

Online Chinese Character Handwriting Recognition for Linux

Author:	Ran Cheng
Student ID:	n5105269
Course:	IT29 - Honours in IT
Primary Supervisor:	Jim Hogan
Associate Supervisor:	Jinhai Cai

Acknowledgements

I would first of all like to give special thanks to Dr James Michael Hogan and Dr Jin hai Cai for supervising my honours project. Thank you to Jim for introducing me to localisation of software research area, and for all academic and personal help. I would like to thank Jim for providing revision for this thesis. I would like to thank Jin hai for providing technical advice and guiding me through the technical issues of HMMs.

I would also like to thank Red Hat for supporting me financially this past year through the QUT Red Hat Honours Scholarship in Internationalization of Software. More importantly, I would like to thank the team at Red Hat for great support, and for including me as part of their team. Especially, I would like to thank Leon Ho, Caius Chance and Jens Petersen. Your help, encouragement and support have meant a lot to me.

Abstract:

Redhat Linux is one of the leading Linux distributions in the market. Redhat Linux provides user friendly Chinese character input by using a keyboard short cut key combination mechanism, such as the popular “Pin Ying” keyboard input method. As part of the further development of Linux, Redhat plans to add more input methods to the distributions, under Redhat Linux sponsorship. In this project, we focus on investigating a suitable mechanism for Chinese character recognition under the Linux environment, and deliver a software prototype for the purpose of demonstrating the feasibility of our approach.

In this project, we begin by investigating the nature of the Chinese characters, and try to find out the most suitable way to apply a recognition mechanism. By considering both complexity and time constraint, the approaches we take in this project consider detail up to Chinese character stroke level, but use the whole character as the basic recognition unit. In terms of the recognition mechanism, we select the mathematical Hidden Markov Mode as a basis, and customise this model according to Chinese character and handwriting characteristics. During the customisation we made a few assumptions, and justify them as necessary. These assumptions can be easily removed by adding a few additional features to our recognition system. Finally, to make our recognition system work well under Linux OS, we investigated several GUI library and implementation programming languages, and selected GTK+ as the development GUI environment and C/C++ as appropriate implementation languages.

Though we focus on Chinese characters and the Linux OS, the handwriting mechanism developed in this project may in principle be applied to other character sets and is portable to other Linux distributions or other operating systems.

Table of Contents

Acknowledgements	1
Abstract:	2
Chapter One: Introduction	6
1.1 Statement of Research Problem	6
1.2 Research Aim and Objects	7
1.3 Rationale/Background	8
1.4 Significance of Study	9
1.5 Limitations of Study	9
Chapter two: Background and Related work	10
2.1 Hidden Markov Model (HMM)	10
2.1.1 Introduction	10
2.1.2 Element of HMM	10
2.1.3 Three classic problems	11
2.1.4 Type of HMMs	12
2.2 Dynamic Programming (DP)	13
2.3 Viterbi Algorithm	15
2.3.1 Exhaustive search for a solution	15
2.3.2 Reducing complexity using recursion	16
2.3.3 Back tracking	18
2.4 Chinese Character Processing	19
2.4.1 Introduction	19
2.4.2 Character segmentation	20
2.4.3 Pre-processing	20
2.4.4 Pattern Representation	21
2.4.5 Classification	22
2.4.6 Context processing	22
2.5 GTK+	23
Chapter Three: Research Method	23
3.1 Research Approach	24
3.2 Implementation of the Project	26
3.3 Types of Outcomes Anticipated	27
3.4 Reliability/validity of Results	28
3.5 Anticipated Problems and Suggestions for their Solution	28
Chapter Four: Handwriting Recognition System	30
4.1 Writing pad	30
4.2 Data collection	30
4.2.1 Data selection criteria	30
4.3 Data organization	32
4.4 Data format	34
4.4.1 Raw data file	34
4.4.2 Feature data file	35
4.4.3 Distribution probability file	36
4.4.4 Transition probability file	38
4.4.5 Result file	38
4.5 Initial raw data collecting and processing	39
4.6 Feature analysis	40
4.6.1 Character decomposition	40
4.6.2 State decomposition	41
4.7 Training state initialisation	42
4.7.1 Observation segmentation	43
4.7.2 Feature distribution	43

4.7.3	State Transition.....	45
4.8	Training state optimisation.....	47
4.8.1	Customised Viterbi algorithm.....	47
4.8.2	Observation segmentation.....	51
4.8.3	Feature distribution.....	52
4.8.4	State Transition.....	52
4.9	Character recognition.....	52
4.9.1	Feature analysis.....	53
4.9.2	Recognition.....	53
Chapter Four: Experiment and Results		53
4.1	Data Sets.....	53
4.2	Evaluation Criteria	54
4.3	Result Evaluation	54
Chapter Five: Conclusion		56
5.1	Writing Pad XInput support	56
5.2	Relative position handling and Duration handling.....	57
References.....		57
Appendix A: Writing pad.....		59
	Event handling	59
	Drawing Area Widget and Drawing.....	61
	GUI	62
	Writing Pad XInput support	63

Table of Figures

Figure 1 Types of HMM	13
Figure 2 Dynamic programming path.....	14
Figure 3 DP in Chinese Character recognition.....	15
Figure 4 Exhaustive search	16
Figure 5 Forward algorithm.....	18
Figure 6 Viterbi probability formula.....	18
Figure 7 Viterbi path formula.....	19
Figure 8 Three types of Chinese Character.....	20
Figure 9 Research approach.....	25
Figure 10 Chinese character stroke and variation list (Education)	31
Figure 11 Data collection	32
Figure 12 Data framework	34
Figure 13 Raw data text file.....	35
Figure 14 Feature data file	36
Figure 15 Distribution data file.....	37
Figure 16 part of a transition probability file.....	38
Figure 17 Result file	39
Figure 18 Coordinate segmentation	41
Figure 19 Feature distribution in a state.....	42
Figure 20 Observation segmentation	43
Figure 21 Probability formula.....	44
Figure 22 State Transition Architecture	45
Figure 23 Transition Distribution Matrix.....	47
Figure 24 Customised Viterbi algorithm.....	48
Figure 25 Customised Viterbi Algorithm demo	51
Figure 26 Optimised Observation Segmentation	52
Figure 27 Recognition Results.....	55
Figure 28 Basic Chinese Strokes	56

Chapter One: Introduction

Handwriting is one of the input methods to let users interact with a computer. Handwriting recognition has been researched for many years, but it is only in recent years, as new types of pen input devices and interfaces have been developed, some major hardware issues have been resolved, the handwriting started to be used in PCs. As a sub-project of “Internationalisation of Software” which is sponsored by Redhat (a leading Linux company), in this project, we’ll try to develop a new algorithm for Chinese Character handwriting recognition to be used under Linux.

1.1 *Statement of Research Problem*

The first research problem concerns online handwriting recognition. There are three types of handwriting recognition: online, offline and signature recognition. Online recognition means that by using some stylus and touch screen, while user are writing some strokes, the system will try to recognize the character at runtime. Offline recognition means that after users write some sentences by using a pen and a piece of paper, the system will try to recognize all the information on a scanned version of that paper at one time. Signature recognition is thus a special kind of offline recognition.

In this project, a couple of reasons help us to choose online handwriting recognition over the other two. Firstly, online handwriting recognition is the only feasible way to recognize the input character at runtime. Secondly, online handwriting recognition is the most frequently used recognition for operating systems, so research result can be used more effectively in the future.

The second research problem concerns Chinese character processing. Unlike English, the Chinese language has more than 10,000 character categories, and more than 50,000 characters. So to recognize a given Chinese character is much harder than recognizing English. Chinese characters are composed by strokes, and the order or position of the strokes have a significant influence on the recognition. In some cases, two or more characters may look very similar, but they differ substantially in terms of meaning. Handwriting recognition for English has been researched for many years, but it is only

recently that handwriting recognition for Chinese has become popular. Handwriting recognition is a sort of pattern classification: for each character, we need some pattern to match. The English language only consists of 26 different characters. The Chinese language, however, consists of more than 50,000 different characters, even though the most commonly used characters are limited to around 1,000. So the patterns we need to recognize Chinese characters are far more difficult than English characters. There are a couple of convincing reasons for focusing on Chinese characters. Firstly, China has potentially the largest market in the world, so research conducted on Chinese characters may well pay off. Secondly, in recent years our industry sponsor has launched a major program focused on “Internationalisation of Linux”, one of the goals of this program is to develop the Chinese market.

The last but not the least problem I need deal with is implementing the recognition algorithm under Linux. Linux is open source, which means potentially a more secure, more stable operating system, with a faster development cycle and cheaper price (sometimes even free). The current Linux input framework is SCIM (Smart Common Input Method). By the end of this project, we need find out how to implement and develop a software prototype using SCIM API under Linux.

1.2 Research Aim and Objects

The following are the main objectives to achieve and questions to be answered by the end of this project.

- Review the current online handwriting recognition techniques in general and specifically those using Hidden Markov Models
- Review the current online handwriting recognition techniques for Chinese characters
- Review the current SCIM techniques under Linux
- Review the HMM techniques in speech recognition
- Modify some existing handwriting algorithm and Chinese character processing

algorithm to try to create new handwriting algorithm for Chinese character suitable for use under the SCIM framework.

- Test modified algorithm
- Modify some exiting speech recognition algorithm (HMM has been widely used for speech recognition) to let these recognition algorithm work for Chinese character recognition.
- Test modified algorithm
- Compare the results of two modified algorithm and find the better one to implement
- Implement the algorithm under Linux
- Produce an appropriate technical report

1.3 Rationale/Background

In recent years, computers are playing a more and more important role in our daily life. However, some people started learning to use computers at quite late age. It's hard for some senior citizens or disabled people to learn input using a standard keyboard. One good choice for them could be handwriting input, and the approach may be equally convenient for ordinary users of Chinese language versions of the software.

There are many existing handwriting recognition techniques, and they use a variety of different methods to achieve it. Among these techniques, only a few of them use HMMs. Since this project focuses on handwriting recognition for Chinese, we can not use existing HMM handwriting recognition directly. For example, through reviewing the literature, I found one research paper which applied HMM techniques systematically to the speech recognition system. Although speech recognition shares a large similarity with a handwriting system, a significant change is still needed, because speech recognition uses a continuous model instead of a discrete model which is the one we are going to use.

1.4 Significance of Study

It's hoped that when the work for this project (both research and implementation) is completed, the study will contribute to the field of online Chinese character handwriting recognition for Linux in the following ways:

- A software application prototype which can be further extended into fully functional software used by Linux
- A systematic handwriting recognition system implementation approach under Linux. (approach here isn't only for Chinese, but can be used to adopt new languages)
- A new Chinese handwriting recognition algorithm which may be portable to other Operating Systems

We hope after this research project, Linux developers only need a few change or extension to create new Chinese character handwriting recognition software, which will increase the chance of the system occupying larger percentage in the international Operating System market.

1.5 Limitations of Study

The work that will be conducted will be limited in the following ways, which could have an effect on the results and the validity achieved by the objectives that this study will endeavour to achieve.

- Database issue. The basic idea of the HMM is that we develop a new model, and train the system based on an example data set. Then the system will try to recognize the character input by the user based on the template constructed during the training phrase. In other words, the more templates and information stored in the database, the better. For this project, the database has been created from scratch using hand-drawn examples, and needs far more information to be useful in a full-scale study.
- Time constraint. A Chinese character can be decomposed into a group of strokes. If we try to recognize each stroke and then recompose them back to a character,

the accuracy of recognition can be dramatically increased. However, the decompose and recompose procedures may require substantial processing time. So in the project, after balancing the time and accuracy, we decided to recognize a whole character each time.

Chapter two: Background and Related work

2.1 *Hidden Markov Model (HMM)*

2.1.1 Introduction

The Hidden Markov Model (HMM) was initially introduced and studied in the late 1960s and early 1970s, and it is a kind of statistical model of a Markov Source. An HMM is assumed to be a Markov process with unknown parameters, and the challenge is to determine the hidden parameters underlying the process from the observable parameters. The extracted model parameters can be used to perform further analysis, and a common example is the pattern recognition application which is the category our project belongs to. The HMM differs from a pure Markov model, because in a Markov model, the state is directly visible to the observer; in the HMM, the state is hidden but the state can influence some variables which are visible. Each state has a probability distribution over the possible output observations, so the sequence of observations generated by an HMM gives some information about the sequence of states.(Wikipedia, 2006b) In our project, for the handwriting analysis, the strokes can be transformed into a sequence of dots, which can be treated as the observation sequence, and the final shape of the character can be treated as the state. Therefore, by observing the sequence of dots input by user, the system will guess the correct character. This is one of the main reasons we choose HMM, and the other two strong reasons the HMM draw our attention are: firstly, the models are very rich in mathematical structure and hence can form the theoretical basis for use in a wide range of applications; secondly, when the models are applied properly, it works well in practice for several important applications. (Rabiner, 1989)

2.1.2 Element of HMM

An HMM is characterized by the following:

- 1) N , the number of states in the model. As we mentioned in the introduction, the

state is hidden in the models, but for many practical applications, such as ours, the number of the state are known in advance.

