite exceptional student and so would be up to it.

To do so you'd need a Kinect unit, a Windows based PC or laptop and a C# compiler. I understand from Alan that you have all of the above so perhaps you could pop over to see us sometime soon for an initial discussion on the project.

Alan will give you instructions as to how to get to us. We're probably going to work with one of your classmates too so perhaps the two of you could come to us together. It would give you a chance to see what we do and hopefully get underway.

Cheers

-John

Email 2:

From: Sheng Hu [HU004@doverbroecks.com]

Sent: 11 November 2013 09:11

To: John Peter Thomas

Subject: RE: Computing Project

Hi John,

So far I have got all the hardware and software you listed. We should probably decide when to meet.

Best Regards,
Jack

Email 3:

From: John Peter Thomas [john.peter.thomas@gmail.com]

Sent: 11 November 2013 14:42

To: Sheng Hu

Subject: Re: Computing Project

Hi Jack

I'm pretty tied up for the next week or so - how about meeting up on tuesday 26th November ? It'll actually be easier if I pop across to see you. I need to chat to Alan about stuff so perhaps 3 o'clock will be sensible. Let me know if that suits you. In addition could you please have a chat to your fellow student Callum who is also looking to work with us on a project. It would be sensible if I could see both of you in the same afternoon.

Best wishes
-John

Email 4:

From: Sheng Hu [HU004@doverbroecks.com]

Sent: 12 November 2013 23:13

To: John Peter Thomas

Subject: RE: Computing Project

Hi John

Sorry to email you at this point. Callum and I have both agreed to meet at that time. Do I need to bring my kits to the meeting?

Best Regards,

Jack.

After having a meeting with my client, we agreed to produce several prototypes for the functions below:

1. A program that able to track the hands of the user and draw points according to the position.
2. A program that use a mouse to draw a character and recognise it. The use should be able to decide when to start and stop the stroke.
3. A model for the database and its traversal method. The real data can be found added some time later.

Email 5:

From: Sheng Hu [HU004@doverbroecks.com]

Sent: 12 December 2013 11:42

To: John Peter Thomas

Subject: Computing Project

Hi John,

It has been about two weeks after the meeting. I have done a program that can record the user's hand motion by Kinect. I've been looking at using the recognition engine for Windows Tablet PC to recognise chinese characters so that I don't even need to make my own database. However, my laptop seems to be lacking this engine, so the second prototype cannot be finished. Therefore, I will need to recognise each stroke and the database must store most of the chinese characters with their stroke representation. Due to the lack of data, I cannot decide which model to use. I'd thought that storing characters in a tree will be most sensible. A tree that start from the root and linking to further nodes. Each node will have several children and the characters the node store. Anyway, I would like to start to program the real application and the database will be much easier to make after having written the recognition function. So shall we arrange another meeting?

Regards,

Jack.

Email 6:

From: John Peter Thomas [john.peter.thomas@gmail.com]

Sent: 15 December 2013 16:49

To: Sheng Hu

Subject: Re: Computing Project

Another meeting seems like a sensible option given the amount of work that you seem to have done. I'm away for the next few days - how about next wednesday at say 4 o'clock at our offices?

cheers

-John

Email 7:

From: Sheng Hu [HU004@doverbroecks.com]

Sent: 2 January 2014 12:48

To: John Peter Thomas

Subject: Recognition Method

Hi John,

I have started writing the program since the previous meeting. What it can do so far is recording the trace of the users hand when it sees a draw gesture. I looked up some paper that describes how to recognise handwritten characters because I need to write a recognition method for the program. However, their approach is quite complicated. So I wonder if you can give me any suggestion and hope we can have another meeting on the recognition engine.

Regards

Jack

Email 8:

From: John Peter Thomas [john.peter.thomas@gmail.com]

Sent: 6 January 2014 08:49

To: Sheng Hu

Subject: Re: Recognition Method

I've seen a couple of papers on this area, both by researchers in China - its a tricky problem I know but if you can find a sensible way of unambiguously identifying the start and end of a stroke you should be well on your way. You're going to need to construct an appropriate data structure, possibly a binary tree or a linked list to hold the lists of strokes that identify characters - efficiency should be top of your list of criteria for choosing this. I'm available this week for a meeting. Bring along the work that you've done already and we can talk through the issues at length. Call me to arrange a time.

cheers

-John

In this meeting, we decided that the recognition will be done by recognising strokes rather than any other method due to the following reasons:

1. The recognition will not be done by pinyin(by the pronunciation), a common way to input chinese characters, because this has nothing to do with the topic "writing in air".
2. Character recognition will not be a suitable method as it may take a while to finish a single character and the user may expect the program to predict his input.

Email 9:

From: Sheng Hu [HU004@doverbroecks.com]

Sent: 9 January 2014 9:34

To: John Peter Thomas

Subject: Database

Hi John,

Having decided the recognition method, it didn't take long to find a database that fits your model. Here is a section of the database:

These columns represent (from left to right) the chinese character, the index of the character in the database, the unicode for that character and the stroke combination. It divides strokes into only 5 categories which will significantly reduce the complexity of the data tree. I've experienced some difficulty with loading chinese characters into C, therefore I think I only need the third and the fourth column of the database. Using that data, I can create a tree of nodes that each has 5 branches. Shall I use this model?

Regards.

Jack.

Email 10:

From: John Peter Thomas [john.peter.thomas@gmail.com]

Sent: 13 January 2014 09:23

To: Sheng Hu

Subject: Database

Why five branches per node ?

Email 11:

From: Sheng Hu [HU004@doverbroecks.com]

Sent: 13 January 2014 9:39

To: John Peter Thomas

Subject: Database

Because there are five types of strokes. I would give each branch a number (1 to 5) to represent a type of strokes and each node will be represented by a string of numbers representing the branches took to travel from the root to this node. In this way, node 1312 will have five children 13121, 13122, 13123, 13124 and 13125.

Regards,
Jack.

Email 12:

From: Sheng Hu [HU004@doverbroecks.com]

Sent: 14 January 2014 11:44

To: John Peter Thomas

Subject: Requirement Specification

Hi John

With the help of the meetings and emails, I have made a summary on what we have agreed on.

1. The program will be gesture driven. Apart from initialisation, every other events will be triggered by a gesture.
2. Recognition will be done by recognising each stroke drawn by the user.
3. Strokes will be categorised into 5 different types.
4. When the program starts, a tree like database will be constructed based on a text file of data.
5. If the program has a string of strokes, the database will be able to return an array of characters in unicode.

Anything to add to it?

Regards,

Jack.

Email 13:

From: John Peter Thomas [john.peter.thomas@gmail.com]

Sent: 18 January 2014 08:56

To: Sheng Hu

Subject: RE:Requirement Specification

1. The program will be gesture driven. Apart from initialisation, every other events will be triggered by a gesture.

OK - I'd like to see a list of the gestures that you're planning on using.

2. Recognition will be done by recognising each stroke drawn by the user.

OK.

3. Strokes will be categorised into 5 different types.

OK.

4. When the program starts, a tree like database will be constructed based on a text file of data.

OK. Just as a matter of interest, where are you getting the text file from?

5. If the program has a string of strokes, the database will be able to return an array of characters in unicode.

OK. I presume that your previous email showed a section of the relevant information.

Anything to add to it?

I've got a meeting with some colleagues in a few days to talk about this project. They're going to quiz me on exactly how it will work. It'll help if you could please explain in detail exactly how the user will write the message 'Welcome to our company' in the air. Please include each and every gesture, stroke etc. and what will be shown on the screen throughout. The more diagrams the better, software engineers like pictures :-)

Thanks

-John

Email 14:

From: Sheng Hu [HU004@doverbroecks.com]

Sent: 20 January 2014 10:32

To: John Peter Thomas

Subject: RE:Requirement Specification

Hi John,

As far as I am concerned, the program will not be able to recognise any English letters nor numbers. Therefore, I can only tell you how to write "欢迎光临本公司" which is "Welcome to our company" in chinese. This is explain in attached files.

Regards,

Jack.

Email 15:

From: John Peter Thomas [john.peter.thomas@gmail.com]

Sent: 22 January 2014 12:03

To: Sheng Hu

Subject: RE:Requirement Specification

Hi Jack

I certainly didn't expect you to recognise english characters :-). I'm clear now on exactly what you're planning - thanks for the files - very clear.

cheers

-John

Email 16:

From: Sheng Hu [HU004@doverbroecks.com]

Sent: 25 January 2014 09:44

To: John Peter Thomas

Subject: RE:Requirement Specification

Hi John,

How was the meeting with your colleagues? I've formally written the requirement specification(attached file). Does it meet your need?

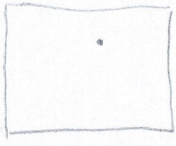
Best Regards,
Jack

Gestures: - user - knee - pointer (not drawing) - pointer (drawing)

① Draw left arm



Screen:

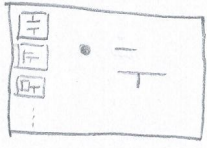
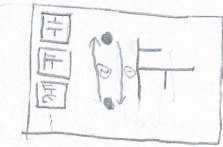


Draw will be started by reaching you left arm out and ended by taking it back.

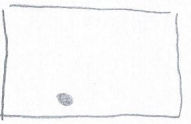
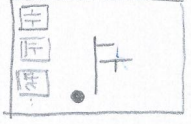
② Clear last stroke.



Screen:



③ Clear screen



both hand hidden.

④ Select

Done by move the pointer to the desired character and perform 'draw' gesture.

To write "欢迎光临本公司"

欢: in strokes: 7 → \ → / → 7 → / → \ → select

on the screen: 7 → 7 → 7' → 7'' → 7''' → 7'''' → select

迎: in strokes: ' → \ → 7 → | → \ → 3 → \ → select

on the screen: ' → ' → ' → ' → ' → ' → ' → ' → ' → ' → select

光: in strokes: | → \ → / → \ → / → \ → / → \ → select

on the screen: | → | → | → | → | → | → | → | → | → | → select

临: in strokes: | → | → / → \ → \ → \ → \ → \ → \ → \ → select

on the screen: | → | → |' → |'' → |''' → |'''' → |''''' → |'''''' → select

本: in strokes: \ → | → / → \ → \ → \ → \ → \ → select

on the screen: \ → \ → \ → \ → \ → \ → \ → \ → \ → \ → select

公: in strokes: | → \ → \ → \ → \ → \ → \ → \ → select

on the screen: | → | → |' → |'' → |''' → |'''' → |''''' → select

司: in strokes: 7 → \ → 7 → | → 7 → | → 7 → | → select

on the screen: 7 → 7 → 7' → 7'' → 7''' → 7'''' → 7''''' → select

The loop of writing will be

```

start draw → draw 'stroke' on the screen
      ↑
      move the pointer to the start of the next stroke
      ← end draw
  
```

Email 17:

From: John Peter Thomas [john.peter.thomas@gmail.com]
Sent: 29 January 2014 11:57
To: Sheng Hu
Subject: RE:Requirement Specification

Hi Jack

Thanks for the requirement spec - short but accurate.
Lets meet up next week some time to confirm everything. I'm going to chat to my colleagues in the meantime to see whether there's anything that we think is missing from the specification.
How about Tuesday at say 5 o'clock at our offices.

Cheers
-John

Email 18:

From: Sheng Hu [HU004@doverbroecks.com]
Sent: 29 January 2014 15:05
To: John Peter Thomas
Subject: RE:Requirement Specification

Thanks for your email. I will see you on Tuesday.

Best Regards,
Jack

Email 19:

From: Sheng Hu [HU004@doverbroecks.com]
Sent: 5 February 2014 09:33
To: John Peter Thomas
Subject: RE:Requirement Specification(final)

Hi John,

I've added what we discussed on Tuesday to the Requirement Specification(in blue). It is again attached to the email. Can this be the final version?

Best Regards,
Jack

Email 20:

From: John Peter Thomas [john.peter.thomas@gmail.com]

Sent: 8 February 2014 18:02

To: Sheng Hu

Subject: RE:Requirement Specification(final)

Hi Jack

That sounds pretty much like it - my colleagues and I were very impressed with you on Tuesday - it does indeed sound like a very impressive project. The requirement specification youve attached covers everything that we can think of at the moment. I know that you need to get cracking on this so lets agree that the requirement spec is final. Send me a copy and Ill return a signed copy back to you.