- 2) M , the number of distinct observation symbols per state.
- 3) The state transition probability distribution $A = \{a_{ij}\}$, where each value is the probability of changing from state S_i to S_j .
- 4) The observation symbol probability distribution in state j , $B = \{b_j(k)\}$
- 5) The initial state distribution π .

Given appropriate values of N , M , A , B and π , the HMM can be used as a generator to give an observation sequence.

The HMM is very general mathematical model, and it can be used in a lot of areas by selecting proper elements and using it in different way. In the other word, not all the elements will be used for every scenario. Generally, when some elements are not used, we sign a default value to them.

2.1.3 Three classic problems

Rabner (1989) outlined three key problems of hidden markov models:

Problem 1: Given the observation sequence and a model, how can we efficiently compute the probability of the observation sequence, given the model?

Problem 2: Give the observation sequence and the model, how can we choose a corresponding state sequence which is optimal in some meaningful sense?

Problem 3: How can we adjust the model parameters to maximize the probability of the observation sequence?

The current work does not attempt to improve existing solutions to these general problems. Rather, we are concerned with applying these techniques to our application.

Problem 1 is the evaluation problem. This problem can be treated as one of scoring how well a given model matches a given observation sequence. In other words, problem 1 allows us to choose the model which best matches the observations.

Problem 2 is trying to uncover the hidden part of the model – the underlying state sequence. One thing we should pay attention is that we can't really find the "correct" state sequence. All we can do is to find out the best or optimal state sequence. Normally, there are several reasonable optimality criteria that can be imposed, and hence the choice of criterion is a strong function of the intended use for the uncovered state sequence. In this project, we'll be using the theory listed in this problem: by observing the observation sequence, the system will try to find out all the possible states (characters) and list the top 10 or whatever number we require.

Problem 3 can be viewed as a "training" problem. The observation sequence used to adjust the model parameters is called a training sequence, since it is used to "train" the HMM. The selection criteria for the training set should be defined according to the project need. Generally speaking, the training set should have the ability to represent a large number of possible cases, and should have some features which are shared by a large fraction of the unseen data. After training, the system should be able to easily memorise the key features of the training data and apply the learned relationship to unseen examples.

2.1.4 Type of HMMs

There are two main types of HMMs and there are many possible variations and combinations possible. The first main type of HMM is called the ergodic model, in which every state can be reached from every other state in a finite number of steps. This kind of HMM is most commonly used in the practice. The second main type of HMM is called a left-right model or a Bakis model. In this kind of model, the state sequence has the property that as time increases the state index increases. We use the following two pictures to show the underlying structure of these two models.

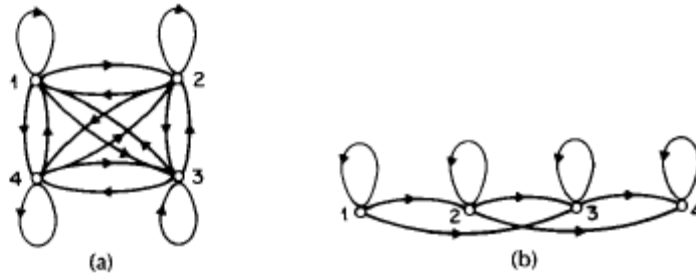


Figure 1 Types of HMM

We can see from picture (a) that all the states are connected with each other, and the arrows indicate the communication directions enabled. In picture (b), at any time, the current state can only communicate with the following states and itself, but no previous state can be reached.

The ergodic model is clearly more powerful than the left-right model, since every state can be reached from every other state in a finite number of steps. But in real-world applications, the left-right model is frequently more useful because it is easier and most of the applications do not need change the state back to previous one.

The left-right model is particular useful for our project, because this kind of model has “the desirable property that it can readily model signals whose properties change over time”(Rabiner, 1989), like speech and handwriting.

2.2 Dynamic Programming (DP)

“In computer science, dynamic programming is a method for reducing the runtime of algorithms exhibiting the properties of overlapping sub-problems and an optimal substructure.”(Wikipedia, 2006)

Optimal substructure means that optimal solutions of subproblems can be used to find the optimal solution of the overall problem. We use the picture shown left to explain the definition. Starting from the leftmost circle, we got three paths to reach the rightmost circle. The number between near the line means the distances. By observing the picture,

we can see at the middle of the overall path, the second path from the top has the shortest distance, however, in terms of the total distance, the top path is the shortest. Although the second path was leading the way at some point, the optimal solution is the top path.

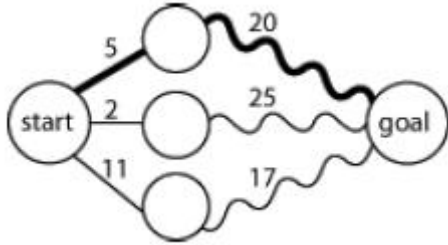


Figure 2 Dynamic programming path

Normally, the three-step process we should take is:

Firstly, break the problem into smaller subproblems.

Secondly, solve these problems optimally using this three-step process recursively.

Thirdly, use these optimal solutions to construct an optimal solution for the original problem. (Wikipedia, 2006)

Overlapping subproblems mean that the same subproblems are used to solve many different larger problems. We take Fibonacci sequence as example, $F_3 = F_1 + F_2$ and $F_4 = F_3 + F_2$, the consequence calculation also include F_2 . The calculation of F_2 is involved in each of the following calculations. So we say F_2 is an overlapping subproblem in calculation of the Fibonacci sequence.

The overlapping subproblem introduces a new problem. In each calculation of the Fibonacci sequence, the calculation may end up computing F_2 twice or more. In this approach, we may waste time recomputing optimal solutions to subproblems the system has already solved. To avoid this, the memorization approach was introduced. Memorization means that if the same problem may be need to re-solved later, the system will save the solutions to the problem and retrieve and reuse the solutions.

DP usually takes one of two approaches:

Top-down approach: break the problem into subproblems, and after resolving these subproblems, recombine all the subproblems back together.

Bottom-up approach: resolve all the subproblems in advance and then use them to build up solutions to larger problems.

DP can be very useful in character recognition processing. The picture shown below simulates DP in Chinese Character recognition. As the new strokes are added, the possible character selections change over time, and eventually, the system will optimize all the possible solutions and find the one most probable.

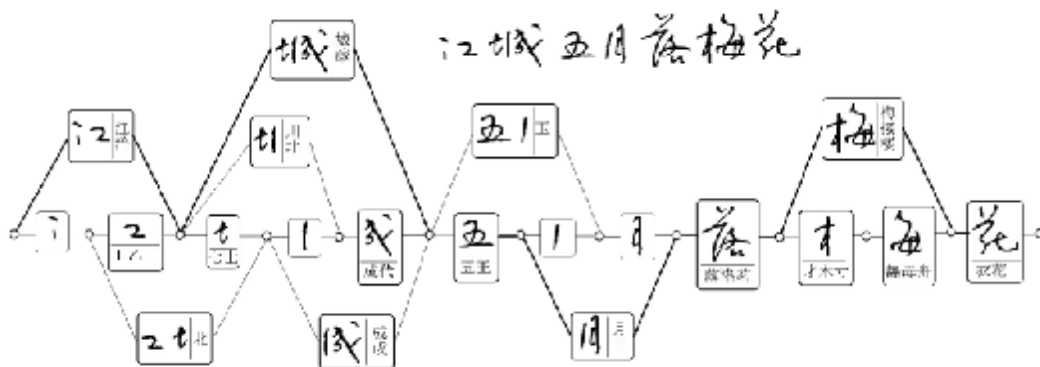


Figure 3 DP in Chinese Character recognition

2.3 Viterbi Algorithm

We often wish to take a particular HMM, and determine from an observation sequence the most likely sequence of underlying hidden states that might have generated it.

2.3.1 Exhaustive search for a solution

One straightforward solution to this problem is to perform an exhaustive search for a solution. We can use a picture of the execution trellis to visualise the relationship between states and observations.

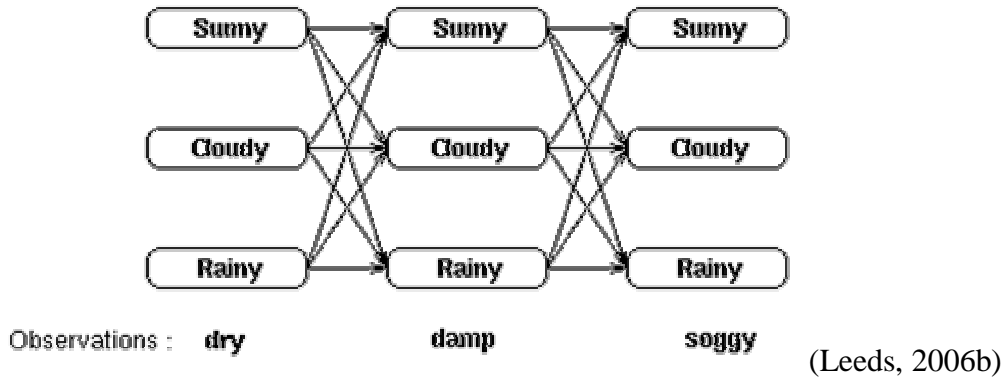


Figure 4 Exhaustive search

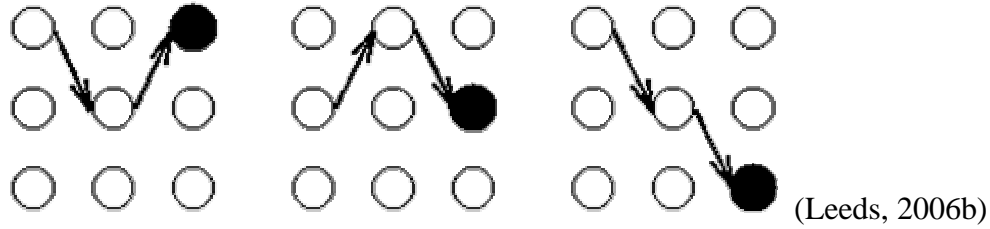
We can find the most probable sequence of hidden states by listing all possible sequences of hidden states and finding the probability of the observed sequence for each of the combinations. The most probable sequence of hidden states is that combination that maximises the probability. This approach may be viable, but to find the most probable sequence by exhaustively calculating each combination is computationally expensive. As with the forward algorithm, we can use the time invariance of the probabilities to reduce the complexity of the calculation.

2.3.2 Reducing complexity using recursion

We will consider recursively finding the most probable sequence of hidden states given an observation sequence and a HMM. We will first define the partial probability δ_t , which is the probability of reaching a particular intermediate state in the trellis. We then show how these partial probabilities are calculated at $t=1$ and at $t=n (> 1)$. These partial probabilities differ from those calculated in the forward algorithm since they represent the probability of the most probable path to a state at time t , and not a complete traversal of the trellis.

2.3.2.1 Partial probabilities (δ_t 's) and partial best paths

Consider the trellis we used in the “exhaustive search for solution” showing the states and first order transitions for the observation sequence: dry, damp, soggy;



We will call these paths partial best paths. Each of these partial best paths has an associated probability, the partial probability or δ . Unlike the partial probabilities in the forward algorithm, δ is the probability of the one (most probable) path to the state. Thus $\delta(i,t)$ is the maximum probability of all sequences ending at state i at time t , and the partial best path is the sequence which achieves this maximal probability. Such a probability (and partial path) exists for each possible value of i and t . In particular, each state at time $t = T$ will have a partial probability and a partial best path. We find the overall best path by choosing the state with the maximum partial probability and choosing its partial best path.

2.3.2.2 Calculating δ 's at time $t = 1$

We calculate the δ partial probabilities as the most probable route to our current position (given particular knowledge such as observation and probabilities of the previous state). When $t = 1$ the most probable path to a state does not sensibly exist; however we use the probability of being in that state given $t = 1$ and the observable state k_1 ; i.e.

$$\delta_1(i) = \pi(i)b_{ik_1}$$

As in the forward algorithm, this quantity is compounded by the appropriate observation probability.

2.3.2.3 Calculating δ 's at time $t (> 1)$

We now show that the partial probabilities δ at time t can be calculated in terms of the δ 's at time $t-1$.

Consider the trellis below :

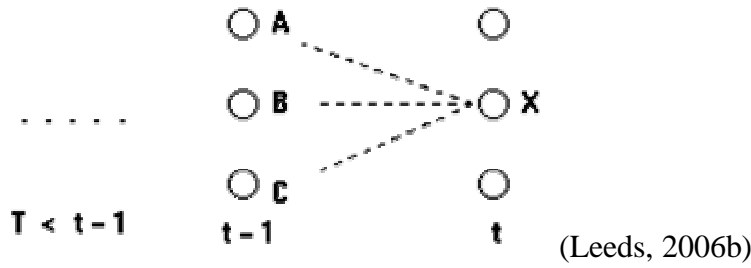


Figure 5 Forward algorithm

We consider calculating the most probable path to the state X at time t; this path to X will have to pass through one of the states A, B or C at time (t-1). Therefore the most probable path to X will be one of

- (sequence of states), . . . , A, X;
- (sequence of states), . . . , B, X;
- Or (sequence of states), . . . , C, X

We want to find the path ending AX, BX or CX which has the maximum probability.

Following this, the most probable path ending AX will be the most probable path to A followed by X. Similarly, the probability of this path will be

$$\Pr(\text{most probable path to A}) \cdot \Pr(X | A) \cdot \Pr(\text{observation} | X)$$

So, the probability of the most probable path to X is :

$$\begin{aligned} \Pr(X \text{ at time } t) = \\ \max_{i=A,B,C} \Pr(i \text{ at time } (t-1)) \times \\ \Pr(X|i) \times \Pr(\text{obs. at time } t|X) \end{aligned} \quad (\text{Leeds, 2006b})$$

Figure 6 Viterbi probability formula

where the first term is given by δ at t-1, the second by the transition probabilities and the third by the observation probabilities.

2.3.3 Back tracking

Consider the trellis we used in the “exhaustive search for solution” again, At each intermediate and end state we know the partial probability, $\delta(i,t)$. However the aim is to

find the most probable sequence of states through the trellis given an observation sequence - therefore we need some way of remembering the partial best paths through the trellis.

Recall that to calculate the partial probability, δ_t at time t we only need the δ 's for time $t-1$. Having calculated this partial probability, it is thus possible to record which preceding state was the one to generate $\delta_t(i)$ - that is, in what state the system must have been at time $t-1$ if it is to arrive optimally at state i at time t . This recording (remembering) is done by holding for each state a back pointer ϕ which points to the predecessor that optimally provokes the current state.

Formally, we can write

$$\phi_t(i) = \underset{j}{\operatorname{argmax}} (\delta_{t-1}(j) a_{ji}) \text{ (Leeds, 2006b)}$$

Figure 7 Viterbi path formula

Here, the argmax operator selects the index j which maximises the bracketed expression.

Notice that this expression is calculated from the δ 's of the preceding time step and the transition probabilities, and does not include the observation probability (unlike the calculation of the δ 's themselves). This is because we want these ϕ 's to answer the question 'If I am here, by what route is it most likely I arrived?' - this question relates to the hidden states, and therefore confusing factors due to the observations can be overlooked. (Leeds, 2006a)

2.4 Chinese Character Processing

2.4.1 Introduction

More than one quarter of world's population use Chinese characters in daily communications. There are three main types of Chinese characters: simplified Chinese characters, which are used in mainland China and Singapore; traditional Chinese characters, which are used in Taiwan, Hong Kong and Macao; and Japanese Kanji, which

are used in Japan. In both traditional and simplified Chinese, about 5,000 characters are frequently used, In Kanji, 2,965 Kanji characters are included in the JIS level 1 and 3,390 characters in level 2.(C. L. Liu, Jaeger, & Nakagawa, 2004) Although there are a lot of styles of Chinese Characters, normally, the most common used three styles are regular script, fluent script and cursive script. The picture on the right-hand side shows the difference between these styles.

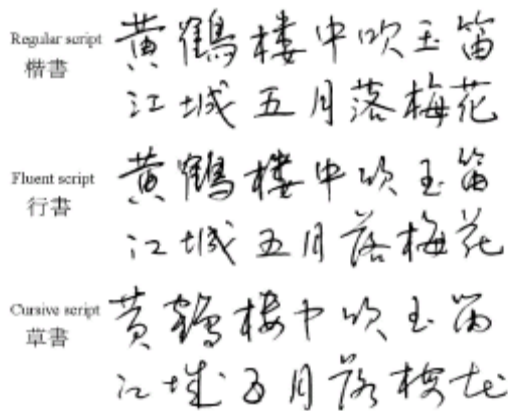


Figure 8 Three types of Chinese Character

2.4.2 Character segmentation

When the handwriting sequence is input into the system, no matter what kind of types of Chinese characters or styles are used, the sequence should be segmented into character patterns according to the temporal and shape information. Frequently, the boundary between characters can't be determined unambiguously before character recognition, so a set of candidate character patterns are selected in contextual processing at the end of the process chain. After segmentation, a set of individual Chinese character are output to the next stage. Although a lot of research work has been done independently, and a number of recognition approaches are available, almost all researchers follow the same approach for the segmentation step.

2.4.3 Pre-processing

Pre-processing consists of noise elimination, data reduction, and shape normalizations. The commonly used noise reduction techniques are smoothing, filtering, wild point correction, and stroke connection (Tappert, Suen, & Wakahara, 1990). As the quality of

input devices is getting better, trajectory noise becomes less influential and simple smoothing operations will suffice.(C. L. Liu et al., 2004)

Two frequently used approaches for data reduction are equidistance sampling and line approximation. With equidistance sampling, the trajectory points are resampled such that the distance between adjacent points is approximately equal. Line approximation, also called feature point detection, is often used in many online recognition systems.

Normalization of character trajectories to a standard size is adopted in almost every recognition system, and our project will follow suit. Three main approaches for normalization are linear, moment and nonlinear normalization. Linear normalization means that the coordinates of stroke points are shifted and scaled such that all points are enclosed in a standard box. Moment normalization means that the centroid of input pattern is shifted to the centre of standard box and the second-order moments are scaled to a standard value (R.G.Casey, 1970). Nonlinear normalization means that the coordinates of stroke points are adjusted according to the line density distribution with the aim of equalizing the stroke spacing.(S.-W., Lee, & Park, 1994)

2.4.4 Pattern Representation

The representation schemes of input pattern and model database are of particular importance since the classification method depends largely on them. The representation schema can be divided into three groups: statistical, structural, and hybrid statistical structural. In statistical representation, the input pattern is described by a feature vector, while the model database contains the classification parameters. The statistical-structural scheme is only used for describing the reference models. It takes the same structure as the traditional structural representation, yet the structure elements and/or relationships are measured probabilistically. HMMs can be regarded as instances of the statistical-structure representation.

Pattern Representation and the following section – Classification will be the main area we are going to focus on in our project. We'll be trying to apply HMM methods to develop a new algorithm.

2.4.5 Classification

Classification is the core of almost every recognition system. Classification can be further divided into coarse classification and fine classification. Coarse classification can be accomplished by class set partitioning or dynamic candidate selection. In class set partitioning, the groups of classes are determined in the classifier design stage using clustering or prior knowledge. Class grouping can be based on overall character(C. K. Lin & Fan, 1994), basic stroke substructure(R. H. Chen, Lee, & Chen, 1994), stroke sequence(Z. Chen, Lee, & Cheng, 1996), and statistical or neural classification (Matic, Platt, & Wang, 2002). In dynamic candidate selection, a matching score is computed between the input pattern and each class and a subset of classes with high scores is selected for more detailed classification.

In fine classification, three common used approaches are Structural Matching, Probabilistic Matching and Statistical Classification. Structural matching means the input pattern is matched with the structural model of each (candidate) class and the class with the minimum matching distance is taken as the recognition result. Probabilistic matching means that using probabilistic attributes in representing structural models and computing matching distance. Statistical classification means that using various statistical techniques for classification when describing the input pattern as a feature vector.(C. L. Liu et al., 2004)

2.4.6 Context processing

The linguistic context can provide valuable information for selecting the optimal class from this set of candidates. In addition, the geometric features of character patterns are useful to segment a handwriting sequence into single characters. Based on the candidate classes given by the character recognizer, additional candidates can be added according to the statistics of confusion between characters in order to reduce the risk of excluding the true class. The selection of final class from candidate class is based on the linguistic knowledge represented in word dictionaries, character-based n-grams, or word-based n-grams.(M.-Y. Lin & W.-H.Tsai, 1988) .Some locale-specific dictionary can be used to help to determine the optimal solutions. The ambiguities in segmentation are generally

solved by generating candidate character patterns and verifying the candidate patterns using geometric features, recognition results, and linguistic knowledge.

2.5 GTK+

GTK+ is a multi-platform toolkit for creating graphical user interfaces. Offering a complete set of widgets, GTK+ is suitable for projects ranging from small one-off projects to complete application suites.

GTK+ is free software and part of the GNU Project. However, the licensing terms for GTK+, the GNU LGPL, allow it to be used by all developers, including those developing proprietary software, without any license fees or royalties.

GTK+ is based on three libraries developed by the GTK+ team:

GLib is the low-level core library that forms the basis of GTK+ and GNOME. It provides data structure handling for C, portability wrappers, and interfaces for such runtime functionality as an event loop, threads, dynamic loading, and an object system.

Pango is a library for layout and rendering of text, with an emphasis on internationalization. It forms the core of text and font handling for GTK+-2.0.

The ATK library provides a set of interfaces for accessibility. By supporting the ATK interfaces, an application or toolkit can be used with such tools as screen readers, magnifiers, and alternative input devices.

GTK+ has been designed from the ground up to support a range of languages, not only C/C++. Using GTK+ from languages such as Perl and Python (especially in combination with the Glade GUI builder) provides an effective method of rapid application development. (team, 2006)

Basically, GTK+ is like Swing in Java, just more complicated, since GTK+ can be used to develop the GUI for entire Operating System. GTK+ provides the overall software framework for hosting our project, but further explanation is difficult without examining the code directly.

Chapter Three: Research Method

The purpose of this section is to present and describe details about the research and the

course of action that undertaken to complete the project. .

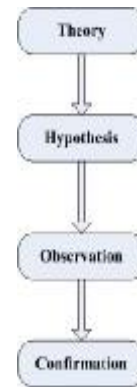
3.1 Research Approach

A number of research methodologies are available. The project used the scientific methodology. The “scientific method is a body of techniques for investigating phenomena and acquiring new knowledge, as well as for correcting and integrating previous knowledge. It is based on gathering observable, empirical, measurable evidence, subject to the principles of reasoning.”(Newton, 1999) Normally, the scientific method consists of the following components:

- “Observe: collect evidence and make measurements relating to the phenomenon you intend to study
- Hypothesize: invent a hypothesis explaining the phenomenon that you have observed
- Predict: use the hypothesis to predict the results of new observations or measurements
- Often advanced mathematical and statistical hypothesis testing techniques are used to design experiments that attempt to effectively test the plausibility of hypotheses
- Verify: perform experiments to test those predictions.
- Attempting to experimentally falsify hypotheses is thought to be a better choice of term here
- Evaluate: if the experiment contradicts your hypothesis, reject it and form another. If the results are compatible with predictions, make more predictions and test it further.
- Publish: Tell other people of your ideas and results, and encourage them to verify the claims themselves, in particular by inviting them to challenge your reasoning and check that your experimental results can be repeated. This process is known as ‘peer review’. “(Gable, 2006)



The scientific method is a general concept, which can be used as research method in different areas rather than in computer science only. In terms of scientific method, there are two possible ways the researchers can take - Deductive Reasoning and Inductive Reasoning. “Deductive reasoning is generally used to predict the results of the hypothesis. That is, in order to predict what measurements one might find if you conduct an experiment, treat the hypothesis as a premise, and reason



deductively from that to some not currently obvious conclusion, then test for that conclusion.”(Gable, 2006) This approach starts from the more general and then narrows the topic and makes it more specific. Sometimes it’s called a ‘top down’ approach. The right-hand side picture shows the procedures of Deductive Reasoning. Inductive reasoning works the reverse way; it starts from the more specific or the bottom to the more general or up. The left-hand side picture shows the procedures of Inductive Reasoning. These two ways of scientific reasoning form the cyclical Nature of Research, which is shown below.

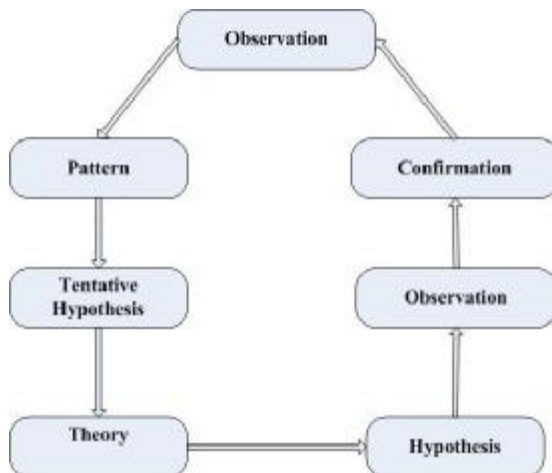


Figure 9 Research approach

We choose this research methodology because of these following reasons:

- Clear observations and predictions
 - Successful speech recognition based on HMM has a long history and the speech recognition shares large similarity with handwriting recognition. So I predict HMM model can be used for handwriting recognition.

- A lot of successful research has been conducted with regards to handwriting recognition and the basic idea of HMM is that it treats any strokes as a sequence of dots, allowing a general approach to character recognition.
- Fortunately, I found one Japanese character handwriting recognition software which has been successfully adopted by SCIM under Linux. So I predict SCIM will be able to adopt Chinese character recognition.
- More suitable than others

Before I made decision, I investigated some other research methodologies, such as experiments, case study, and so on. In terms of this project, our sponsor expects some useful software prototype rather than a very good algorithm. In the other words, this project focuses more on the practical work. So the research methods, like case study method, which focus on investigation, will not be suitable.

3.2 Implementation of the Project

The primary goal of this project is to develop a Chinese character handwriting recognition algorithm and to create a software prototype under Linux. Since this project involved Linux programming, C++ or C is essential implementation language. While I have a lot of programming experience with Java and .Net languages, in terms of C++ or C, I'm a relative novice. The software prototype will involve a lot of Linux GUI programming, GTK+ is the GUI language library under Linux OS. To develop the software prototype, I need learn GTK+ as well. Redhat Linux is developing very fast. Every six month, new distribution of Linux is released. For this project, I'll be using Fedora Core 5 as the Operating System; the entire library used in this project will be compiled under FC5. To use the software applications developed in this project in future Linux distribution, user can recompile from the source code. During this research project, the following tasks need to be done.