If there are any other ideas that we have later we can add them to a list of stuff to do once this current project has completed.

Best wishes

-John

1.2.2 Requirement Specification

User Requirements

- user should have full control during the writing process

Hardware Requirements

- Kinect Sensor
- 32bit(x86) or 64 bit(x64) Intel core processor
- Dual-core 2.66-GHz or faster processor
- Dedicated USB 2.0 bus
- 2 GB RAM
- Graphics card that supports DirectX 9.0c

Software Requirements

Operating System

Windows 7

Windows 8

Hand Tracking

- able to track the user's hand
- able to correctly recognise user's gestures for drawing and wiping
- able to correctly recognise any strokes drawn by the user
- tracking should be accurate with nearly no delay

Interface

- able to pass data to the database
- able to retrieve data from the database
- able to output the outcomes and allows the user to choose
- able to output the chosen character

Database

- able to receive data from the main program
- able to give a response of with possible outcomes in unicode with no delay

1.2.3 Client's confirmation to the Requirement Specification

By signing this document, both parties agree that the requirement specification is satisfactory and meets the client's vision in regard to the application design and that the final product will be very close to that described.

Developer's Signature

Client's Signature

Date: ____/____/____

Chapter 2

Design

2.1 Nature of the solution

2.1.1 Objectives and Limitations

Objectives:

1. The program should be easy to use
2. The gestures involved should be distinctive
3. The program should recognise the characters by the order of strokes
4. The program should give instant response to the drawn figure
5. The user interface should be within a screen size, with appropriate arrangement of tools

Limitation:

1. The accuracy drops when the motion is fast, because the program looks for hand movement which is relatively harder to track than the whole body movement
2. The Kinect has a effective range of 0.8m to 3.5m, to ensure the accuracy, there need to be a sufficient separation between the sensor and the user.
3. The Kinect tracks "skeletons" that has a shape like human body, it can get confused between a real human and a hanging jacket.

2.1.2 Stroke Categorisation

Writing a character usually obeys a conventional order of writing stroke(even you don't have to). Briefly speaking, people write characters from top to bottom, from left to right and from outer to inner. Therefore, for this project,

recognising hand written character by stroke order would be reasonably accurate. However, if the user write a character in a different order, there will be no way of recognising it correctly.






























	A	B	C	D	E	F
1						
2						
3						
4						
5						

Figure 2.1.1: all strokes in use

From *Figure 2.1.1* we can see that apart from the first 6 strokes, other strokes are the combination of these 6 strokes. There are 29 different strokes in total. The database used in this program categorises them into 5 stacks.

1. Stroke *B1*
2. Stroke *C1*, Stroke *F2*
3. Stroke *D1*, Stroke *F1*
4. Stroke *A1*, Stroke *E1*
5. All other strokes.

2.1.3 Window design

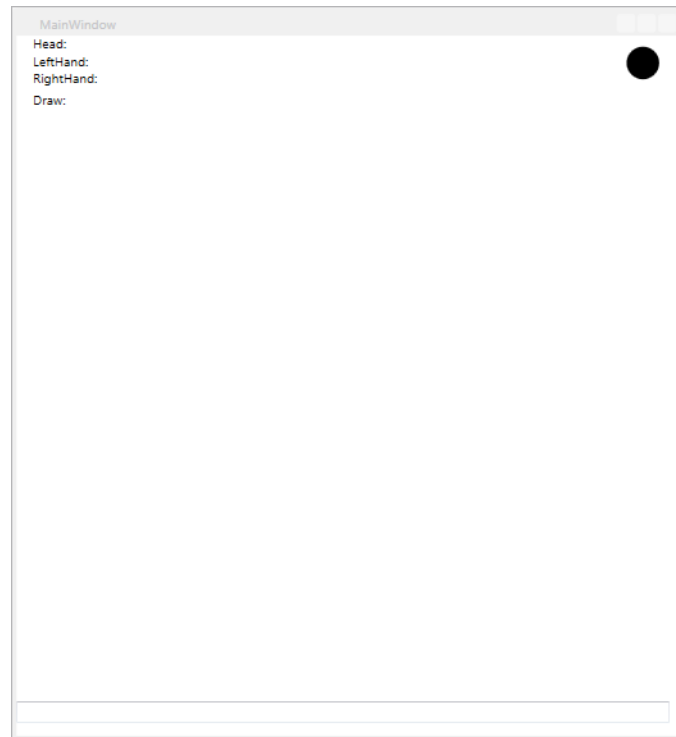


Figure 2.1.2

2.1.4 Design of database

Chinese Character	Index number	Unicode representation	Stroke representation
一	00001	4E00	1
丨	00002	4E28	2
丿	00003	4E85	2
㇏	00004	4E3F	3
丶	00005	4E36	4
㇏	00006	4E40	4
㇏	00007	4E41	4
乙	00008	4E59	5
乚	00009	4E5A	5
㇏	00010	4E5B	5
二	00011	4E8C	11
丁	00012	4E01	12
𠂇	00013	4E05	12
十	00014	5341	12
㇏	00015	4E06	13
厂	00016	5382	13
𠂇	00017	4E02	15

Figure 2.1.3: raw data

The figure shows the first bit of the database. There exists problems in handling with chinese characters, so the database used in this program will store unicode representation instead. Information is extracted and rearranged into the next figure.

Stroke Representation	Unicode Representation of characters with the same Stroke Representation
/	
1 4E00	
2 4E28, 4E85	
3 4E3F	
4 4E36, 4E40, 4E41	
5 4E59, 4E5A, 4E5B	
11 4E8C	
12 4E01, 4E05, 5341	
13 4E06, 5382	
15 4E02, 4E03, 531A, 5338	
21 4E04	
22 5202	
24 535C	
25 5182	
32 4EBB	
34 4E42, 4EBA, 5165, 516B	
35 4E5D. 513F. 51E0. 52F9. 5315	
...	

Figure 2.1.4

In this file, all the characters that have the same stroke order are stored in the same line. This file will be loaded into the program every time it runs. This file will be stored in .txt format within the same folder as all other executable file.

At this stage, it is pretty obvious that tree structure will be a suitable model of the database as the order of strokes is taken in count. Storing characters in nodes that somehow represent the stroke order seems easy to implement.

However, by testing, if a node only stores the character(s) its stroke order represents, it will be hard to design an efficient searching method as users expect the program to make prediction about what character they are drawing. Then, the search will take a long time as one node has 5 branches. Collecting 6 relevant characters is inefficient in that way because most of the nodes it visit will be empty. In addition, if we increase the number of desired output, the performance will be even worse.

Therefore, it is decided that during the insertion, the parent node will store the content of its child nodes. This ensures that the content of each node will store both the characters it represent and the prediction about the character the user is looking for.

Structure of database

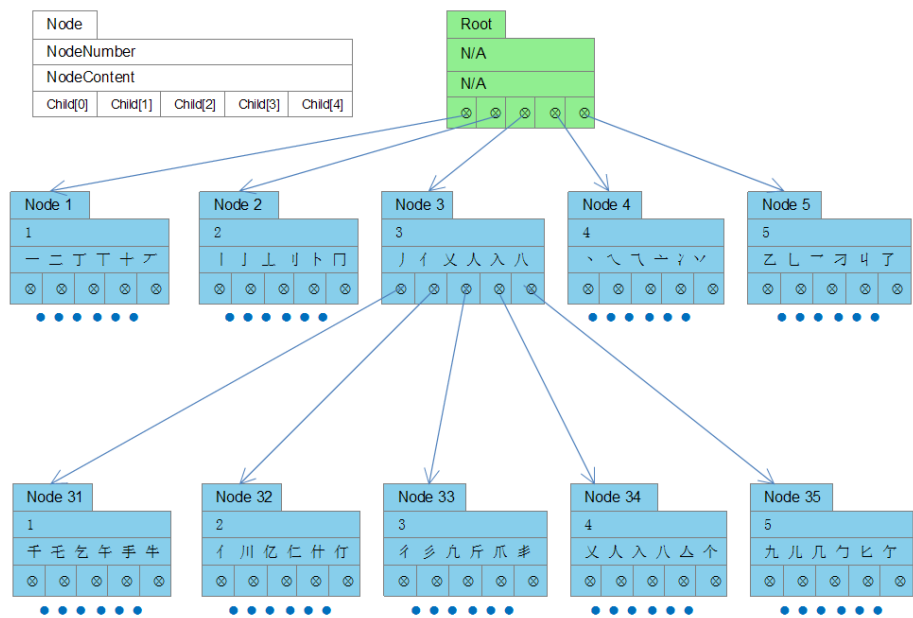


Figure 2.1.5

2.1.5 Client’s confirmation to the Design Specification

By signing this document, both parties agree that the design specification is satisfactory and meets the client’s vision in regards to the application design and that the final product will be very close to that described.

Developer’s Signature

Client’s Signature

Date:____/____/____

2.2 Algorithms

2.2.1 Main Program

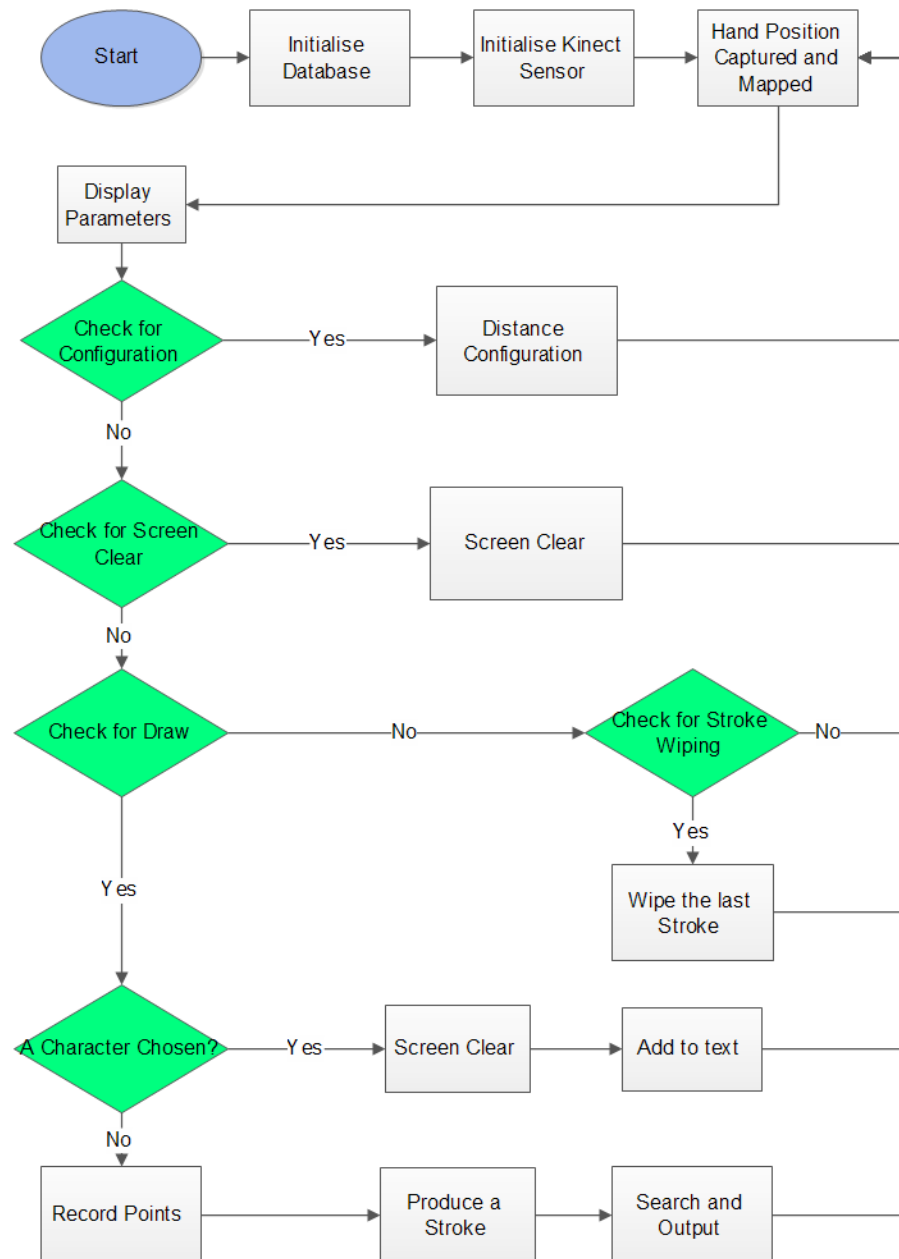


Figure 2.2.1

When the application is started, the database and the Kinect sensor will be initialised. At this point, the Kinect will continuously send frames to the computer. Due to the different position each time the user stands at, the threshold value for drawing need to be adjusted according to the value observed by the Kinect. Therefore, before the user can starting drawing, the threshold distance need to be set. This will be done by the user performing draw gesture.