- Learn and familiarize myself with C++
- Learn and familiarize myself with GTK+
- Learn HMM and Dynamic Programming
- Decide the particular HMM type and model to use
- Implement three classic HMM problems using C++
- Compare HMM technique with some other existing handwriting techniques to find out if I can adopt some approaches and use them to help apply HMM to handwriting

- Analyse both handwriting and speech techniques (a lot of speech algorithm use HMM), try to find out the common part to reuse.
- Investigate the Chinese character processing techniques, including segmentation, pre-processing, pattern representation, classification and context processing.
- Investigate the existing Linux Japanese handwriting recognition software (Tomoe), and find out the parts we can reuse
- Create simple testing Chinese character database
- Investigate the SCIM API
- Create writing pad using GTK+ and C++
- Deal with feature analysis of input character
- Deal with unit matching system (apply HMM and DP)
- Deal with Lexical decoding (add in word dictionary)
- Deal with syntactic analysis (add in grammar checking)
- Deal with semantic analysis (add in task model)

Some of tasks are at a low level or programming level. It is not easy to explain further details without showing the actual source code.

3.3 *Types of Outcomes Anticipated*

It is hoped that upon completion of this research project that the work undertaken will help improve the understanding of handwriting technique under Linux for not only the researcher but also field in general. Especially as SCIM is new, and there will be a lot of research work can be done in the future. A good research outcome in this project can give useful suggestions to future researchers.

The following is a list of the major deliverables for this research project that have been decided upon and that the researcher will aim to produce by the completion of the work.

- A software prototype which show how well the SCIM and adopt Chinese character handwriting technique and how it works under Linux
- An enhanced writing pad which can be reused to input any language character in the future
- An online Chinese character handwriting recognition algorithm which can be portable to other OS
- A detailed instruction and approach of handwriting implementation under Linux which can be

used as a guide for any other language handwriting recognition under Linux in the future

3.4 *Reliability/validity of Results*

To verify that the results obtained are correct rigorous testing is needed to ensure that any outcomes achieved can be reproduced. Since this project lies at the beginning of handwriting recognition for Linux, the primary issue is not how well the handwriting system works under Linux but whether the handwriting system works at all. So the validation of the results is straight forward. As long as the system can reliably recognize the characters, the job is done. As mentioned in the previous section, two possible hypotheses are considered in this project. One is to incorporate HMM into an existing handwriting technique, the other is to modify an existing approach from speech recognition which is already using HMM.

Scientific method is used as the approach to this research, which tests an assumption or hypothesis that has been made through the entire project, testing is naturally part of the process. Experiments will be conducted to test the two hypotheses. Experiments may be repeated until expected results come out or the predictions are approved. This will hopefully ensure that the results reported are valid and reliable and can be of benefit to the field of handwriting recognition under Linux.

3.5 *Anticipated Problems and Suggestions for their Solution*

As at this stage in the project we have resolved most problems we had with the process or desired outcomes, although additional problems arising from extensions to the specification cannot be discounted. There are several aspects of the project that may prove to cause problems later on and are detailed here.


Firstly, the stroke input by user may be too short. To improve the performance of the recognition system, when we capture the character data from the writing pad, we set the distance between dots to a particular number¹. If the stroke is too short, the system will

¹ The number equals 5 in our project.

not be able to capture enough information for that stroke. To fix this problem, we can take either of the following two approaches:

- Rewrite the stroke, and make sure the stroke is long enough
- Reset the distance between dots to a smaller number. For a good recognition accuracy, we recommend that the number be always larger than 3.

Secondly, the transition number between states may be too small. This is another problem caused by a short stroke. In this case, while the system is able to capture enough information for the stroke, the number of useful features in each stroke may be relatively small.

Thirdly, the number of states may be too small. We use a number of states to represent a stroke in our project. A complicated stroke, may require more states than the simple stroke does. For example, for these two strokes “—” and “”, we can use three states to represent the first stroke, but for the second one, we have to at least use five states.

Most of the problems are caused by our small testing database. The testing database used during the project was created by the author. Relatively few characters are stored in this database. Although we resolved many of the problems encountered, it is unclear how performance will be affected as the database gets larger. In a small database, when the recognition system tries to recognize the character, it's not too hard for the system to find the correct one. However, as the database gets larger, there could be many characters which share significant shape similarity, and the performance of the recognition system may fall away. This problem can be resolved in an extension of this project, and some suggestions will be given in the future work section.

Chapter Four: Handwriting Recognition System

4.1 *Writing pad*

To recognize the Chinese character, we need some mechanism in place to take users' input. An electronic writing pad is the most commonly used mechanism to make the users' input available. Under Linux, GTK+ is handy a multi-platform toolkit for creating graphical user interfaces. Besides creating a handwriting pad, GTK+ can be extended to provide support for XInput devices, such as drawing tablets. Furthermore, GTK+ provides support routines which make getting extended information, such as pressure and tilt, from such devices relatively easy. In this project, we'll be using GTK+ to create the writing pad only; further support can be added in future work. To develop the writing pad, there are three things we need to cope with: Event handling, the Drawing Area Widget and Drawing itself. Please refer to Appendix A "Writing pad" for more information.

4.2 *Data collection*

In this project, we selected a set of characters to be used for training and recognition. There is huge number of Chinese characters, and we can't try to recognize them all. So what we are trying to do is selecting a set of characters which can represent the huge number of Chinese characters. In terms of the data selection criteria, please refer to the next section.

The data set was created as follows. For each character in the data collection, we repeatedly write ten times in the writing pad and store them. Eight times of the characters are used for training and two times of them are reserved for recognition.

4.2.1 *Data selection criteria*

The objective of this project is to develop a mechanism for Chinese character recognition, and deliver the software prototype. We do not expect this software prototype to compete with the existing commercial software, but we do expect that this prototype can demonstrate the feasibility of the recognition mechanism and can be expanded and used as the foundation for a real world recognition system. To prove the scalability of the prototype, we will try to make good use of the limited resource by selecting a small number of Chinese characters which can represent variations in the most commonly used

What we do in this project is that for each stroke, we select a set of characters which contains that stroke, then we pick up the one character which is more frequently used than the others. Finally, we arrived at the following data set:

口	十	
又	士	
买	戈	
山	冰	
四	把	
长	川	
公	去	凸
女	五	及
犹	可	寄
代	千	阳
凹	人	马
朵	月	乃
计	主	
同	州	
飞	这	
鼎	义	
专	之	
己	心	

Figure 11 Data collection

4.3 Data organization

After getting the data collection, we need work out how to organize the data. In this project, we will create a data framework according to the role the data plays. At the top of the data framework, all the data will be categorised into two types – training data and recognition data. We need to allocate some of the data in the data collection as the training data, and the rest as the recognition data. As mentioned before, for each Chinese character, we generate examples by letting the user write the character forty times in the

writing pad and storing the results in the central repository.

In terms of the training data, it can be further categorised into three types – Raw data, Initial Data and Optimised Data. Raw data is the data we get directly from the writing pad without any processing; after pre-processing or initialising the raw data, we get the initial data; after optimising the initial data, we end up with the optimised data. For further details about these three types of data and processing, please refer to the following section. Under the Raw Data directory, we classify all the raw data by character, and store all the raw data file which represent the same Chinese character within one directory. Following the same pattern, all the feature data (initialised data) is categorised by character as well. There are two text file are associated with each character. One of the text files contains the distribution probability of features for each character. The other text file contains the transition probability of the states associated with each character. With regards to the distribution and transition probabilities, please refer to the following sections. The “Optimised Data” folder uses exactly the same structure as the “Initial Data” folder. It contains a sequence of folders categorised by character and text files associated with each character. However for the character folders under “Initial Data”, the text files contain the feature data, whereas in the character folders under “Optimised Data”, the text files contain the Viterbi sequence data.

The Recognition Data files are structured similarly.

The picture below shows the data framework.

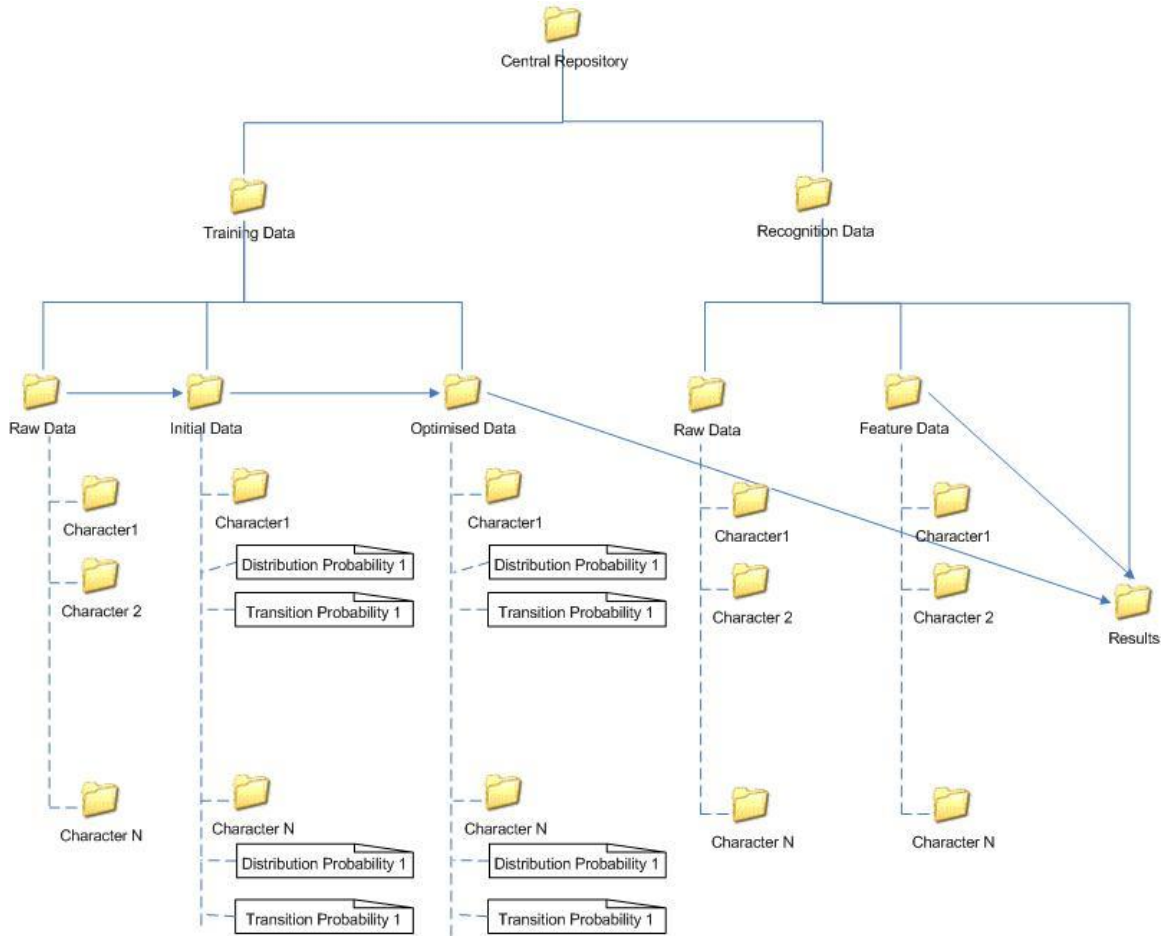


Figure 12 Data framework

4.4 Data format

So far we have discussed the overall data framework structure. Now we get into more details about the data files. There are two options for us to store the data files, text and binary. In terms of the efficiency, binary should be a better choice. In this project, however, we focus more on the feasibility and the accuracy of the mathematic model and algorithm, furthermore, for easy diagnostic purpose, we used text format for all the data files. To improve the efficiency of the recognition system, binary format can be used in the future work.

4.4.1 Raw data file

We follow the order we discuss the folders in previous section. We start with “Raw data” folder in the “Training data” folder. In each Character folder, there are a number of files

which represent the same character. Those files contain the raw data which means the data directly retrieved from the writing pad without any processing. From the system point of view, the characters which are drawn by the users are just a sequence of dots in the coordinate and nothing more than that. So when we store those dots for a character, we should see a sequence of X, Y coordinates. The separator should be used between X, Y coordinates, and we use “,” in the raw data files. While retrieving the sequence of the dots which represent the characters, we need take some action to clearly identify the border between strokes. We use tag mechanism to do so. All the dots which belong to the same stroke will be stored between tag “<s>” and “</s>”. A screen shot of two-stroke character raw data file can be found below:

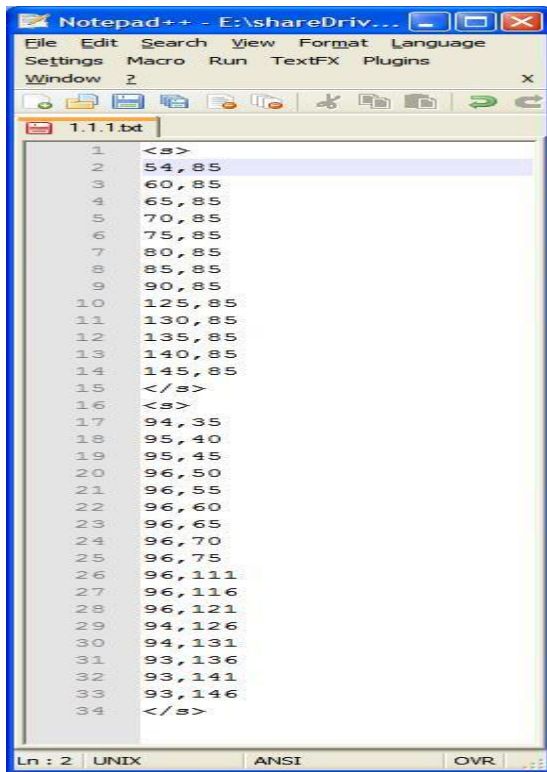


Figure 13 Raw data text file

4.4.2 Feature data file

In essence, feature data files are the raw data files after the feature analysis process. There will be one to one mapping between raw data files and feature files. Furthermore, during the feature analysis process, information loss should never occur, so the feature data file contains the exactly same number of rows as the raw data file. By comparing the two

screen shots (one for raw data from previous section and one for feature data below), we can see there 34 rows in raw data file, and 4 of them are tags. So the corresponding feature file contains 30 rows. It should be noted that the feature doesn't employ the tag mechanism, since the recognition system uses a different method to handle these types of files. To segment the strokes in the character in the feature file, we use an "add/minus" mechanism. We already know that the numbers appearing in the feature data files are always between "0" and "15", so we need to find some particular number, and after adding that particular number to the original value, the original value can be significant change and is out of the normal range. The particular number we pick up is "16". For the first value of each stroke, we add number "16" to it, and for the last value of each stroke, we minus "16" from it. Every time the system meets a number larger than 15 or less than 0, it will notice that is the beginning or end of a stroke.

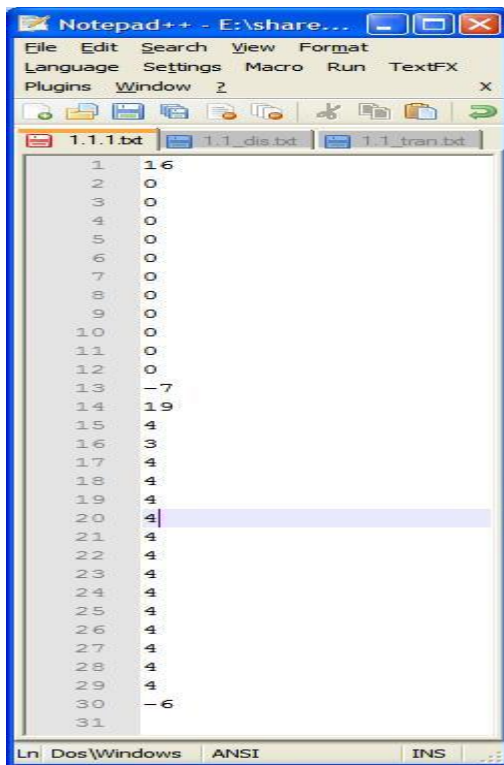
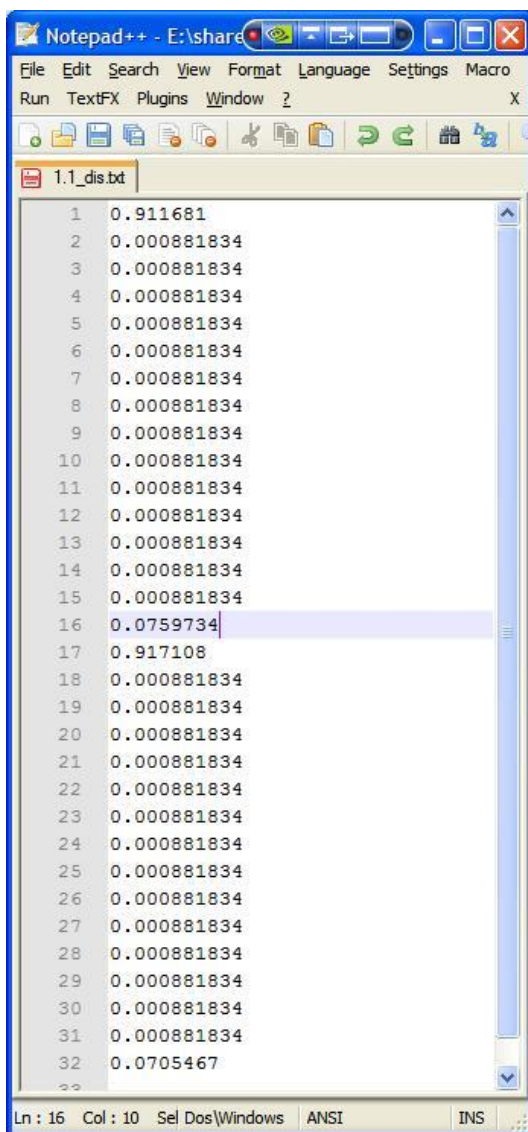


Figure 14 Feature data file

4.4.3 Distribution probability file

We mentioned in the previous section that there can be sixteen different numbers in the feature data files (not including the heading and ending process). Actually, those sixteen

numbers represent the sixteen possible feature values. Every feature takes one of the sixteen values in each state, and each value has an associated probability drawn from the value distribution. The probability distribution files contains a list of sections, and each section consists of sixteen rows which represent the sixteen different values available in the state. Like the raw data and feature data files, we still need to locate the border of the states, but no additional mechanism is required to do so. Since each state section always occupies sixteen rows in the file, we can locate the border by counting the number of rows. One thing we should notice is the sum of each state section should always equal one.

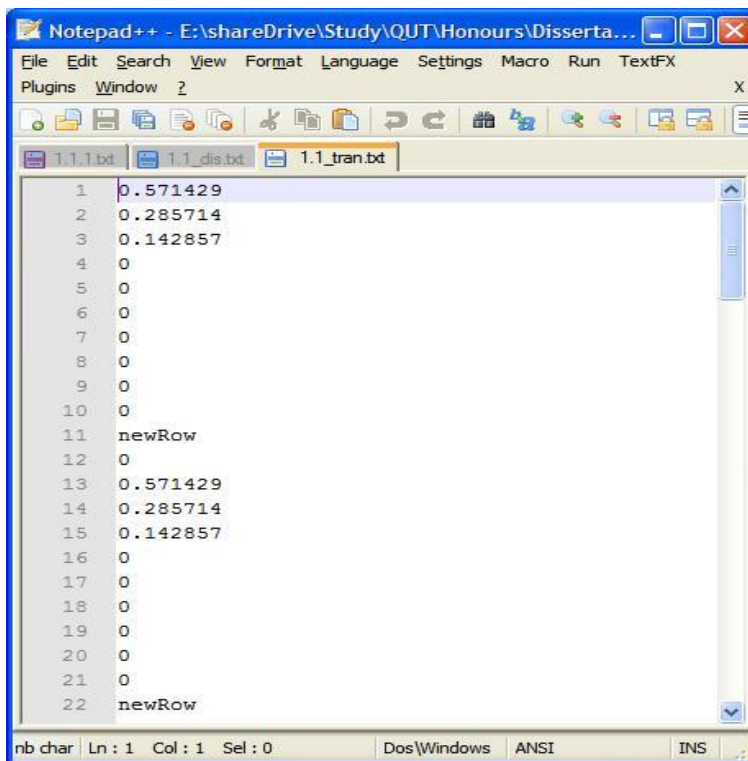


```
1 0.911681
2 0.000881834
3 0.000881834
4 0.000881834
5 0.000881834
6 0.000881834
7 0.000881834
8 0.000881834
9 0.000881834
10 0.000881834
11 0.000881834
12 0.000881834
13 0.000881834
14 0.000881834
15 0.000881834
16 0.0759734
17 0.917108
18 0.000881834
19 0.000881834
20 0.000881834
21 0.000881834
22 0.000881834
23 0.000881834
24 0.000881834
25 0.000881834
26 0.000881834
27 0.000881834
28 0.000881834
29 0.000881834
30 0.000881834
31 0.000881834
32 0.0705467
```

Figure 15 Distribution data file

4.4.4 Transition probability file

The transition probability describes the probability of moving from one state to another. Since different characters may contain a different number of states, we can not use the “counting row” method to locate the border of states. The probability is always between 0 and 1, so we can use the mechanism used in raw and feature data file. However, in some middle stages of the recognition process, we want to keep the summary of probability equal one. Furthermore, each section in the transition probability files represents one row in a 2D array. So instead of using the “add/minus” mechanism, we manually add some tag between sections. The tag we used is “new row”. The screen shot below shows part of the transition probability file.



```
1 0.571429
2 0.285714
3 0.142857
4 0
5 0
6 0
7 0
8 0
9 0
10 0
11 newRow
12 0
13 0.571429
14 0.285714
15 0.142857
16 0
17 0
18 0
19 0
20 0
21 0
22 newRow
```

Figure 16 part of a transition probability file

4.4.5 Result file

The result file contains the recognition results for a character. At the top of the file, it shows information about input character for recognition, and a sequence of match patterns are listed below in descending order by their similarity. A ranking number mechanism is used to make the result easier to read. The screen shot below shows the

recognition result of character with ID 1.2.

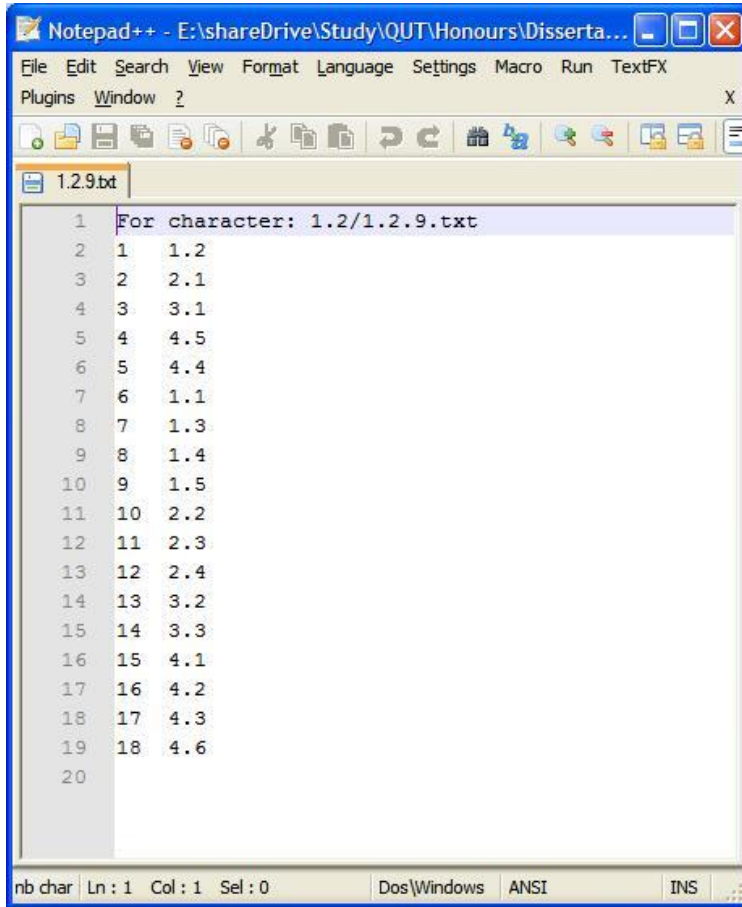


Figure 17 Result file

4.5 Initial raw data collecting and processing


As discussed in the writing pad section, every time the `GDK_POINTER_MOTION_HINT_MASK` is called, it will record the current position of the cursor in the writing pad, and until the user issue a signal to the writing pad to indicate the end of input, the writing pad will output the list of position to a text file. The interval between `GDK_POINTER_MOTION_HINT_MASK` events is only 0.05 second, as the result, we could end up with a huge number of cursor positions. If we use these data without some necessary processing, the overall efficiency of the recognition system could be significantly decreased. The `GDK_POINTER_MOTION_HINT_MASK` interval is fixed, and we can not do much about it. But we can do is periodically pick up some cursor positions rather than pick up all. There are two easy implementation solutions for this problem. One is setting up some time frame, for example double or

triples the GDK_POINTER_MOTION_HINT_MASK interval. Another one is setting up the distance between two cursor positions. The former one may introduce a new problem to the data collection. For example, if the user inputs the first half part of the stroke quite slow, and second half part of the stroke quite fast. Most of the data will be representing the first half of the stroke, the first part will become the dominating part and the whole stroke is not balanced. In this project, we take the latter approach, and set the distance interval to 5 pixels. The size of distance interval can be slightly adjusted, but can not be less than 3. The reason for this will be given in “Feature analysis” section.

4.6 Feature analysis

HMM is a pure mathematical model, but handwriting recognition has elements of Graphic User Interface processing. We need have some mechanism sitting between them, to enable them to talk to each other. As mentioned in the literature review, the HMM needs five essential elements. To apply the HMM to the real world research problem, the elements “state” and “observation” should be firstly resolved. We need to decide what the states and observation should correspond to in the model correspond to, and then deciding how many states should be in the model.