As described in 2.1, strokes in chinese characters can be divided into two categories. The way the program recognise strokes will be count the number of simple strokes draw in one drawing gesture. Notice that a complex stroke is seen as several simple strokes drawn consecutively.

2.2.2 Result search

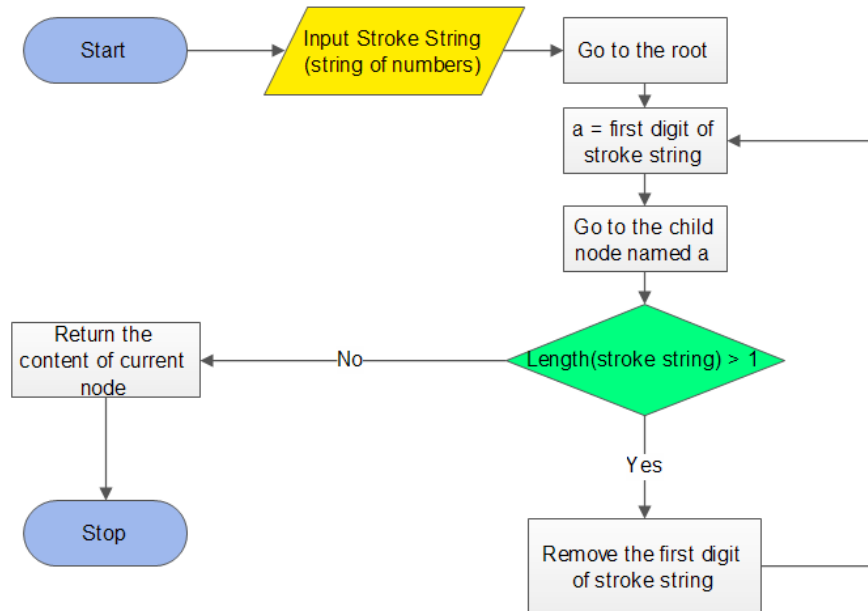


Figure 2.2.2

When the main program send a string of strokes, like 35251, to the database, the database will perform an algorithm that will find the node that represent 35251 in the tree. This algorithm is also used in data insertion during the initialisation of the database, so it is described first.

The algorithm can be described as below:

Starting at the root. Each time take the first digit of the stroke string, go to the child node that has its node name equal to the value. Remove the first digit of that string. Repeat that process until there is no more digits, then return the content of current node.

2.2.3 Data Insertion

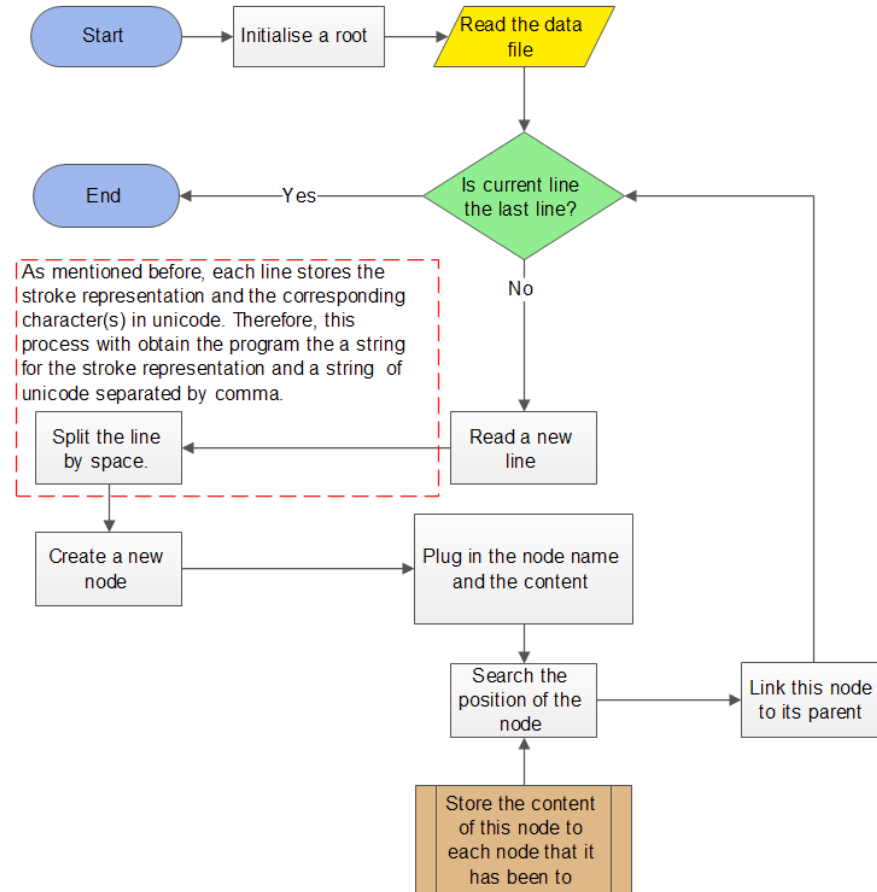


Figure 2.2.3

The algorithm starts with loading the data file into array of strings. Each string stores one line of the file. For example:

521 536B,5B50,5B51,5B52

After the first split, we can get the position the node should be and the content it should store. However, the node does not exist yet. So the we create a new node and by the searching algorithm, we can find and link to its parent node.

2.3 Test Strategy

2.3.1 Initialisation

ID	Case	Data	Expected Result
1.1	Distance Calibration Initialised	N/A	Distance Calibration window pops up
1.2	Main Window Initialised	N/A	Main Window pops up
1.3	Database Initialised	N/A	A five to ten seconds halt between the windows appear and start of tracking.
1.4	Kinect unit initialised	N/A	The programs can track the user.

2.3.2 Distance Calibration

The main purpose of this module is that if the Kinect is not fixed in a position, the reading of relative distance between user's left hand and a fixed point(say user's left shoulder) varies each time when the position of kinect and the angle to the user changes. Therefore, this module will help stabilise the draw gesture so that the user can have better control on the drawing process.

ID	Case	Data	Expected Result
2.1	Calibration window should be at the top of all windows	N/A	Calibration window appear at the top of all windows.
2.2	Distance correctly read	N/A	Reasonable value for distance displayed
2.3	Window shut after 3 seconds of stable readings	N/A	Window shut after 3 seconds of consecutive readings.
2.4	Value correctly returned	N/A	Able to draw when user's left hand fully reached out

2.3.3 Stroke Recognition

When a Kinect unit tracks a user, it sends the position of the user relative to its own sight which is excessively larger than the movable region of user's hand. To fix this, we can either have a huge window to represent the sight of the Kinect or we can 'reduce' the sight of the Kinect by mapping the raw data points to a 600*600 canvas by using the relative position of user's right hand and another point of his body(can be his right shoulder). Test 3.1 is used to test the mapping.

ID	Case	Data	Expected Result
3.1	Points correctly mapped to the screen	User's movement of right hand	The movement of the pointer is in the same direction as the user's right hand
3.2	Stroke type 1 correctly recognised	Draw a horizontal stroke with a slightly upward slope	The stroke should be recognised as type 1
3.3	Stroke type 2 correctly recognised	Draw a vertical stroke(or combination of 14)	The stroke should be recognised as type 2
3.4	Stroke type 3 correctly recognised	Draw a stroke from up right to down left (or down left to up right)	The stroke should be recognised as type 3
3.5	Stroke type 4 correctly recognised	Draw a stroke from up left to down right (or down right to up left)	The stroke should be recognised as type 4
3.6	Stroke type 5 correctly recognised	Draw multiple strokes in one draw gesture(not combination of 14)	The stroke should be recognised as type 5

2.3.4 Gestures

ID	Case	Data	Expected Result
4.1	Draw gesture recognised	Perform draw(reach out left hand as far as possible)	The pointer shrink into small point and while moving it, it leaves a trace on the screen
4.2	Clear screen	Hide both hands	All strokes should be cleared and all the option boxes should be gone
4.3	Delete last stroke	Move the right hand from left to right and back.	Last stroke should be deleted(if any)
4.4	Choose a character	Move the point to the desired character and perform a draw gesture	The character should be displayed in the text box and all strokes should be cleared and all the option boxes should be gone
4.5	Able to draw after a clear screen	Draw a stroke after clear screen	The stroke should be recognised correctly and all previous strokes should have no sign of existence
4.6	Able to draw after a stroke delete	Draw a stroke after a stroke delete	The stroke should be recognised correctly and the previous stroke should have no sign of existence

2.3.5 Database

The database is behind the screen so to check whether the output is correct, we need to read the data file and check it manually.

ID	Case	Data	Expected Result
5.1	Able to give correct output	strokes	Option boxes appear with characters that matches the strokes
5.2	Able to limit the number of output	strokes	6 option boxes maximum
5.3	Options less than 6	strokes	No empty option boxes appear
5.4	No matched characters	strokes	No option box

Chapter 3

Software Development and Testing

3.1 Alpha Testing

3.1.1 Kinect Tracking Parameters

```
var parameters = new TransformSmoothParameters
{
    Smoothing = 0.7f,
    Correction = 0.7f,
    Prediction = 0.1f,
    JitterRadius = 1.0f,
    MaxDeviationRadius = 1.0f
};
sensor.SkeletonStream.Enable(parameters);
```

By default, the Kinect only filter out small jitters to minimise latency. In this project, precision is much more important than minimising latency as stroke recognition only works when there is no abnormal input.

After trying some different sets of parameters listed in "<http://msdn.microsoft.com/en-us/library/jj131024.aspx>" the most suitable set is found.

3.1.2 Scaling Input to the Screen

```
private Point ScalePosition(Joint rightHand, Joint shoulder)
{
    double distX = rightHand.Position.X - shoulder.Position.X
        + 0.09;
    double distY = shoulder.Position.Y - rightHand.Position.Y
        + 0.36;
```

```

distX = distX / 0.45 * 600;
distY = distY / 0.45 * 600;

if (distX < 0 || distX > 600 || distY < 0 || distY > 600)
{
    return new Point(701, 701);
}
else
{
    return new Point(distX, distY);
}
}

```

Without a function to map the input, the Kinect will return the position of the user's joints relative to its view. Most part of the screen will be wasted as the user will not be able to reach the edges of the view. More importantly, the program will have a unnecessarily large window otherwise the pointer will likely to get lost.

While writing the calculation, it is important to make sure the sign is correct. Otherwise, the tracking can be reversed.

3.1.3 Input Selection

```

if (writingPoints.Count>0)
{
    if (newPoint.Equals(writingPoints[writingPoints.Count - 1]))
    {
        return;
    }
}
if (count < 4 && leaveOut)
{
    count++;
    return;
}
if (distance < 6)
{
    writingPoints.Remove(writingPoints[writingPoints.Count - 1]);
    return;
}
if (num != 0)
{
    strokeNumberList.Add(num);
}
else if (strokeNumberList.Count !=0)
{
    strokeNumberList.Add(strokeNumberList.Last());
}

```

```
}  
else  
{  
    writingPoints.Remove(writingPoints[writingPoints.Count -  
        1]);  
    return;  
}
```

During alpha test, I found that the pointer doodles a bit when starting a new stroke. Therefore, some selection rules are added like the first four points are ignored when starts a new stroke, same input is not recorded and only record new input if it is 6 pixels away from the previous one. These filters out a lot of abnormal inputs. In addition, there is a rule for points that does not make sense(if this point and the last point were to make a stroke, the stroke would not be recognised). Here, the program will assume that the new point is meant to follow the pattern of the stroke (which is usually the case) and record this point, otherwise if there is no previous point, do not record it.

```

Point(211.190059979757, 170.433661142985)    deleted
Point(212.055797576904, 170.689404805501)    deleted
Point(213.204617500305, 170.778334935506)    deleted
Point(214.940781593323, 170.634965896606)    deleted
Point(217.654302914937, 170.170288085937)
Point(221.433277130127, 169.32048479716)      deleted
Point(226.612006823222, 167.931299209595)    deleted
Point(233.283356030782, 166.066627502441)    deleted
Point(241.797839800517, 163.603843053182)    deleted
Point(251.977637608846, 160.691083272298)    deleted
Point(263.980701764425, 157.344719568888)    deleted
Point(277.378832499186, 153.78568649292)      deleted
Point(291.910683314006, 150.103232065837)    deleted
Point(307.432964642843, 146.455666224162)    deleted
Point(324.293926556905, 142.546714146932)    deleted
Point(341.460700035095, 138.738056818644)    deleted
Point(358.444844881693, 135.074119567871)    deleted
Point(374.915118217468, 131.629050572713)    deleted
Point(390.607511202494, 128.562669754028)    deleted
Point(405.340150197347, 125.799318949381)    deleted
Point(418.954765001933, 123.285353978475)
Point(431.394810676575, 120.990734100342)
Point(442.64769077301, 118.978719711304)
Point(452.619309425354, 117.216885884603)
Point(461.30581219991, 115.713338851929)
Point(468.78412882487, 114.46839650472)
Point(475.166832605998, 113.443752924601)
Point(480.548535982768, 112.631381352742)    deleted
Point(485.040938059489, 112.027228673299)
Point(488.784189224243, 111.592909495036)    deleted
:
Point(251.88894589742, 156.213582356771)
Point(223.49170366923, 159.928700129191)
Point(198.005115191142, 163.187007904053)
Point(175.118322372437, 165.913483301798)
Point(155.102923711141, 168.117504119873)
Point(137.851909001668, 169.830462137858)
Point(123.188331921895, 171.131989161173)
Point(110.833679835002, 172.078034083048)
Point(100.578720569611, 172.762454350789)
Point(92.1554517745972, 173.251371383667)
stroke finished: draw = false

```

We can see that the rejecting rate is high when starting a stroke, significantly higher than when finishing a stroke.

3.1.4 Stroke Pattern Recognition

```
int GetStrokeNum(double sin, Point first, Point second)
{
    if (first.X < second.X && sin >= -0.23 && sin < 0.23)
    {
        return 1;
    }
    else
        if (first.Y < second.Y && (sin >= 0.955 || sin <= -0.955))
        {
            return 2;
        }
    else
        if (first.X >= second.X && first.Y <= second.Y && sin
            >= 0.23 && sin <= 0.955)
        {
            return 3;
        }
    else
        if (first.X <= second.X && first.Y <= second.Y &&
            sin >= 0.23 && sin <= 0.955)
        {
            return 4;
        }
    else
        if (first.X >= second.X && first.Y >= second.Y &&
            sin <= -0.23 && sin >= -0.955)
        {
            return 5;
        }
    else
        if (first.X <= second.X && first.Y <= second.Y
            && sin <= 0.955 && sin >= 0.23)
        {
            return 6;
        }
    else
    {
        return 0;
    }
}
```

Stroke pattern recognition looks at the stroke pattern of two consecutive points and returns the stroke type the user is currently writing.

(217.654, 170.17)	(418.954, 123.285)	-0.226	1
(418.954, 123.285)	(431.394, 120.99)	-0.181	1
(431.394, 120.99)	(442.647, 118.978)	-0.176	1
(442.647, 118.978)	(452.619, 117.216)	-0.173	1

```

( 452.619, 117.216 ) ( 461.305, 115.713 ) -0.17 1
( 461.305, 115.713 ) ( 468.784, 114.468 ) -0.164 1
( 468.784, 114.468 ) ( 475.166, 113.443 ) -0.158 1
( 475.166, 113.443 ) ( 485.04, 112.027 ) -0.142 1
( 485.04, 112.027 ) ( 491.89, 111.298 ) -0.105 1
( 491.89, 111.298 ) ( 498.207, 111.05 ) -0.039 1
( 498.207, 111.05 ) ( 503.647, 113.772 ) 0.447 4
stroke finished: draw = false
( 372.87, 99.683 ) ( 368.245, 107.288 ) 0.854 3
( 368.245, 107.288 ) ( 365.741, 114.999 ) 0.951 3
( 365.741, 114.999 ) ( 363.269, 126.393 ) 0.977 2
( 363.269, 126.393 ) ( 362.121, 133.541 ) 0.987 2
( 362.121, 133.541 ) ( 361.186, 142.236 ) 0.994 2
( 361.186, 142.236 ) ( 360.443, 151.886 ) 0.997 2
( 360.443, 151.886 ) ( 359.962, 162.178 ) 0.998 2
( 359.962, 162.178 ) ( 359.691, 173.162 ) 0.999 2
( 359.691, 173.162 ) ( 359.64, 184.696 ) 0.999 2
( 359.64, 184.696 ) ( 359.753, 196.522 ) 0.999 2
( 359.753, 196.522 ) ( 360.027, 208.569 ) 0.999 2
( 360.027, 208.569 ) ( 360.4, 220.51 ) 0.999 2
( 360.4, 220.51 ) ( 360.844, 231.942 ) 0.999 2
( 360.844, 231.942 ) ( 361.304, 242.73 ) 0.999 2
( 361.304, 242.73 ) ( 361.775, 252.799 ) 0.998 2
( 361.775, 252.799 ) ( 362.244, 262.083 ) 0.998 2
( 362.244, 262.083 ) ( 362.686, 270.537 ) 0.998 2
( 362.686, 270.537 ) ( 363.093, 278.158 ) 0.998 2
( 363.093, 278.158 ) ( 363.452, 285.01 ) 0.998 2
( 363.452, 285.01 ) ( 363.763, 291.097 ) 0.998 2
( 363.763, 291.097 ) ( 364.291, 301.17 ) 0.998 2
( 364.291, 301.17 ) ( 364.657, 308.764 ) 0.998 2
( 364.657, 308.764 ) ( 364.963, 316.665 ) 0.999 2
( 364.963, 316.665 ) ( 364.866, 322.813 ) 0.999 2
stroke finished: draw = false
( 227.412, 146.279 ) ( 249.109, 141.628 ) -0.209 1
( 249.109, 141.628 ) ( 271.498, 137.551 ) -0.179 1
( 271.498, 137.551 ) ( 295.53, 134.132 ) -0.14 1
( 295.53, 134.132 ) ( 320.878, 131.326 ) -0.11 1
( 320.878, 131.326 ) ( 347.968, 128.662 ) -0.097 1
( 347.968, 128.662 ) ( 376.38, 126.229 ) -0.085 1
( 376.38, 126.229 ) ( 404.156, 124.338 ) -0.067 1
( 404.156, 124.338 ) ( 431.695, 122.895 ) -0.052 1
( 431.695, 122.895 ) ( 458.07, 121.861 ) -0.039 1
( 458.07, 121.861 ) ( 482.368, 121.086 ) -0.031 1
( 482.368, 121.086 ) ( 504.853, 120.599 ) -0.021 1
( 504.853, 120.599 ) ( 524.689, 120.395 ) -0.01 1
( 524.689, 120.395 ) ( 541.797, 120.438 ) 0.002 1

```

```

( 541.797, 120.438 ) ( 556.251, 120.661 ) 0.015 1
( 556.251, 120.661 ) ( 568.257, 120.998 ) 0.028 1
( 568.257, 120.998 ) ( 578.071, 121.406 ) 0.041 1
( 578.071, 121.406 ) ( 585.869, 121.866 ) 0.058 1
( 585.869, 121.866 ) ( 591.875, 122.316 ) 0.074 1
( 591.875, 122.316 ) ( 599.802, 123.202 ) 0.111 1
( 599.802, 123.202 ) ( 593.488, 126.45 ) 0.457 3
( 593.488, 126.45 ) ( 584.245, 127.378 ) 0.099 3
( 584.245, 127.378 ) ( 569.798, 128.635 ) 0.086 3
( 569.798, 128.635 ) ( 550.249, 130.142 ) 0.076 3
( 550.249, 130.142 ) ( 526.283, 131.61 ) 0.061 3
( 526.283, 131.61 ) ( 496.728, 133.461 ) 0.062 3
( 496.728, 133.461 ) ( 464.052, 135.536 ) 0.063 3
( 464.052, 135.536 ) ( 426.856, 138.138 ) 0.069 3
( 426.856, 138.138 ) ( 389.935, 140.989 ) 0.076 3
( 389.935, 140.989 ) ( 352.528, 144.399 ) 0.09 3
( 352.528, 144.399 ) ( 317.118, 148.128 ) 0.104 3
( 317.118, 148.128 ) ( 283.096, 152.16 ) 0.117 3
( 283.096, 152.16 ) ( 251.888, 156.213 ) 0.128 3
( 251.888, 156.213 ) ( 223.491, 159.928 ) 0.129 3
( 223.491, 159.928 ) ( 198.005, 163.187 ) 0.126 3
( 198.005, 163.187 ) ( 175.118, 165.913 ) 0.118 3
( 175.118, 165.913 ) ( 155.102, 168.117 ) 0.109 3
( 155.102, 168.117 ) ( 137.851, 169.83 ) 0.098 3
( 137.851, 169.83 ) ( 123.188, 171.131 ) 0.088 3
( 123.188, 171.131 ) ( 110.833, 172.078 ) 0.076 3
( 110.833, 172.078 ) ( 100.578, 172.762 ) 0.066 3
( 100.578, 172.762 ) ( 92.155, 173.251 ) 0.057 3
stroke finished: draw = false

```

3.1.5 Stroke Recognition

```
void CheckStroke(List<int> strokeNumList)
{
    if (strokeNumberList.Count < 4)
    {
        return;
    }
    int first = strokeNumList[strokeNumList.Count - 4];
    int second = strokeNumList[strokeNumList.Count - 3];
    int third = strokeNumList[strokeNumList.Count - 2];
    int fourth = strokeNumList[strokeNumList.Count - 1];
    if (first == second && first == third && first == fourth
        && first != 0)
    {
        CheckForNewStroke(first);
        currentStroke = first;
    }
}

// check if it is a new stroke
void CheckForNewStroke(int strokeNum)
{
    if (currentStroke == strokeNum)
    {
        return;
    }
    if (currentStroke == 0)
    {
        return;
    }
    if (currentStroke != strokeNum)
    {
        SaveStroke(true);
        leaveOut = false;
        count = 4;
        Point[] points = new Point[5];
        for (int i = 0; i < points.Length; i++)
        {
            points[i] = writingPoints[writingPoints.Count - 5 + i];
        }
        writingPoints.Clear();
        for (int i = 0; i < points.Length; i++)
        {
            writingPoints.Add(points[i]);
        }
    }
}
```

Many chinese characters have at least one type 5 stroke (a combination of several strokes, see section 2.1.2). This means the program need the ability to recognise the change of the stroke pattern. Here the program sees a new stroke pattern when the last four inputs makes a new pattern. Each strokes are recorded because when each stroke is drawn, I replace the dots by lines. Each stroke will be missing its first four points (either ignored or stored in the last stroke) and also storing the first four points of the next stroke (because it takes four points to realise a new stroke is started). Therefore when a new stroke pattern is seen, the last five points from its previous stroke is stored in the new stroke otherwise the strokes seen on the screen will not the connected.

3.1.6 Wiping Gesture Recognition

```
if (newPoint.X > 250 && wipe.leftMost == 600)
{
    return;
}
if (wipe.phase == Phase.LeftToRight)
{
    if (newPoint.X < wipe.leftMost)
    {
        wipe.leftMost = newPoint.X;
    }
    if (newPoint.X > wipe.rightMost)
    {
        wipe.rightMost = newPoint.X;
    }
    if (wipe.rightMost - wipe.leftMost >= 300)
    {
        wipe.phase = Phase.RightToLeft;
    }
}
else
if (wipe.phase == Phase.RightToLeft)
{
    if (newPoint.X <= wipe.leftMost)
    {
        RemoveStroke();
    }
}
```

The first design of wipe gesture, the only restriction was it must be perform when the user is not drawing. The mechanism is that when the user is not drawing, the program records the left most point and the right most point reached by the pointer and see if the separation between them is wide enough. If the separation is sufficiently wide, when the pointer moves back to the left most point, the program will wipe the last stroke.

However, a problem occurs when the user finishes a character with the pointer on the right side of the screen, when he tries to choose the left most character displayed in the option box, the program will wipe the last stroke.