4.6.1 Character decomposition

In this project, we use the single Chinese character as the basic unit for training and recognition. Each unit consists of several sections which correspond to the strokes in that character. The section is the container of the “states”, and each section contains a fixed number of states. In other words, we used a fixed number of states to represent a stroke. Now, we go back to section 3.2.1. By observing the Figure “Chinese character stroke list”, we can see the most complicated stroke of the Chinese character is “”, which consists of five segments. What this means is that we need at least use five states in a stroke to cope with all the possible occurrences. Theoretically, the more segments (or states) the strokes have, the more accurate the recognition results should be, but the efficiency of the recognition system will go down as it needs to process more states in each stroke. In this project, we focus on the feasibility of HMM and reasonable accuracy. So we start with five states in each stroke. To make the definition clearer, we use a two-stroke character as

an example. The character “人” can be represented using the following state sequence: $S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9$.

4.6.2 State decomposition

Up to this stage, we have decomposed the Chinese character into a sequence of stages, but we haven't completely finished the mapping between the Chinese character and the raw data. Furthermore, we haven't decided what the HMM observations correspond to yet. What we are going to do here is to pick up a pair of cursor positions from a raw data file. Their position vectors relative to the co-ordinate origin define an angle. We may divide the entire circular arc into sixteen equal parts, and assign each part a number starting from 0 and ending with 15. Then, based on the angle between successive positions, we use the corresponding number to replace the position data, and output to a text file called a feature file. We repeat the procedure for all the raw position data, and finally we will get a collection of feature data which corresponds to the original raw data files. The picture below shows the sixteen segments of the arc.

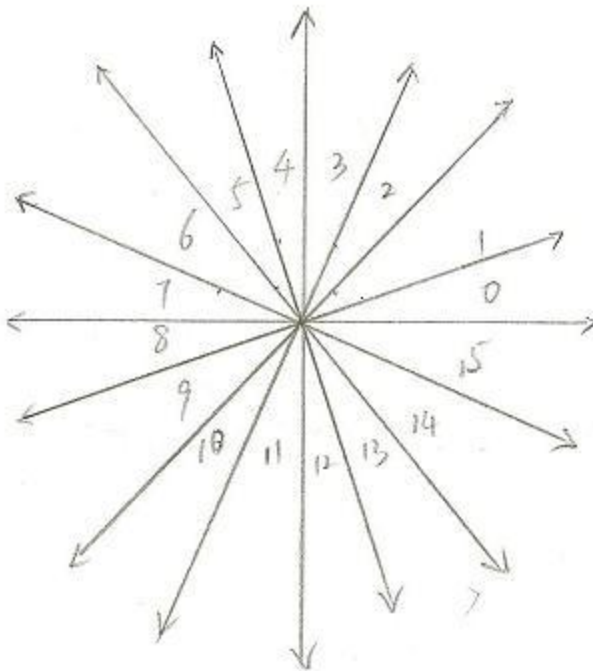


Figure 18 Coordinate segmentation

After getting the feature data, the observation problem has been solved implicitly.

Basically, we can use the feature data as the observation within the HMM. The last thing we need to do is to create the relation between the states and features. We let the state be the container of features, and store the features by allocating them according to their feature number. From the software engineering point of view, the state contains an array of size 16. Every item in the array contains the number of the same type of features. For example, a state which contains 35 features may have the following structures:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	3	1	5	0	4	3	1	1	1	1	2	5	2	4

Figure 19 Feature distribution in a state

This state does not contain feature with number “0” and “5”; contains only one feature for feature with number “3”, “8”, “9”, “10” and “11”; contains two features for feature with number “1”, “12” and “14”; contains three features for feature with number “2” and “7”; contains four features for feature with number “6” and “15”; contains five features with number “4” and “13”.

By now, the entire GUI character has been transformed into a sequence of number which can be managed by the HMM.

4.7 Training state initialisation

In the feature analysis phase, we have already figured out two elements of HMM, the “state” and the “observation”. In order to apply the HMM to our recognition system, we need to figure out the remaining three elements before we move on. In this project, we will deliver some software prototype rather than a perfect recognition system. So some assumptions need to be made at the very beginning. Firstly, we assume the users know the correct order in which to write the strokes for particular Chinese characters. Secondly, we assume the users know the correct way to write the Chinese strokes. Every Chinese character has a standard form, and we made the assumption that all the users are aware of this standard. What it means is that for a character, we always start at the same stroke and use the same way to draw that stroke. In terms of the initial state distribution π , it is certain we always start at state S_0 .

4.7.1 Observation segmentation

The HMM is associated with a formal statistical methodology. In our project, we have a number of training examples for each character. Since all the raw data files which represent the same character are grouped together (see Data organization section), we can easily process them group by group. In each raw data or feature file, there is no separator between the states. One of the goals we want to achieve by using the Viterbi algorithm (see section “Customized Viterbi Algorithm”) is to find out the proper border between the states, but before we do that, we need roughly to specify a border to the states. The approach we use is to try to equally divide the list of feature observations in each stroke in each feature file into a fixed number (the number is 5 in our project). For example, if there are 16 feature observations in a stroke, we will end up with 3, 3, 3, 3 and 4. What this means is that the first three feature observations belong to state one; the second three feature observation belong to state two, and so on.

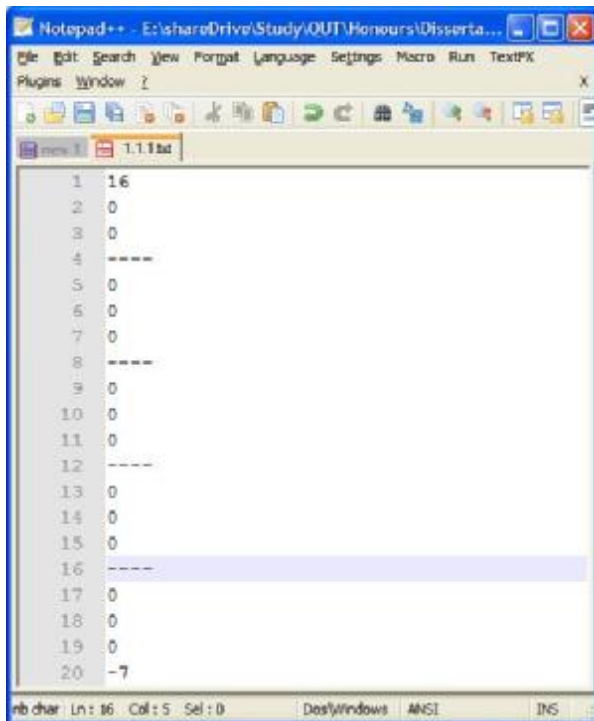


Figure 20 Observation segmentation

4.7.2 Feature distribution

After roughly dividing the strokes into five states, now we can process the states. We go through each file in the group and count the accumulated value of each feature in each

stroke. For example, if we have two training example files one and two, for state X, after the process, we will end up with the last table shown below.

State X in file one: (table one)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	3	1	5	0	4	3	2	0	0	2	2	5	2	4

State X in file two: (table two)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	4	0	3	2	3	3	2	0	0	3	6	4	2	2

Static State X value: (table three)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	3	7	1	8	2	7	6	4	0	0	5	8	9	4	6

After we get the statistic in the state, we can start calculating the feature probability distribution in each state. Still using the example above, we can see the total number of features in the state is 71, so the probability of feature “0” d_0 is $1/71 = 1.41\%$, $d_1 = 4.23\%$ and so on. One interesting thing is that for feature “8” and “9”, there is no occurrence in table one and table two which lead to the assignment of d_8 and d_9 to 0. Due to the time constraints of this project, we only can employ a few training examples, but it is quite possible that these training example didn’t completely cover all the possible feature occurrences. If we only use the training example to make the training model, when another variation emerges, this model may not handle it successfully. The improvement we use to fix this problem is to give an extremely small probability to those kinds of features. The formula to calculate the small smoothing value is:

$$Probability = 1 / (80 * N)$$

Figure 21 Probability formula

Where N is the number of features which have no occurrence. In the above example, N =

2.

It seems to be trivial improvement at this stage, but it can make a huge difference in terms of the recognition results. For instance, suppose we have two user inputs which represent the same character, and one of the inputs has a 70% match and all features observations are contained in the character model; and another one has a 99% match but a few feature observations have no probability in the character model. Before the employ the improvement, the latter higher match character will be abandoned, as the zero probabilities will cause the overall prediction to have probability zero.

By adding these extra values to the probability, even though extra values are extremely small, it will make the overall probability exceed 1. Before the distribution can be used, we need normalise it to 1. The approach we take is quite simple, we recalculate the total probability mass P_{new} (should be larger than 1), and let each probability in the distribution be weighted by $1/P_{\text{new}}$.

4.7.3 State Transition

To understand and calculate the state transition probability, we need to figure out the state transition architecture used in our project. As mentioned before, every stroke consists of five states, and every character consists of a list of strokes. So a character can be described by the picture below:

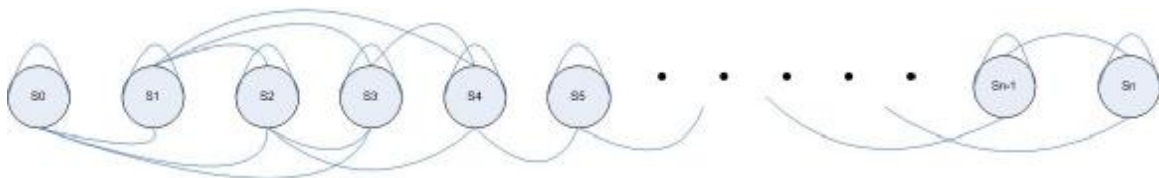


Figure 22 State Transition Architecture

As the time goes, the system can stay in the same state or it can jump to following states. In the picture above, we set the max state jump to 3. In addition, we know users always need to finish drawing one stroke before they start drawing the second one. Lastly, the last state in the stroke can jump the first state in the following stroke and repeat the procedure. So we combine these two rules, and get the following rule: as time goes on, the system may stay in the state or jump to following states; a state jump is permitted

up to length MAX or until it reaches the last state of the stroke it belongs to; and the last state allows a transition to the first state in the following stroke. By observing the picture above, we can see, S0 and S1 can jump 3 times; S2 can only jump twice; S3 can only jump to S4; and S4 only can stay at the same place or jump to S5 which is the first state in the following stroke. Actually, the procedure happened to the first stroke will repeat in the following strokes as well. In every stroke, it always starts at the first state, and end at the last state then jump to the first state in the next stroke.

Now we go further to the inside of the state. The features in the state follow a similar pattern to the state transition architecture. There is a sequence of features labelled as f0, f1 and so on, and only the last feature can jump to the first feature in the next state. For instance, there are two adjacent states, and both of them contain 6 features. So we used the following feature sequence to present these two states: f0, f1, f2 ... f11. The border between these two states sits between f5 and f6. In other words, in the first state only f5 can jump to f6 which is the first feature in the second state. By observing the sequence of the features, we can see that if we randomly pick up one feature, there is only 1/6 chance a state transition could happen, and 5/6 chance the feature transition could happen which happen only within one state. It's not hard to note that the calculation just discussed is only suitable for the state transition with max jump number equal one. In our project, however, we set the max jump number equal three. So we need develop another calculation method on top of the existing one. In the new method, we retain the same probability of a self-transition, and for all the following transition probabilities, we let the next probability equal half of the current probability. Still using the same example shown in Figure "State Transition Architecture", we suppose that S0 has N features, so the probability no transition is $(N-1)/N$, the probability of $S0 \rightarrow S1$ is $0.5*((N-1)/N)$, and the probability of the transition $S0 \rightarrow S2$ is $0.5*0.5*((N-1)/N)$. Finally, we normalise the probability using the approach we discussed before, and we should get a state transition matrix like this:

	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
S0	0.5714	0.2857	0.1429	0	0	0	0	0	0	0
S1	0	0.5714	0.2857	0.1429	0	0	0	0	0	0
S2	0	0	0.5714	0.2857	0.1429	0	0	0	0	0
S3	0	0	0	0.6667	0.3333	0	0	0	0	0
S4	0	0	0	0	1	1	0	0	0	0
S5	0	0	0	0	0	0.5714	0.2857	0.1429	0	0
S6	0	0	0	0	0	0	0.5714	0.2857	0.1429	0
S7	0	0	0	0	0	0	0	0.5714	0.2857	0.1429
S8	0	0	0	0	0	0	0	0	0.6667	0.3333
S9	0	0	0	0	0	0	0	0	0	1

Figure 23 Transition Distribution Matrix

This is transition distribution matrix for a character that consists of two strokes. We can see this matrix follows all the rules we defined before. The only thing we didn't mention but appears in the matrix is the transition probability $S4 \rightarrow S4$ and $S4 \rightarrow S5$. We can see all the rows except row S4 in the matrix follow the rule: the total probability should equal one. The reason for this is that we the last state must stay at the same place, but after that we need to give some instruction to make sure next state must start at the first state in the second stroke.