Therefore, another restriction was added, the program changes the initial value(start recording) only if the pointer is on the left half of the screen.

69.4035243988037	69.4035243988037	LeftToRight
63.45490137736	69.4035243988037	LeftToRight
56.133861541748	69.4035243988037	LeftToRight
49.1657606760661	69.4035243988037	LeftToRight
42.537088394165	69.4035243988037	LeftToRight
36.366974512736	69.4035243988037	LeftToRight
31.0924402872721	69.4035243988037	LeftToRight
26.7742824554443	69.4035243988037	LeftToRight
23.7228425343831	69.4035243988037	LeftToRight
21.9138018290202	69.4035243988037	LeftToRight
21.3614654541016	69.4035243988037	LeftToRight
21.3614654541016	69.4035243988037	LeftToRight
21.3614654541016	69.4035243988037	LeftToRight
21.3614654541016	69.4035243988037	LeftToRight
21.3614654541016	69.4035243988037	LeftToRight
21.3614654541016	69.4035243988037	LeftToRight
21.3614654541016	69.4035243988037	LeftToRight
21.3614654541016	69.4035243988037	LeftToRight
21.3614654541016	69.4035243988037	LeftToRight
21.3614654541016	70.4470825195312	LeftToRight
21.3614654541016	84.8598035176595	LeftToRight
21.3614654541016	100.046590169271	LeftToRight
21.3614654541016	115.431423187256	LeftToRight
21.3614654541016	130.640621185303	LeftToRight
21.3614654541016	145.185346603394	LeftToRight
21.3614654541016	159.422671000163	LeftToRight
21.3614654541016	172.611668904622	LeftToRight
21.3614654541016	184.617951711019	LeftToRight
21.3614654541016	195.297673543294	LeftToRight
21.3614654541016	204.718386332194	LeftToRight
21.3614654541016	212.952092488607	LeftToRight
21.3614654541016	220.092649459839	LeftToRight
:		
26.3119093577067	252.867972056071	LeftToRight
26.3119093577067	256.074225107829	LeftToRight
26.3119093577067	268.061434427897	LeftToRight
26.3119093577067	276.992117563883	LeftToRight
26.3119093577067	284.206584294637	LeftToRight
26.3119093577067	290.026540756226	LeftToRight
26.3119093577067	296.959276199341	LeftToRight
26.3119093577067	303.735211690267	LeftToRight
26.3119093577067	311.506385803223	LeftToRight
26.3119093577067	319.48689142863	LeftToRight
26.3119093577067	330.206906000773	RightToLeft

Stroke Wiped

3.2 Source Code

3.2.1 Main Window

MainWindow.xaml

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
        presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:WpfViewers="clr-namespace:Microsoft.Samples.Kinect
        .WpfViewers;assembly=Microsoft.Samples.Kinect.
        WpfViewers" x:Class="WritingInAir.MainWindow"
    Title="MainWindow" Height="672" Width="616" Loaded="
        Window_Loaded_1" Closing="Window_Closing_1">
    <Canvas x:Name="canvas1" HorizontalAlignment="Left" Height
        ="640" VerticalAlignment="Top" Width="610">
        <Label x:Name="leftHandLabel" Content="LeftHand: "
            Canvas.Left="10" Canvas.Top="10"/>
        <Label x:Name="rightHandLabel" Content="RightHand: "
            Canvas.Left="10" Canvas.Top="25"/>
        <WpfViewers:KinectSensorChooser x:Name="
            kinectSensorChooser1" Canvas.Left="97" Height="224"
            Width="248"/>
        <Ellipse x:Name="hand" Fill="#000000" Height="30" Canvas
            .Left="560" Canvas.Top="10" Width="30"/>
        <Label x:Name="drawLabel" Content="Draw: " Canvas.Left="
            10" Canvas.Top="45" />
        <Label x:Name="shoulderLabel" Content="Head:" Canvas.
            Left="10" Canvas.Top="-6"/>
        <Canvas x:Name="ellipseCanvas" Canvas.Top="0" Canvas.
            Left="0" Height="600" Width="600"/>
        <TextBox x:Name="text" Canvas.Bottom="10" Canvas.Left="0
            " Width="600" Height="20"/>
    </Canvas>
</Window>
```

MainWindow.xaml.cs

```
//
//  MainWindow.xaml.cs
//  WritingInAir
//
//  Created by Hu on 15/03/2014
//  Copyright 4014 __MyCompanyName__. All rights reserved.
//

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Microsoft.Kinect;
using Coding4Fun.Kinect.Wpf;

namespace WritingInAir
{
    public partial class MainWindow : Window
    {
        bool closing = false; // true when the window is closed

        const int skeletonCount = 6; // maximum number of
            skeleton the kinect can track
        Skeleton[] allSkeletons = new Skeleton[skeletonCount];

        //radius of pointer
        const int radiusHover = 30;
        const int radiusWrite = 10;

        bool draw = false; // only record the points when draw
            == true
        bool config = true; // whether or not the distance need
            to be calibrated
        bool newValue = true; // whether or not the first value
            has been sent

        int currentStroke = 0; // initialise the stroke number
        // a list of points recorded
    }
}
```

```

private List<Point> writingPoints = new List<Point>(1);
// a list of stroke number
private List<int> strokeNumberList = new List<int>(1);
private List<int> strokeIn = new List<int>(1);
private List<int> strokeOutput = new List<int>(1);
// a list of strokes
private List<Stroke> strokeList = new List<Stroke>(1);
private StrokeTree tree;

/* When starting a stroke, it is found that the hand
   cannot stay statically.
   * This causes the system draw a small circle at that
   position which
   * might be recognised as other strokes.
   * Thus, the first 4 points are ignored to improve the
   precision.*/
int count = 0;
bool leaveOut = true;

DistanceConfig myConfig = new DistanceConfig();
double writingDistance = 0;

//initialise 6 option boxes
Rectangle[] rects = new Rectangle[6];
Label[] options = new Label[6];

/* create an enumerable type "Phase"
   * used in recognising wiping gesture*/
public enum Phase {LeftToRight = 0, RightToLeft = 1};

/* declare a new data structure
   * used to store data for recognising wipe gesture*/
struct Wipe
{
    public double leftMost;
    public double rightMost;
    public Phase phase;
}
Wipe wipe = new Wipe();

public MainWindow()
{
    //initilise the program
    myConfig.Show();
    InitializeComponent();
    tree = InitialiseDatabase();
    InitialiseInterface();
    ClearWipe();
    text.Text = ""
    /* put the distance calibration at the top

```

```

    * of all windows for user to see*/
    myConfig.Topmost = true;
}

////////////////////////////////////
//----- Stroke Class -----//
//--- Stores Stroke as two points and a number ---//
public class Stroke
{
    public Stroke(Point first, Point second, int strokeNum
        , bool link)
    {
        start = first;
        end = second;
        strokeNumber = strokeNum;
        linked = link;
    }
    public Point start { get; set; }
    public Point end { get; set; }
    public int strokeNumber { get; set; }
    public bool linked { get; set; }
}
//-----//
////////////////////////////////////

////////////////////////////////////
//----- Kinect Initialisation -----//
void kinectSensorChooser1_KinectSensorChanged(object
    sender, DependencyPropertyChangedEventArgs e)
{
    KinectSensor old = (KinectSensor)e.OldValue;
    StopKinect(old);
    KinectSensor sensor = (KinectSensor)e.NewValue;

    if (sensor == null)
    {
        return;
    }

    // parameters for smoothing the motion recorded
    var parameters = new TransformSmoothParameters
    {
        Smoothing = 0.7f,
        Correction = 0.7f,
        Prediction = 0.1f,
        JitterRadius = 1.0f,
        MaxDeviationRadius = 1.0f
    };

    sensor.SkeletonStream.Enable(parameters);
}

```

```

        sensor.SkeletonStream.EnableTrackingInNearRange = true
        ;
        sensor.AllFramesReady += sensor_AllFramesReady;
        sensor.DepthStream.Enable(DepthImageFormat.
            Resolution640x480Fps30);
        sensor.ColorStream.Enable(ColorImageFormat.
            RgbResolution640x480Fps30);

        try
        {
            sensor.Start();
        }
        catch (System.IO.IOException)
        {
            kinectSensorChooser1.AppConflictOccurred();
        }
    }

    void sensor_AllFramesReady(object sender,
        AllFramesReadyEventArgs e)
    {
        //when the Kinect is ready, tracking begins
        if (closing)
        {
            return;
        }

        Skeleton first = GetFirstSkeleton(e);
        GetCameraPoint(first, e);
    }

    Skeleton GetFirstSkeleton(AllFramesReadyEventArgs e)
    {
        using (SkeletonFrame skeletonFrameData = e.
            OpenSkeletonFrame())
        {
            if (skeletonFrameData == null)
            {
                return null;
            }

            skeletonFrameData.CopySkeletonDataTo(allSkeletons);

            /* only the first, usually the most active skeleton
             * is recognised as the writer */
            Skeleton first = (from s in allSkeletons
                             where s.TrackingState ==
                                 SkeletonTrackingState.Tracked
                             select s).FirstOrDefault();

            return first;
        }
    }

```



```

}

//-----//
////////////////////////////////////

////////////////////////////////////
//----- Camera Adjusting -----//
void GetCameraPoint(Skeleton first,
    AllFramesReadyEventArgs e)
{
    using (DepthImageFrame depth = e.OpenDepthImageFrame()
        )
    {
        if (depth == null || kinectSensorChooser1.Kinect ==
            null || first == null)
        {
            return;
        }
        /* set the contents of labels according to the
           tracking state of Joints
           * here we track left hand, right hand and left
           shoulder
           * tracking state can be Tracked, Infered or Not
           Tracked */
        leftHandLabel.Content = "Left Hand:" + Convert.
            ToString(first.Joints[JointType.HandLeft].
                TrackingState);
        rightHandLabel.Content = "Right Hand:" + Convert.
            ToString(first.Joints[JointType.HandRight].
                TrackingState);
        shoulderLabel.Content = "ShoulderLeft:" + Convert.
            ToString(first.Joints[JointType.ShoulderLeft].
                TrackingState);

        //obtain a point for right hand(the Pointer) by
        scalePosition function
        Point rightHand = ScalePosition(first.Joints[
            JointType.HandRight], first.Joints[JointType.
                ShoulderRight]);

        //calculate the distance between the left shoulder
        and the left hand, keep 6 decimal places
        double x = first.Joints[JointType.ShoulderLeft].
            Position.X - first.Joints[JointType.HandLeft].
                Position.X;
        double y = first.Joints[JointType.ShoulderLeft].
            Position.Y - first.Joints[JointType.HandLeft].
                Position.Y;
    }
}

```

```

double z = first.Joints[JointType.ShoulderLeft].
    Position.Z - first.Joints[JointType.HandLeft].
    Position.Z;
double distance = Math.Truncate(Math.Sqrt(x * x + y
    * y + z * z) * 1000000) / 1000000;

if (writingDistance != 0)
{
    config = false;
    myConfig.Close();
}
//open the distance calibration if needed
if (config)
{
    if (newValue)
    {
        myConfig.Trig(distance);
        newValue = false;
    }
    else
    {
        writingDistance = myConfig.Count(distance);
    }
}
else
{
    // reaching out left arm to start drawing
    if (distance > writingDistance && first.Joints[
        JointType.HandRight].TrackingState ==
        JointTrackingState.Tracked && first.Joints[
        JointType.HandLeft].TrackingState ==
        JointTrackingState.Tracked && !config)
    {
        ChangePointerToWrite();
    }
    else
    {
        ChangePointerToHover();
    }

    // label display the distance from the left hand
    to the left shoulder
    drawLabel.Content = "Draw = " + Convert.ToString(
        distance);

    //if the hand is not in the effective tracking
    region, scalePosition function returns Point
    (601,601)
    if (rightHand.X != 701 && rightHand.Y != 701)
    {

```

```

        Canvas.SetTop(hand, rightHand.Y - hand.Height /
            2);
        Canvas.SetLeft(hand, rightHand.X - hand.Width /
            2);
        HandTracking(rightHand);
    }
    // clear the screen when both hands are hidden
    if (first.Joints[JointType.HandRight].
        TrackingState == JointTrackingState.Inferred &&
        first.Joints[JointType.HandLeft].TrackingState
        == JointTrackingState.Inferred)
    {
        ClearAll();
    }
}
}

private Point ScalePosition(Joint rightHand, Joint
    shoulder)
{
    /* ScalePosition function output the relative position
       of right hand to the right shoulder the effective
       region is -0.09m to 0.36m to the position of both
       horizontally and vertically the relative position
       is translated into the position in 600pix *600pix
       canvas.
       * If right hand is out of the region, it will return
       Point(601,601)*/
    double distX = rightHand.Position.X - shoulder.
        Position.X + 0.09;
    double distY = shoulder.Position.Y - rightHand.
        Position.Y + 0.36;

    distX = distX / 0.45 * 600;
    distY = distY / 0.45 * 600;

    if (distX < 0 || distX > 600 || distY < 0 || distY >
        600)
    {
        return new Point(701, 701);
    }
    else
    {
        return new Point(distX, distY);
    }
}

//-----//
////////////////////

```

```

////////////////////////////////////
//----- Pointer -----//
private void ChangePointerToHover()
{
    draw = false;
    hand.Height = radiusHover;
    hand.Width = radiusHover;

    /* When the pointer is back to hover,
     * it indicates the end of a complete stroke.
     * Therefore, save it and display the possible
     * characters*/
    if (writingPoints.Count > 3 && currentStroke != 0)
    {
        SaveStroke(currentStroke, false);
        writingPoints.Clear();
        CombineAndOut();
        DisplayChoices(strokeOutput);
    }
}

private void ChangePointerToWrite()
{
    draw = true;
    leaveOut = true;
    hand.Height = radiusWrite;
    hand.Width = radiusWrite;
    //When starting a new stroke, wipe is reset
    ClearWipe();
}
//-----//
////////////////////////////////////

////////////////////////////////////
//----- Writing Events -----//

private void HandTracking(Point newPoint)
{
    if (!draw)//wipe can only be done while not drawing
    {
        if (newPoint.X > 250 && wipe.leftMost == 600)
        {
            //this prevent wiping when a stroke is ended at the
            //right half of the screen and then moving the
            //pointer to the left
            return;
        }
    }
}

```

```

}
if (wipe.phase == Phase.LeftToRight)
{
    /* Wipe will be recognised if the pointer moves
    * first from left to right and back.
    * This part is used to detect see if the
    * condition meets
    * to move onto the next phase.
    * This phase will be ended if the pointer moves
    * from left to right across at least half the
    * screen*/
    if (newPoint.X < wipe.leftMost)
    {
        wipe.leftMost = newPoint.X;
    }
    if (newPoint.X > wipe.rightMost)
    {
        wipe.rightMost = newPoint.X;
    }
    if (wipe.rightMost - wipe.leftMost >= 300)
    {
        wipe.phase = Phase.RightToLeft;
    }
}
else
    if (wipe.phase == Phase.RightToLeft)
    {
        /* In the second phase, if the pointer comes
        back
        * the gesture is recognised.*/
        if (newPoint.X <= wipe.leftMost)
        {
            RemoveStroke();
        }
    }
return;
}
if (newPoint.Y >= 495 && newPoint.Y <= 590)
{
    int x = Convert.ToInt32(Math.Truncate((newPoint.X -
        5) / 100));
    if (text.Text.Length != 0)
    {
        text.Text = text.Text.Insert(text.Text.Length - 1,
            options[x].Content.ToString());
    }
    else
    {
        try
        {

```

```

        text.Text = options[x].Content.ToString();
    }
    catch (Exception)
    {
    }
}
ClearAll();
return;
}
if (writingPoints.Count>0)
{
    // check if new point is on the same position as the
    // last
    if (newPoint.Equals(writingPoints[writingPoints.
        Count - 1]))
    {
        return;
    }
}
if (count < 4 && leaveOut)
{
    count++;
    return;
}
writingPoints.Add(newPoint); // otherwise record the
    new point
if (writingPoints.Count <= 1)
{
    DrawEllipse(newPoint);
    return;
}
//obtain the stroke number
int num = GetStrokeNum(Sine(writingPoints),
    writingPoints[writingPoints.Count - 2],
    writingPoints[writingPoints.Count - 1]);
//obtain the latest displacement
double distance = dist(writingPoints);
//only record the meaningful stroke and the
    displacement over 6 pixels
if (distance < 6)
{
    writingPoints.Remove(writingPoints[writingPoints.
        Count - 1]);
    return;
}
if (num != 0)
{
    strokeNumberList.Add(num);
}
else if (strokeNumberList.Count !=0)

```

```

        {
            strokeNumberList.Add(strokeNumberList.Last());
        }
        else
        {
            writingPoints.Remove(writingPoints[
                writingPoints.Count - 1]);
        }
        DrawEllipse(newPoint);

        if (strokeNumberList.Count > 3)
        {
            CheckStroke(strokeNumberList);
        }
    }

    //The point is displayed as a circle
    private void DrawEllipse(Point point)
    {
        Ellipse newEllipse = new Ellipse();
        Brush myBrush = new SolidColorBrush(Color.FromRgb(0,
            0, 0));
        newEllipse.Height = radiusWrite;
        newEllipse.Width = radiusWrite;
        newEllipse.StrokeThickness = 0;
        newEllipse.Fill = myBrush;
        ellipseCanvas.Children.Add(newEllipse);
        Canvas.SetLeft(newEllipse, point.X - newEllipse.Width
            / 2);
        Canvas.SetTop(newEllipse, point.Y - newEllipse.Height
            / 2);
    }

    // draws a stroke as a straight line
    private void drawLine(Stroke stroke1)
    {
        Line newLine = new Line();
        Brush myBrush = new SolidColorBrush(Color.FromRgb(0,
            0, 0));
        newLine.X1 = stroke1.start.X;
        newLine.Y1 = stroke1.start.Y;
        newLine.X2 = stroke1.end.X;
        newLine.Y2 = stroke1.end.Y;
        newLine.Stroke = myBrush;
        newLine.StrokeThickness = 2;
        ellipseCanvas.Children.Add(newLine);
    }

    private void ClearAll()
    {

```

```

ellipseCanvas.Children.Clear();
strokeList.Clear();
strokeNumberList.Clear();
writingPoints.Clear();
strokeOutput.Clear();
count = 0;
for (int i = 0; i < 6; i++)
{
    rects[i].Stroke = new SolidColorBrush(Color.FromRgb
        (255, 255, 255));
    options[i].Content = "";
}
}
//-----//
////////////////////////////////////

////////////////////////////////////
//----- Stroke Related Functions -----//

int GetStrokeNum(double sin, Point first, Point second)
{
    if (first.X < second.X && sin >= -0.23 && sin < 0.23)
    {
        return 1;
    }
    else
    {
        if (first.Y < second.Y && (sin >= 0.955 || sin
            <=-0.955))
        {
            return 2;
        }
        else
        {
            if (first.X >= second.X && first.Y <= second.Y &&
                sin >= 0.23 && sin <= 0.955)
            {
                return 3;
            }
        }
        else
        {
            if (first.X <= second.X && first.Y <= second.Y
                && sin >= 0.23 && sin <= 0.955)
            {
                return 4;
            }
        }
        else
        {
            if (first.X >= second.X && first.Y >= second.Y
                && sin <= -0.23 && sin >= -0.955)
            {
                return 5;
            }
        }
    }
}

```



```

        else
            if (first.X <= second.X && first.Y <= second
                .Y && sin <= 0.955 && sin >= 0.23)
            {
                return 6;
            }
            else
            {
                return 0;
            }
    }

    // recognise a stroke only when last 4 numbers are the
    // same
    void CheckStroke(List<int> strokeNumList)
    {
        if (strokeNumList.Count < 4)
        {
            return;
        }
        int first = strokeNumList[strokeNumList.Count - 4];
        int second = strokeNumList[strokeNumList.Count - 3];
        int third = strokeNumList[strokeNumList.Count - 2];
        int fourth = strokeNumList[strokeNumList.Count - 1];
        if (first == second && first == third && first ==
            fourth && first != 0)
        {
            CheckForNewStroke(first);
            currentStroke = first;
        }
    }

    // check if it is a new stroke
    void CheckForNewStroke(int strokeNum)
    {
        if (currentStroke == strokeNum)
        {
            return;
        }
        if (currentStroke == 0)
        {
            return;
        }
        if (currentStroke != strokeNum)
        {
            SaveStroke(true);
            leaveOut = false;
            count = 4;
            Point[] points = new Point[5];
            for (int i = 0; i < points.Length; i++)

```

```

        {
            points[i] = writingPoints[writingPoints.Count - 5 + i];
        }
        writingPoints.Clear();
        for (int i = 0; i < points.Length; i++)
        {
            writingPoints.Add(points[i]);
        }
    }
}

void SaveStroke(bool link)
{
    //prepare for next stroke
    Point first = writingPoints[0];
    Point second = writingPoints[writingPoints.Count - 1];
    if (link)
    {
        second = writingPoints[writingPoints.Count - 5];
    }
    else
    {
        second = writingPoints[writingPoints.Count - 1];
    }
    Stroke newStroke = new Stroke(first, second,
        currentStroke, link);
    strokeList.Add(newStroke);
    strokeIn.Add(currentStroke);
    ellipseCanvas.Children.Clear();
    count = 0;
    currentStroke = 0;
    leaveOut = false;
    /* last step clear everything on the screen
     * here we redraw every stroke back*/
    for (int i = 0; i < strokeList.Count; i++)
    {
        drawLine(strokeList[i]);
    }
}

/* This function reads what is drawn in the last stroke
 * and analyses the stroke(s) as some strokes are
 * combination of others. */
void CombineAndOut()
{
    bool tf = true;
    for (int i = 0; i < strokeIn.Count(); i++)
    {
        if (!strokeIn[i].Equals(strokeIn[0]))

```

```

    {
        tf = false;
    }
}
if (tf)
{
    if (strokeIn[0] <= 4)
    {
        strokeOutput.Add(strokeIn[0]);
    }
    else if (strokeIn[0] == 5)
    {
        strokeOutput.Add(4);
    }
    else if (strokeIn[0] == 6)
    {
        strokeOutput.Add(3);
    }
}
else
{
    switch (strokeIn.Count())
    {
        case 1: if (strokeIn[0] <= 4)
            {
                strokeOutput.Add(strokeIn[0]);
            }
            else if (strokeIn[0] == 5)
            {
                strokeOutput.Add(4);
            }
            else if (strokeIn[0] == 6)
            {
                strokeOutput.Add(3);
            }
            break;
        case 2: if (strokeIn[0] == 2 && strokeIn[1] == 5)
            {
                strokeOutput.Add(2);
            }
            else
            {
                if (strokeIn[0] == 1 && strokeIn[1] == 4)
                {
                    strokeOutput.Add(4);
                }
                else if (strokeIn[0] == 2 && strokeIn[1] == 3)
                {
                    strokeOutput.Add(3);
                }
            }
        }
    }
}

```

```

        }
        else if (strokeIn[0] == strokeIn[1])
        {
            strokeOutput.Add(strokeIn[0]);
        }
        else
        {
            strokeOutput.Add(5);
        }
        break;
    case 3: strokeOutput.Add(5);
        break;
    default: strokeOutput.Add(5);
        break;
    }
}
strokeIn.Clear();
}

public void RemoveStroke()
{
    ClearWipe();
    if (strokeOutput.Count != 0)
    {
        strokeOutput.Remove(strokeOutput.Last());
    }
    if (strokeList.Count != 0)
    {
        strokeList.RemoveAt(strokeList.Count - 1);
    }
    try
    {
        while (strokeList.Last().linked)
        {
            strokeList.RemoveAt(strokeList.Count - 1);
        }
    }
    catch (Exception)
    {
    }
    ellipseCanvas.Children.Clear();
    for (int i = 0; i < strokeList.Count; i++)
    {
        drawLine(strokeList[i]);
    }
    if (strokeOutput.Count != 0)
    {
        DisplayChoices(strokeOutput);
    }
    else

```

```

{
    for (int i = 0; i < 6; i++)
    {
        rects[i].Stroke = new SolidColorBrush(Color.
            FromRgb(255, 255, 255));
        options[i].Content = "";
    }
}