4.8 Training state optimisation

The distribution and transition probability we got from Training state initialization are approximate values, so we can not use them straight way. But all the values we got from previous steps are very good starting point. In this section, we will introduce our customised Viterbi algorithm, and use the existing distribution, transition probability and feature files to find out the best state path with which to interpret the observed feature sequence. Then starting from there, we review the feature files, and recalculate the optimised distribution and transition probability for each character. Some of the steps in this section are just repetition of training state initialisation. For consistency, we still list them as parts of this section, but in terms of the actual content, please refer to the previous section.

4.8.1 Customised Viterbi algorithm

Step 1: Initialization

$$\alpha_1 = \log b_i(O_1)$$

$$\Phi_1(i) = 0$$

Step 2: Recursion. From time $t=2$ to T .

$$\alpha_t(j) = \max[\alpha_{t-1}(i) + \log a_{ij} + \log b_j(O_t)],$$

$$\Phi_t(j) = \operatorname{argmax}[\alpha_{t-1}(i) + \log a_{ij}] \quad i < j$$

Step 3: Termination. (S_F is the final state set.)

$$B_1(\omega) = \log p^*(O|\lambda_\omega) = \max[\alpha_T(s)],$$

$$s_T = \operatorname{argmax}[\alpha_T(s)]$$

Step 4: State path backtracking. From time $t = T-1$ to 1.

$$s_t = \Phi_{t+1}(s_{t+1})$$

Figure 24 Customised Viterbi algorithm

As discussed previously, when drawing the character, we always start at the first state S_0 , so in the “initialization” stage, there is no the initial state distribution π required. In the initialization stage, we initialize two values, the probability at first state α_1 , and the best path to first state Φ_1 . With regards to Φ_1 , one thing I want to make clear is the path indicates the best path between two states, not the best path through the whole sequence of states. The best path through the whole sequence of states will be calculated at the “State path backtracking” stage. Basically, the probability at first state only depends on the feature distribution probability.

In the “Recursion” stage, the probability at the current stage depends on three factors: the probability at the previous stage, the transition probability between previous and current states and the feature distribution probability. Ideally, any states at time $t-1$ can jump to the current state at time t , so there will be a list of probabilities. We pick up the largest value, and store it as the probability of the current state². The best path calculation use the similar calculating approach, but one more additional step need to be taken. After we find

² It should be noted that the log probability values we get from the formula are always be negative number, so when we try to pick up the bigger, we try to pick up the one closer to 0, not the absolute value of the number

out the biggest value, we need map the biggest value to its origin. In other word, we need find out which path (connecting the previous state and the current state) can make the largest probability value occur, and record it as the current Φ .

In “Termination” and “State path backtracking” stages, when we reach the last observation at time T, we will end up with a list of probability values. The state with the largest probability will be the ending state of the overall state sequence, and in this project we should always expect the ending state to be last state in a character. Since in the “Recursion” stage we already the best path for each state, now we just simply trace back.

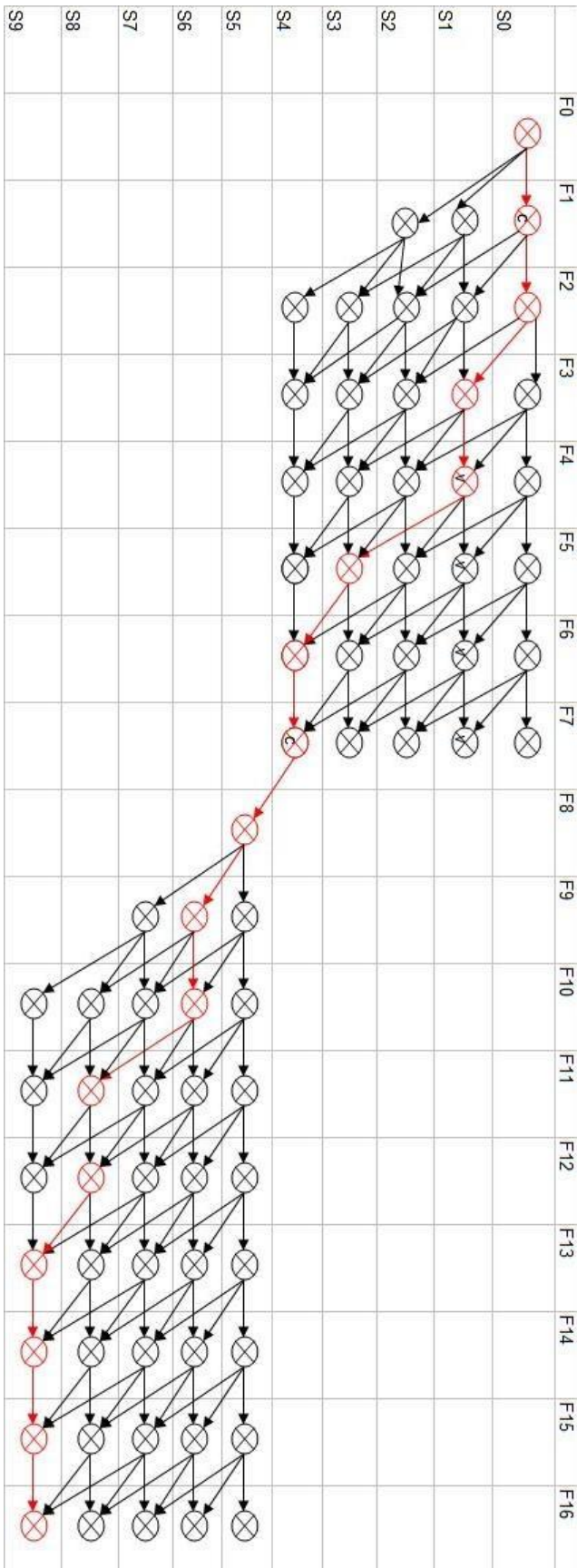


Figure 25 Customised Viterbi Algorithm demo

The picture above is a simple demo for a character with two strokes. We assume one of the feature files contains 17 features (the first 8 feature represent the first stroke and the rest represent the second stroke), so we should form a 10 row (we set the state number for each stroke equal 5, and the max jump number of state equal 3) and 17 column matrix. The icons including black ones and red ones show all the possible paths which occur and the red ones indicate the final best path. Actually, the picture does not show all the actual paths occurring in the system, it only shows the paths which have enough weighting to affect the final result. For example, for transitions between F7 and F8, this picture shows the transition between S4 and S5. Actually, a transition between any two of S0, S1, S2, S3, S4 may still happen, but since they are heavily penalized, their probabilities are too small to affect the overall result, so to make the effective paths more clear, we simply ignore the rest in this picture.

By observing the picture above, we can see the state jump in forward direction. That is because in the transition probability file, we set the backward probability to 0. In addition,, we can see that the border between two strokes is very clear. To make this happen, we use a penalty mechanism. We give a large penalty to the state transition which we do not expect to make sure that state transition will not be counted in the best path even it does happen. For instance, if a feature is the first feature for a stroke, the state must be the first state of that stroke, like feature F0 must stay at state S0 and F8 must stay at S5.

4.8.2 Observation segmentation

After applying the customised Viterbi to a feature file which is from the state training initialization, we get a corresponding state sequence file. We use the corresponding state sequence as the index original feature file, and redefine the border of the state in the feature files. We use the following files (one is an optimised segmentation file, the other one is the corresponding state sequence file) to demonstrate:

```

1 16
2 -----
3 -----
4 0
5 -----
6 0
7 0
8 0
9 0
10 0
11 0
12 0
13 0
14 0
15 0
16 0
17 0
18 0
19 0
20 0
21 0
22 -7

```

```

1 0
2 2
3 4
4 4
5 4
6 4
7 4
8 4
9 4
10 4
11 4
12 4
13 4
14 4
15 4
16 4
17 4
18 4
19 4

```

Figure 26 Optimised Observation Segmentation

By using the state sequence as the index, we can tell the first feature in the original feature file belongs to S0; there no feature belongs to S1 and S3; the second feature in the original feature file belongs to S2; all the rest belong to S4. By comparing the new observation segmentation file and the one in initialisation, we can see that these two files which are original from the same feature file now have different borders.

4.8.3 Feature distribution

Please refer to section “Feature distribution” in “Training state initialisation”.

4.8.4 State Transition

Please refer to section “State Transition” in “Training state initialisation”.

4.9 Character recognition

This is the final stage of the entire recognition system. There is no new mechanism added in this stage, and it just a simple reuse of the algorithm and mechanism we developed before and do a comparison between the runtime inputs with the training model stored in the system.

4.9.1 Feature analysis

Please refer to section “Feature analysis” in previous section.

4.9.2 Recognition

The recognition stage is very similar to the optimisation stage, and we still use our customized Viterbi algorithm. For testing purpose, we use the stored input files (the two files we reserved in the very beginning) as the recognition files, but in the real system, we let use to input the character runtime and store it in some temp file to use as the recognition file. Here is a list of steps we need follow to recognise the input character:

1. Create a ranking list.
2. Pick up a reserved input file as the observation file in our customised Viterbi algorithm.
3. Pick up the distribution probability and transition probability files for a character stored in the database or file system.
4. Run the customised Viterbi algorithm and record the overall probability (we only used the overall path in the state transition optimisation, and only use overall probability here).
5. According to the probability, insert the character at the proper position into the ranking list.
6. Repeat step 2 to 5 until no more character data is left in the database or file system.

If everything is all right, we should end up with a list of character names, the top one in the rank is the most probable matching character. In other words, the recognition system predicts that the top one in the rank is the character the user tried to input.

Chapter Four: Experiment and Results

4.1 Data Sets

As mentioned before, two data sets were used in this experiment. One set contains all the training examples and the other set contains all the recognition examples. In the training

sets, there are 42 groups which represent 42 characters respectively, and each group contains 40 training example for a character. In the recognition data set, there are 42 corresponding groups and each group contains two additional recognition examples. The data selection criteria were considered in the previous section.

4.2 Evaluation Criteria

In this project, we use only a small database for testing and use C++ as the implementation programming language, so the system response is pretty good, and the recognition takes less than one second to finish. Our focus is then on the accuracy of the system, and the evaluation criteria are relatively simple: we look at the positive recognition rate.³

4.3 Result Evaluation

	First Try	Second Try
十	1	2
士	1	1
戈	2	2
冰	1	1
把	1	1
川	1	1
去	2	3
五	1	1
可	1	1
千	5	8
人	1	1
月	1	1
主	3	3
州	1	1
这	1	1
义	5	5
之	1	1
心	1	1
口	1	1
又	1	1
买	1	1
山	3	4
四	1	1
长	2	2
公	1	1
女	4	4

³ Positive recognition rate: the number of correct predictions/ the number of total recognition examples

犹	1	1
代	1	1
凹	1	2
朵	2	2
计	2	2
同	1	1
飞	3	3
鼎	1	1
专	1	1
己	1	1
凸	1	1
及	1	1
寄	3	5
阳	1	1
马	4	4
乃	1	1

Figure 27 Recognition Results

The first column shows the character we are trying to recognise. We try to recognise each character twice in the system, and the results are shown in the second and third columns. The number in each row indicates the ranking index of the target character in the system. As shown in the results matrix, 67% (56/84) of the characters are correctly recognised, and 98.8% (83/84) of the character are recognised in the top five positions.

We can see that our recognition system has a moderate positive recognition rate of around 70%, which is not exceptionally good, but not too bad either. There are a couple of reasons for that: Firstly, we use only 40 training examples for each character, around the minimal number one might use for training HMMs. To get better accuracy, more training examples should be used. Secondly, in the data selection stage, for better generalisation in the future, when we pick up the character stroke example, we must consider additional details. Originally, there are only five stroke groups (see figure below) in the Chinese characters, but when we select the stroke example, we included all the possible variations⁴ of the strokes which lead to the large similarity between training character examples.

⁴ See Figure 10 Chinese character stroke and variation



Figure 28 Basic Chinese Strokes

Chapter Five: Conclusion

In this project, we investigated the use of the Hidden Markov Model and Viterbi algorithm used them as a basis for customised recognition system for Chinese character and handwriting characteristics. We developed a data framework and a feasible approach to the problem, and conducted some experiments on our recognition system using a small database and got some reasonably promising results. We have delivered a working prototype system which may form the basis of a new system for red hat linux.

We use HMM and Viterbi as the basis of our project. Obviously, some other approaches are promising avenues of research for this important problem. For instance, Fuzzy Stroke Type (Chang & Wan, 1998), Structural approach (Y. J. Liu & Tai, 1988), and Rule-Based approach (J.-W. Chen & Lee, 1996) are very good candidates for this research problem.

Due to the constraint of time, we have only partially addressed the problem of recognition of Chinese characters. Obviously, the handwriting recognition system can be extremely complicated and the accuracy and performance may vary as more features are added in. In this section, we will give some suggestion with regard to the further work.

5.1 *Writing Pad XInput support*

While the mouse is quite a good writing input device, it is not as good as a dedicated input stylus or pen. It is now possible to buy quite inexpensive input devices such as drawing tablets, which allow drawing with a much greater ease of artistic expression than does a mouse. To enable the XInput support, please refer to “Writing pad XInput Support” section in Appendix A.