//-----//
////////////////////////////////////

////////////////////////////////////
//----- Interface -----//
void DisplayChoices(List<int> tList)
{
    string t = "";
    for (int i = 0; i < tList.Count; i++)
    {
        t = t.Insert(i, Convert.ToString(tList[i]));
    }
    List<string> resultList = tree.Search(t);
    char[] result = new char[resultList.Count];
    for (int i = 0; i < resultList.Count; i++)
    {
        result[i] = (char)int.Parse(resultList[i], System.
            Globalization.NumberStyles.HexNumber);
    }
    for (int i = 0; i < 6; i++)
    {
        try
        {
            rects[i].Stroke = new SolidColorBrush(Color.
                FromRgb(0, 0, 0));
            options[i].Content = result[i];
        }
        catch(Exception e)
        {
            rects[i].Stroke = new SolidColorBrush(Color.
                FromRgb(255, 255, 255));
            options[i].Content = "";
        }
    }
}

void ClearWipe()
{
    wipe.leftMost = 600;
    wipe.rightMost = 0;
    wipe.phase = Phase.LeftToRight;
}

```

```

}

void InitialiseInterface()
{
    for (int i = 0; i < 6; i++)
    {
        rects[i] = new Rectangle();
        Canvas.SetTop(rects[i], 495);
        Canvas.SetLeft(rects[i], 5 + i * 100);
        rects[i].Height = 95;
        rects[i].Width = 95;
        rects[i].Stroke = new SolidColorBrush(Color.FromRgb(
            0, 0, 0));
        options[i] = new Label();
        Canvas.SetTop(options[i], 495);
        Canvas.SetLeft(options[i], 17 + i * 100);
        options[i].FontSize = 60;
        canvas1.Children.Add(rects[i]);
        canvas1.Children.Add(options[i]);
    }
}

//-----//
//Customer Defined Calculation functions -----//
double Sine(List<Point> tempList)
{
    double changeInX = tempList[tempList.Count - 1].X -
        tempList[tempList.Count - 2].X;
    double changeInY = tempList[tempList.Count - 1].Y -
        tempList[tempList.Count - 2].Y;
    double distance = Math.Sqrt(changeInX * changeInX +
        changeInY * changeInY);
    return changeInY / distance;
}

double dist(List<Point> tempList)
{
    double changeInX = tempList[tempList.Count - 1].X -
        tempList[tempList.Count - 2].X;
    double changeInY = tempList[tempList.Count - 1].Y -
        tempList[tempList.Count - 2].Y;
    double distance = Math.Sqrt(changeInX * changeInX +
        changeInY * changeInY);
    return distance;
}

double KeepThreeDecimals(double x)

```

```

{
    return Math.Truncate(x * 1000) / 1000;
}
//-----//
////////////////////////////////////

////////////////////////////////////
//----- Window Events -----//
private void Window_Loaded_1(object sender,
    RoutedEventArgs e)
{
    kinectSensorChooser1.KinectSensorChanged +=
        kinectSensorChooser1_KinectSensorChanged;
}

private void Window_Closing_1(object sender, System.
    ComponentModel.CancelEventArgs e)
{
    closing = true;
    StopKinect(kinectSensorChooser1.Kinect);
}

private void StopKinect(KinectSensor sensor)
{
    if (sensor != null)
    {
        if (sensor.IsRunning)
        {
            sensor.Stop();
            if (sensor.AudioSource != null)
            {
                sensor.AudioSource.Stop();
            }
        }
    }
}
//-----//
////////////////////////////////////
}
}

```

DistanceConfig.xaml

```

<Window x:Class="WritingInAir.DistanceConfig"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
        presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mc="http://schemas.openxmlformats.org/markup-
        compatibility/2006"

```

```

xmlns:d="http://schemas.microsoft.com/expression/blend
/2008"
mc:Ignorable="d" Height="159" Width="639">
<Grid>
<Label Content="Extend your left arm as much as possible
and hold for 3 seconds..." FontSize="20" Margin="20"
/>
<Label x:Name="d" Content="Distance from your left hand
to your left shoulder is : " Margin="20,60,113,10"/>
<Label Content="This value will be used as the threshold
for detecting a draw action." HorizontalAlignment="
Left" Margin="20,83,0,0" VerticalAlignment="Top"/>
</Grid>
</Window>

```

DistanceConfig.xaml.cs

```

//
// DistanceConfig.xaml.cs
// WritingInAir
//
// Created by Hu on 15/03/2014
// Copyright 4014 __MyCompanyName__. All rights reserved.
//

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Timers;

namespace WritingInAir
{
    public partial class DistanceConfig : Window
    {
        public DistanceConfig()
        {
            InitializeComponent();

            //add an elapsed time event
            mytimer.Elapsed += mytimer_Elapsed;

```



```

}

//set my timer to only live for 3 seconds
Timer mytimer = new Timer(3000);
double contrast;
double distance = 0;

public void Trig(double f)
{
    /* get f from the kinect
     * this method is used when first value is sent
     * or when the recorded distance is unstable
     * so that the timer is reset */
    mytimer.AutoReset = false;
    Reset(mytimer);
    contrast = f;
    d.Content = "Distance from your left hand to your left
        shoulder is : " + Convert.ToString(f);
}

private void Reset(Timer timer)
{
    timer.Stop();
    timer.Start();
}

public double Count(double f)
{
    if (distance != 0)
    {
        //to reduce stress on user, we return a slightly
        smaller value
        return distance*0.9;
    } else
        /* the variation of distance cannot be more than 3
        cm
        * from the first value */
        if (Math.Abs(f - contrast) > 0.03)
        {
            Trig(f);
            return 0;
        }
        else
        {
            d.Content = "Distance from your left hand to
                your left shoulder is : " + Convert.ToString(
                f);
            return 0;
        }
    }
}

```

```

        void mytimer_Elapsed(object sender, ElapsedEventArgs e)
        {
            //when the distance is stable for 3 seconds, we can
            //set the threshold
            distance = contrast;
        }
    }
}

```

Database.cs

```

//
// Database.cs
// WritingInAir
//
// Created by Hu on 15/03/2014
// Copyright 4014 __MyCompanyName__. All rights reserved.
//

using System;
using System.Windows;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Text;
using System.Threading.Tasks;

namespace WritingInAir
{
    public partial class MainWindow : Window
    {
        public class Node
        {
            public Node[] children = new Node[5];
            public int nodeName; // This stores the stroke type
            public List<string> nodeContent = new List<string>(1);
            //This stores first 6 possible characters

            public Node(int id)
            {
                nodeName = id;
                nodeContent = null;
            }
        }

        //The stroke tree class will hold the database and any
        //methods
        public class StrokeTree

```

```

{
    public Node root;
    public StrokeTree()
    {
        //First initialise the tree with a root
        root = new Node(0);
        for (int i = 0; i < 5; i++)
        {
            root.children[i] = new Node(i);
        }
    }

    public void Insert(string strokes, string content)
    {
        /* first create a empty node, set its name/label to
           be the last digit of the string
           * which is the last stroke if the character */
        Node newNode = new Node(Convert.ToInt32(strokes[
            strokes.Length - 1].ToString()));
        newNode.nodeContent = new List<string>(1);
        //content is the characters in the second column of
        the data file
        string[] newContent = content.Split(',');
        for (int i = 0; i < newContent.Length; i++)
        {
            newNode.nodeContent.Add(newContent[i]);
        }
        Node current = root; //start at the root of the tree
        Node parent;
        string s = strokes;
        while (true)
        {
            parent = current; //store the parent of the current
                               node
            if (s.Length > 1)
            {
                //if this child node is empty, create one
                if (current.children[Convert.ToInt64(s[0].
                    ToString()) - 1] == null)
                {
                    current.children[Convert.ToInt32(s[0].ToString
                        ()) - 1] = new Node(Convert.ToInt32(s[0]) -
                            1);
                }
                current = current.children[Convert.ToInt32(s[0].
                    ToString()) - 1];
                int i = 0;
                /* to simplify the search process, a node will
                   store the characters written with its stroke
                   representaion,

```

```

        * together with all characters in its children
        nodes */
    if (current.nodeContent == null)
    {
        current.nodeContent = new List<string>(1);
    }
    //only store at most 6 in one node.
    while(current.nodeContent.Count < 6 && i <
        newNode.nodeContent.Count)
    {
        current.nodeContent.Add(newNode.nodeContent[i
            ]);
        i++;
    }
    s = s.Remove(0, 1);
}
else
{
    parent.children[Convert.ToInt32(s[0].ToString())
        - 1] = newNode;
    return;
}
}
}

public List<string> Search(string objectNumber)
{
    Node start = FindNode(objectNumber);
    if (start == null)
    {
        return null;
    }
    return start.nodeContent;
}

public Node FindNode(string objectNumber)
{
    /* we receive the string of stroke numbers from
        either the interface
        * or somewhere else */
    string s = objectNumber;
    Node current = root; //start at the root
    while (true)
    {
        if (s.Length > 1)
        {
            //go to the node pointed by the first digit of
            the string
            current = current.children[Convert.ToInt32(s[0].
                ToString()) - 1];
        }
    }
}

```

```

        s = s.Remove(0, 1);
    }
    else
    {
        //until the string has only 1 digit
        try
        {
            //return the content of the node
            return current.children[Convert.ToInt32(s[0].
                ToString()) - 1];
        }
        catch (Exception e)
        {
            //just in case if the node is empty
            return null;
        }
    }
}
}
}

public StrokeTree InitialiseDatabase()
{
    StrokeTree tree = new StrokeTree();

    //set path to be the path of the data file
    string path = Directory.GetCurrentDirectory();
    String[] lines = File.ReadAllLines(path + @"\raw.txt");
    ;