5.2 *Relative position handling and Duration handling*

As the improvement of accuracy, relative position handling and duration handling can significantly increase the positive recognition rate. As mentioned in the “Result Evaluation” section, one of possible reason we did not get very good results is that there are a lot of similar characters in our training set. In our project, we try to recognise the characters at a very generic level, and did not go to a fine level of detail. We tried to recognise the characters by the number of strokes and difference of features, but did not consider the position of strokes and their relative length.. For example, the characters “工” and “土” are treated as the same character in our recognition system. By adding relative position handling, the first stroke “一” will matter in terms of final result. With regards to the duration handling, we use character “士” and “土” as example. This time the stroke position are exactly same in two strokes, but the upper “一” in “士” is longer than the one in “土”.

We have listed two suggestions for the future work. Actually, there are quite a lot of more improvements can be made. We hope our research can provide helpful information for future researchers.

References

- Chang, C.-H. (1994). *Word class discovery for postprocessing Chinese handwriting recognition*. Paper presented at the *Proceedings of the 15th conference on Computational linguistics, Kyoto, Japan*.
- Chang, J.-Y., & Wan, M.-H. (1998). Fuzzy stroke type identification for online Chinese character recognition. Paper presented at the Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on.
- Chen, J.-W., & Lee, S.-Y. (1996). On-line handwriting recognition of Chinese characters via a rule-based approach. Paper presented at the Pattern Recognition, 1996., Proceedings of the 13th International Conference on.
- Chen, R. H., Lee, C.-W., & Chen, Z. (1994). *Preclassification of Handwritten Chinese Characters Based on Basic Stroke Substructures*. Paper presented at the Proc. Fourth Int'l Workshop Frontiers in Handwriting Recognition.
- Chen, Z., Lee, C.-W., & Cheng, R. H. (1996). *Handwritten Chinese Character Analysis and Preclassification Using Stroke Structural Sequence*. Paper presented at the Proc. 13th Int'l Conf. Pattern Recognition.
- Education, D. o. Chinese primary school text book.

- Gable, G. (2006). *Scientific Method - ITN100 Research Methodology Lecture Note*. Brisbane.
- Ge, Y., Guo, F.-J., Zhen, L.-X., & Chen, Q.-S. (2005). Online Chinese character recognition system with handwritten Pinyin input. Paper presented at the Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on.
- Hasegawa, T., Yasuda, H., & Matsumoto, T. (2000). Fast discrete HMM algorithm for online handwriting recognition. Paper presented at the Pattern Recognition, 2000. Proceedings. 15th International Conference on.
- Kim, H. J., Kim, K. H., Kim, S. K., & Lee, J. K. (1996). Online Recognition of Handwritten Chinese Characters Based On Hidden Markov Models. *Pattern recognition (Pattern recogn.)*, 30(9), 1489-1500.
- Leeds, U. o. (2006a). *Hidden Markov Model online tutorial*, from http://www.comp.leeds.ac.uk/roger/HiddenMarkovModels/html_dev/main.html
- Leeds, U. o. (2006b). *Viterbi algorithm online tutorial*, from http://www.comp.leeds.ac.uk/roger/HiddenMarkovModels/html_dev/viterbi_algorithm/s1_pg1.html
- Lin, C. K., & Fan, K.-C. (1994). Coarse Classification of On-Line Chinese Characters via Structure Feature-Based Method. *Pattern Recognition*, 17(10), 1365-1377.
- Lin, C. K., Fan, K. C., & Lee, F. T. P. (1993). Online Recognition by Deviation-Expansion Model and Dynamic-Programming Matching. *Pattern Recognition*, 26(2), 259-268.
- Lin, M.-Y., & W.-H.Tsai. (1988). *A New Approach to On-Line Chinese Character Recognition by Sentence Contextual Information Using the Relaxation Technique*. Paper presented at the Proc. Int'l Conf. Computer Processing of Chinese and Oriental Languages.
- Liu, C. (2006). Smart Common Input Method platform project, from <http://www.scim-im.org/>
- Liu, C. L., Jaeger, S., & Nakagawa, M. (2004). Online recognition of Chinese characters: The state-of-the-art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2), 198-213.
- Liu, J., Cham, W. K., & Chang, M. M. Y. (1996). Stroke order and stroke number free on-line Chinese character recognition using attributed relational graph matching. Paper presented at the Pattern Recognition, 1996., Proceedings of the 13th International Conference on.
- Liu, Y. J., & Tai, J. W. (1988). A structural approach to online Chinese character recognition. Paper presented at the Pattern Recognition, 1988., 9th International Conference on.
- Main, I., & team, T. G. (2007). *GTK+ 2.0 Tutorial*, from <http://www.gtk.org/tutorial/>
- Matic, N., Platt, J., & Wang, T. (2002). *QuickStroke: An Incremental On-Line Chinese Handwriting Recognition System*. Paper presented at the Proc. 16th Int'l Conf. Pattern Recognition.
- Nakai, M., Akira, N., Shimodaira, H., & Sagayama, S. (2001). Substroke approach to HMM-based on-line Kanji handwriting recognition. Paper presented at the Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on.
- Newton, I. (1999). *The System of the World* (B. Cohen & A. Whitman, Trans. 3 ed.): University of California Press.

- R.G.Casey. (1970). Moment Normalization of Handprinted Character. *IBM J. Research Development*, 14, 548-557.
- Rabiner, L. R. (1989). A Tutorial on Hidden Markov-Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2), 257-286.
- S.-W., Lee, & Park, J.-S. (1994). Nonlinear Shape Normalization Methods ofr the Recognition of Large-Set Handwritten Characters. *Pattern Recognition*, 27(7), 895-902.
- Tappert, C. C., Suen, C. Y., & Wakahara, T. (1990). The state of the art in online handwriting recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 12(8), 787-808.
- team, G. (2006). *Introduction to GTK+*, from <http://www.gtk.org/>
- Wakahara, T., Murase, H., & Odaka, K. (1992). Online Handwriting Recognition. *Proceedings of the Ieee*, 80(7), 1181-1194.
- Wikipedia. (2006). *Dynamic programming*, from http://en.wikipedia.org/wiki/Dynamic_programming
- Wikipedia. (2006). Hidden Markov Model, from http://en.wikipedia.org/wiki/Hidden_Markov_model

Appendix A: Writing pad

Event handling

Before considering the writing pad event handling, we give a brief explanation about theory of signals and callbacks in GTK+. GTK is an event driven toolkit, which means it will sleep the main controller event until an event occurs and control is passed to the appropriate function. This passing of control is done using the idea of "signals". (Note that these signals are not the same as the Unix system signals, and are not implemented using them, although the terminology is almost identical.) When an event occurs, such as the press of a mouse button, the appropriate signal will be "emitted" by the widget that was pressed. This is how GTK does most of its useful work. There are signals that all widgets inherit, such as "destroy", and there are signals that are widget specific, such as "toggled" on a toggle button. The necessary callbacks are just like method or function in a programming language.

Mouse moving and key pressing are low-level GTK+ events. The corresponding low-level signals and event handlers which have an extra parameter that is a pointer to a structure containing information about the event are required to handle these kinds of

events. For example, motion event handlers are passed a pointer to a `GdkEventMotion` structure which looks (in part) like:

```
struct _GdkEventMotion
{
    GdkEventType type;
    GdkWindow *window;
    guint32 time;
    gdouble x;
    gdouble y;
    ...
    guint state;
    ...
};
```

Where `type` will be set to the event type, `window` is the window in which the event occurred. `x` and `y` give the coordinates of the event, `state` specifies the modifier state when the event occurred (that is, it specifies which modifier keys and mouse buttons were pressed).

In the writing pad program, we are interested in finding when the mouse button is pressed and when the mouse is moved, so we specify `GDK_POINTER_MOTION_MASK` and `GDK_BUTTON_PRESS_MASK` as the `GdkEventType`. `GDK_POINTER_MOTION_MASK` introduces a new problem to the writing pad. This will cause the server to add a new motion event to the event queue every time the user moves the mouse. Imagine that it takes us 0.1 seconds to handle a motion event, but the X server queues a new motion event every 0.05 seconds, which means if the user keeps drawing for a several seconds, the server will be far behind to process the signal. The solution we use to resolve this problem is that we use `GDK_POINTER_MOTION_HINT_MASK` to replace `GDK_POINTER_MOTION_MASK`. When specifying `GDK_POINTER_MOTION_HINT_MASK`, the server sends us a motion event the first time the pointer moves after entering our window, or after a button press or release event. Subsequent motion events will be suppressed until we explicitly ask for the position of the pointer using the function `gdk_window_get_pointer`. (Main & team, 2007)

Drawing Area Widget and Drawing

A drawing area widget is essentially an X window and nothing more. It is a blank canvas in which the users can draw the character. It should be noted that when we create a `DrawingArea` widget, we are completely responsible for drawing the contents. If our window is obscured and then uncovered, we get an exposure event and must redraw what was previously hidden. Having to remember everything that was drawn on the screen so we can properly redraw it can, to say the least, be a nuisance. In addition, it can be visually distracting if portions of the window are cleared, then redrawn step by step. The solution to this problem is to use an offscreen *backing pixmap*. Instead of drawing directly to the screen, we draw to an image stored in server memory but not displayed, and then when the image changes or new portions of the image are displayed, we copy the relevant portions onto the screen.

To create an offscreen pixmap, we use the following function:

```
GdkPixmap* gdk_pixmap_new          (GdkWindow  *window,
                                     gint         width,
                                     gint         height,
                                     gint         depth);
```

The window parameter specifies a GDK window that this pixmap takes some of its properties from. width and height specify the size of the pixmap. depth specifies the color depth, that is the number of bits per pixel, for the new window. If the depth is specified as -1, it will match the depth of window. For a good look ‘n’ feel, we call `gdk_draw_rectangle()` to clear the pixmap initially to white. Finally, the exposure event handler then simply copies the relevant portion of the pixmap onto the screen.

After discussing how to keep the screen up to date with our pixmap, now we need to find out how to draw character on our pixmap. There are a large number of calls in GTK's GDK library for drawing on *drawables*. A drawable is simply something that can be drawn upon. It can be a window, a pixmap, or a bitmap (a black and white image). Here is a list of drawables:

```
gdk_draw_point ()
gdk_draw_line  ()
gdk_draw_rectangle ()
gdk_draw_arc   ()
```

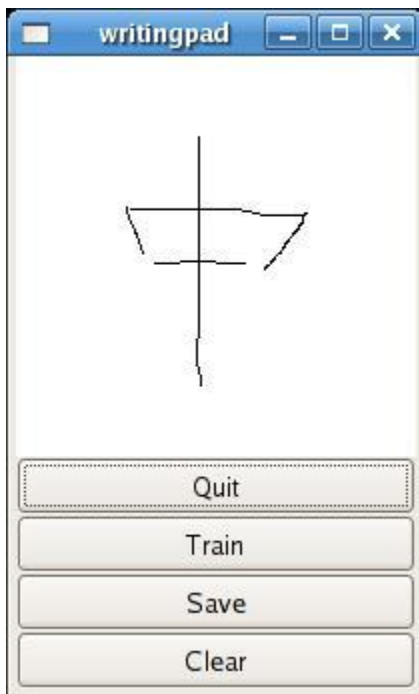
```

gdk_draw_polygon ()
gdk_draw_pixmap ()
gdk_draw_bitmap ()
gdk_draw_image ()
gdk_draw_points ()
gdk_draw_segments ()
gdk_draw_lines ()
gdk_draw_pixbuf ()
gdk_draw_glyphs ()
gdk_draw_layout_line ()
gdk_draw_layout ()
gdk_draw_layout_line_with_colors ()
gdk_draw_layout_with_colors ()
gdk_draw_glyphs_transformed ()
gdk_draw_glyphs_trapezoids ()

```

In our writing pad, we retrieve the co-ordinations from `GDK_POINTER_MOTION_HINT_MASK` which is discussed in previous section, then use `gdk_draw_line ()` to connect all the co-ordinations. It should be noted that the default line side of `gdk_draw_line ()` should be override to a larger value to achieve the better look 'n' feel.

GUI



This is a screen shot of the handwriting pad we developed. In the middle, there is a while spare space for user input. Users can use “Quit” button or the cross on the top-right

corner to close the application. The button “Clear” is used to erase the unwanted user input. Button “train” and “Save” are used in pair to take user input and store as training example. When users try to input a character example character, before draw anything on the writing pad, they need click button “Train” to inform the system the start of input. When users finish input the whole character, they need click button “Save” to save the input to file system.

Writing Pad XInput support

While the mouse is quite a good writing input device, it is not as good as a dedicated input stylus or pen. It is now possible to buy quite inexpensive input devices such as drawing tablets, which allow drawing with a much greater ease of artistic expression than does a mouse. In the GTK environment, the following steps need to be taken to make XInput support happen.

1. Enabling extended device information.

To let GTK know about our interest in the extended device information, we merely have to add a single line to the program:

```
gtk_widget_set_extension_events(drawing_area,GDK_EXTENSION_EVENTS_CU  
RSOR);
```

This statement will tell the system that we are interested in extension events.

2. Using extended device information

Once we've enabled the device, we can just use the extended device information in the extra fields of the event structures. In fact, it is always safe to use this information since these fields will have reasonable default values even when extended events are not enabled. Once change we do have to make is to call `gdk_input_window_get_pointer()` instead of `gdk_window_get_pointer`. This is necessary because `gdk_window_get_pointer` doesn't return the extended device information.(Main & team, 2007)