    for (int i = 0; i < lines.Length; i++)
    {
        //data[0] is the stroke representation of the node
        //data[1] is the unicode for characters
        string[] data = lines[i].Split(' ');
        tree.Insert(data[0], data[1]);
    }
    return tree;
}
}
}

```

3.3 Testing

ID	Case	Data	Expected Result	Result
1.1	Distance Calibration Initialised	N/A	Distance Calibration window pops up	OK
1.2	Main Window Initialised	N/A	Main Window pops up	OK
1.4	Kinect unit initialised	N/A	The programs can track the user.	OK
2.1	Calibration window should be at the top of all windows	N/A	Calibration window appear at the top of all windows.	OK
2.2	Distance correctly read	N/A	Reasonable value for distance displayed	OK
2.3	Window shut after 3 seconds of stable readings	N/A	Window shut after 3 seconds of consecutive readings.	OK

Note that

both windows receive values from the Kinect.

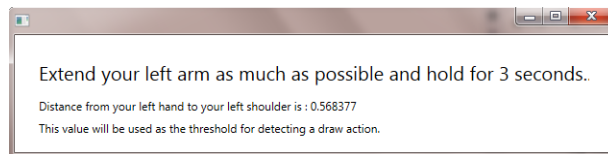


Figure 3.1: Distance Calibration

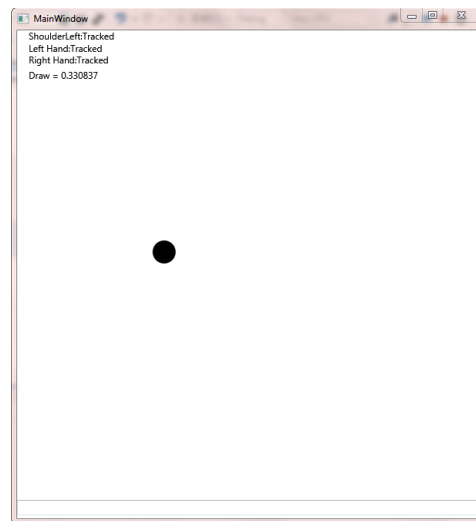


Figure 3.2: Main Window

ID	Case	Data	Expected Result	Result
2.4	Value correctly returned	N/A	Able to draw when user's left hand fully reached out	OK
3.1	Points correctly mapped to the screen	User's movement of right hand	The movement of the pointer is in the same direction as the user's right hand	OK
4.1	Draw gesture recognised	Perform draw(reach out left hand as far as possible)	The pointer shrink into small point and while moving it, it leaves a trace on the screen	OK

The program is able to recognise draw gesture.

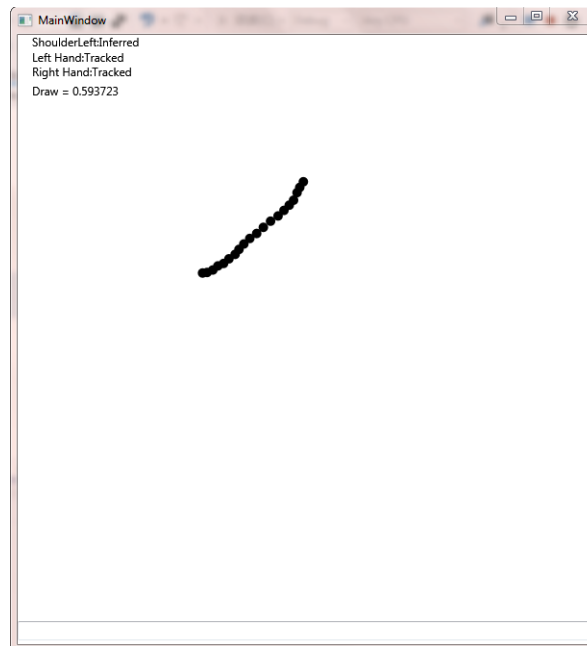


Figure 3.3: Draw dots

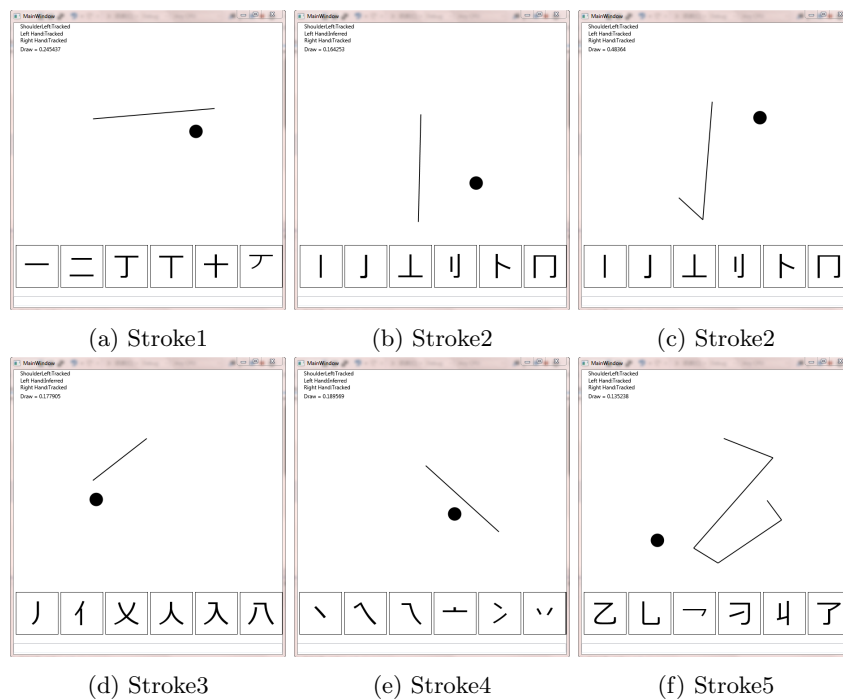


Figure 3.4: Strokes

ID	Case	Data	Expected Result	Result
3.2	Stroke type 1 correctly recognised	Draw a horizontal stroke with a slightly upward slope	The stroke should be recognised as type 1	OK
3.3	Stroke type 2 correctly recognised	Draw a vertical stroke(or combination of 24)	The stroke should be recognised as type 2	OK
3.4	Stroke type 3 correctly recognised	Draw a stroke from up right to down left (or down left to up right)	The stroke should be recognised as type 3	OK
3.5	Stroke type 4 correctly recognised	Draw a stroke from up left to down right (or down right to up left)	The stroke should be recognised as type 4	OK
3.6	Stroke type 5 correctly recognised	Draw multiple strokes in one draw gesture(not combination of 24)	The stroke should be recognised as type 5	OK

ID	Case	Data	Expected Result	Result
1.3	Initialise the Database	N/A	A five to ten seconds halt between the windows appear and start of tracking.	OK
4.4	Choosing a character	Move the point to the desired character and perform a draw gesture	The character should be displayed in the text box and all strokes should be cleared and all the option boxes should be gone	OK

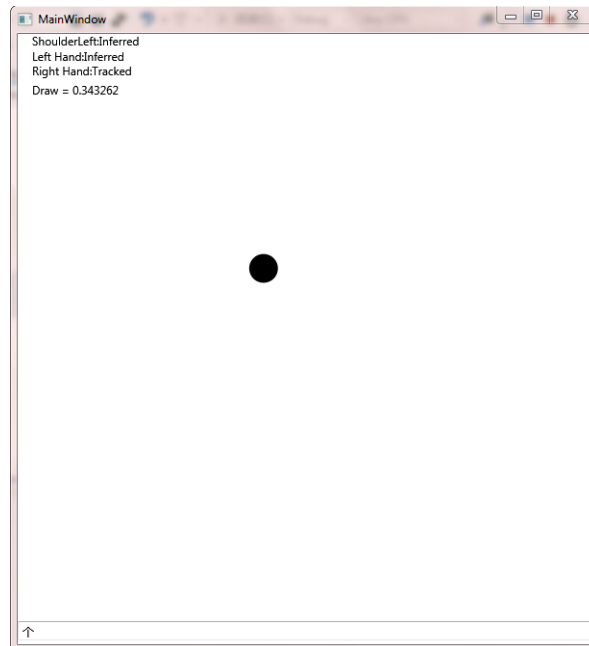


Figure 3.5: Choose a character

ID	Case	Data	Expected Result	Result
5.1	Able to give correct output	strokes	Option boxes appear with characters that matches the strokes	OK
5.2	Able to limit the number of output	strokes	6 option boxes maximum	OK
5.3	Options less than 6	strokes	No empty option boxes appear	OK
5.4	No matched characters	strokes	No option box	OK

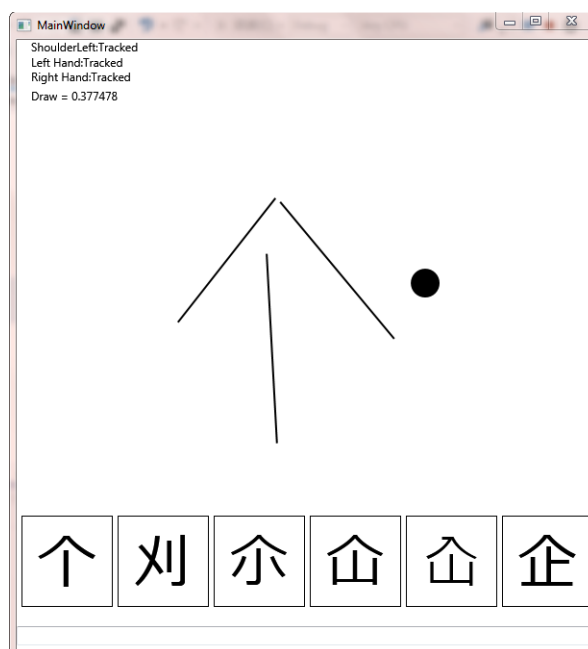


Figure 3.6: Draw character

ID	Case	Data	Expected Result	Result
4.2	Clear screen	Hide both hands	All strokes should be cleared and all the option boxes should be gone	OK
4.3	Delete last stroke	Move the right hand from left to right and back.	Last stroke should be deleted(if any)	OK
4.5	Able to draw after a clear screen	Draw a stroke after clear screen	The stroke should be recognised correctly and all previous strokes should have no sign of existence	OK
4.6	Able to draw after a stroke delete	Draw a stroke after a stroke delete	The stroke should be recognised correctly and the previous stroke should have no sign of existence	OK

3.4 Acceptance Testing

After all the programming is done and testes, the user was given a copy of the software. The aim is to let the user test the program outside of the developing environment. The user was given a list of objectives.

Email 21:

From: Sheng Hu [HU004@doverbroecks.com]

Sent: 10 March 2014 15:05

Hi John,

I feel like I've finished my project and went through all the testing I've designed. I think it is the time to hand it to you for acceptance testing. Shall we meet sometime this week so that I can give you the installation file by USB.

1. Install the program and run "WritingInAir.exe" to start the program.
2. Two windows should pop up. Extend your left arm (as instructed in the window) to set a threshold value for draw.
3. When the calibration is over, you can start drawing.
4. Move your right hand around, the pointer should follow your right hand.
5. Try draw gesture by extending your left arm.
6. Try to draw strokes type 1-5.
7. Try to draw a character and select it by performing a draw gesture when the pointer is over the desired character.

Looking forward to see your reply.

Best regard,

Jack

Email 22:
From: John Peter Thomas [john.peter.thomas@gmail.com]
Sent: 12 March 2014 11:20
To: Sheng Hu
Subject: RE:Requirement Specification

Hi Jack

One of my colleagues installed the program - no problems there. She then followed your instructions to the letter. Calibration was very short - the software says it needs 3 seconds and it was up and running in 3.

She found it slightly confusing at first to work with both hands simultaneously but your use of the left arm to indicate a draw gesture is a good choice.

There is a slight delay with each of the strokes but once shed got used to that she was pretty quick, though the wipe motion comes in pretty handy I have to admit.

She worked through the range of strokes and quite quickly moved on to drawing complete characters.

Your method of character selection at the bottom of the page is excellent and works extremely well.

Its a very neat piece of software - its certainly passed our acceptance tests and I shall email you again in due course once weve had more time to play with the system.

Best wishes

-John

The reply from the client cannot be more positive. So far I can conclude that the alpha testing has eliminated most of the bugs. With this, the beta testing can be seen as finished.

Chapter 4

Evaluation

4.1 Meeting the objectives

4.1.1 User Requirements

User should have full control during the writing process

Requirement met. While writing, the user can decide where and how to draw the stroke; the user can erase the last stroke if wrongly drawn or clear the screen.

4.1.2 Hardware Requirements

- Kinect Sensor
- 32bit(x86) or 64 bit(x64) Intel core processor
- Dual-core 2.66-GHz or faster processor
- Dedicated USB 2.0 bus
- 2 GB RAM
- Graphics card that supports DirectX 9.0c

Requirement met, the program ran successfully on the computer that equipped the hardware listed above.

4.2 Software Requirements

Operating System

Windows 7

Windows 8

The program can run on both Window 7 and Window 8.

Hand Tracking

- able to track the user's hand
- able to correctly recognise user's gestures for drawing and wiping
- able to correctly recognise any strokes drawn by the user
- tracking should be accurate with nearly no delay

Interface

- able to pass data to the database
- able to retrieve data from the database
- able to output the outcomes and allows the user to choose
- able to output the chosen character

Database

- able to receive data from the main program
- able to give a response of with possible outcomes in unicode with no delay

From the feedback from the user, we can see that all the software requirements are met. One thing to point out is that tracking is accurate under assumption of good positioning of the user and the Kinect unit. In terms of reaction speed of the program, the plotting of points is nearly instantaneous.

A few days later, I received user's evaluation on the project.

Email 23:

From: John Peter Thomas [john.peter.thomas@gmail.com]

Sent: 18 March 2014 15:21

To: Sheng Hu

Subject: RE:Requirement Specification

Hi Jack

My colleagues and I have had some time to play with the system and were all extremely impressed. Thanks for the source code too - it was an interesting read. You've clearly got one hell of a career ahead of you as a programmer if maths loses its attraction. The code is highly professional and clearly quite complex in parts. Your design choices were intelligent and well made.

The project itself is very challenging and to my knowledge uniquely successful. I've done some digging around on the net and it seems that attempts to do this sort of thing have been made before but with very limited success and certainly not in the way that you have. Some time ago we had a student of Alan's do a project for us and the last we heard of him he was working in a startup whilst teaching part time at Cambridge from where he'd graduated 2nd in his year. Your work is of the same calibre.

I understand that you're off to read maths at Oxford next year - good luck and keep in touch - I am sure that you have a first class career ahead of you - and thanks for the project, our clients will undoubtedly be impressed and amazed by it.

Best wishes

-John

4.3 Desirable extensions

During the alpha testing, I've noticed the drawback of the design of draw gesture which is tiring. The gesture can be eased by looking for hand gestures rather than body gestures, say use "grip" to indicate draw.

Though the writing process is mostly fluent, it takes quite a long time to finished a character with 10 or more strokes. Mainly there are two ways to ease the process. One will be display more options on the screen which either need to shrink the option box and increase the difficulty on choosing or increase the size of the window. Another way can be recognising characters by its shape which is likely to involve machine learning. By recognising the shape and ink distribution of the character, the user will be able to write the character in one stroke without pauses.