國立交通大學

多媒體工程研究所

碩 士 論 文

線 上 樂 譜 手 寫 辨 識 系 統

An Online Handwritten Recognition System of Music Score

研 究 生：龔信嘉

指導教授：陳玲慧　教授

中 華 民 國 九 十 七 年 六 月

線上樂譜手寫辨識系統
An Online Handwritten Recognition System of Music Score

研 究 生：龔信嘉　　Student：Sin-Jia Gong

指導教授：陳玲慧　　Advisor：Dr. Ling-Hwei Chen

國 立 交 通 大 學
多 媒 體 工 程 研 究 所
碩 士 論 文

A Thesis
Submitted to Institute of Multimedia Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer Science

June 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年六月

線上樂譜手寫辨識系統

研究生：龔信嘉　　　　　　　　指導教授：陳玲慧 博士

國立交通大學多媒體工程研究所

摘要

本篇論文中我們提出了一個線上的樂譜手寫辨識系統。樂譜是用來記錄樂曲的工具，作曲家常用其於創作與交流音樂。論文中，我們使用與在紙上相同的書寫方法，利用多筆劃組合出音樂符號。而筆劃的特徵有三種，高度，組成之基本圖形以及方向，可用來辨識出此筆劃所屬類型，再將其組合成所需要的音樂符號。本系統支援基本創作需要之全部音樂符號，辨識率為 98.35%，並且提供方便及完善的樂譜修改功能。

索引詞：手寫、音樂符號、線上、筆劃

An Online Handwritten Recognition System of Music Score

Student: Sin-Jia Gong Advisor: Dr. Ling-Hwei Chen

Institute of Multimedia and Engineering

National Chiao Tung University

Abstract

In this thesis, we present an online handwritten system for music score recognition. Music score is used to record a music song. People often used to compose a music score on the sheet of paper. In our system, we propose the pen based writing method and use multi-strokes to form a music notation. We extract the height, shape and direction from a stroke as the features and recognize it as a symbol. Then the symbol is combined with other symbols to form a music notation. The system is robust for a general use and supports enough music notations for composition. The recognition rate is 98.35%.

Index term: handwritten, music score, music notation, online, stroke

誌謝

這篇論文能夠完成，最主要感謝指導教授陳玲慧博士在這兩年內細心的教導，讓我在這兩年的碩士生涯中，學習到研究的方法以及態度，也體認到師生之間相處的情誼。

接著也感謝實驗室的學長、同學和學弟們，對於系統的架構以及資料庫的建置上給予了非常多的建議，在生活上的大小事情也給了非常多的協助。特別感謝井民全學長和郭萓聖學長在技術上的支援；同屆的三位夥伴，王偉全、徐子翔和黃薰瑩在課業及研究上的陪伴；還有李惠龍學長、楊文超學長、陳俊旻學長、尤瓊雪學姊、陳立人學長、何維中學長、林芳如學姊、林佩瑩學姊、張明旭學弟、楊志鴻學弟、郭益成學弟和林志達學弟，少了你們實驗室將會失色許多；另外還有大學時期同窗好友，陪我嬉鬧談笑解苦悶，增添許多人生樂趣。

最後要感謝我的父母，一直能提供這麼棒的環境讓我專心於研究上，讓我無後顧之憂。僅以誠摯的心將此論文獻給我父母。

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Music score is a handwritten or printed form of music notations, and it is often used in music composition and music representation. It consists of staff, clefs, notes, rests, and signatures …, etc.

The common way to record a music score is to write the score on sheets of papers by pencil or pen. As the computer technology grows rapidly, Musicians use computer to aid their composition. In early period, optical music recognition (OMR) is used to recognize the music score which is scanned to an image. However, the error rate of OMR system is relatively high and the editing work of the music score is slow and tedious [1]. Due to the inconvenience of OMR, the online music editing system is proposed. The system can directly output the editing resultant to musicians. Besides, more convenient systems are rapidly developing for user to write on the tablet. One is the "point and click" system, such as MagicScore Maestro [2] and Allegro [3], which selects music notations from menus or icons. Hence, the system can directly input the music notations without recognizing them. Nevertheless, the input processes are tedious and complicated due to many pen and mouse movements [4].

In order to reduce the tedious input processes, gesture-based music score recognition systems are developed. Musicians could use specific gestures to represent specific notations defined by systems. Forsberg *et al.* [5] proposed such a system which uses gesture and voice to input the music notations. In the gesture part, it combines Calligrapher system [6], Rubine's gesture recognition system [7] and their recognizer to recognize the input gesture. The supported music notations are limited and are not sufficient for professional music editors, and some gestures are irrelative to the shapes of the corresponding music notations. This makes learning curve long and difficult. Anstice *et al.* [1] also proposed a gesture-based system called Presto. After that, Ng *et al.* [8] proposed an improved version denoted as Presto2, which improves both usability and speed of input, but the gestures in the system have little relation with the actual writing. The recognition accuracy of gesture-based system may be acceptable. However, in the gesture-based systems, users must learn and remember these miscellaneous gestures. Therefore, the gesture-based system is often very constraining for the user.

Instead of learning miscellaneous gestures, pen-based handwritten systems are developed to catch the human writing styles. The characteristic of pen-based systems is that the writing styles is as the same as on sheets of papers. There are several methods proposed, like neural network, context-free grammar and SVM. In 2003, George *et al.* [4] proposed such a system with artificial neural networks. They used a multi-layer perception to learn music notations and extract the features. The inputs of these handwriting systems are natural and direct for users, but the error rate may be alarmingly high. Subsequently, music notations can be recognized by the trained neural networks. Taubman *et al.* [9] proposed a handwritten music recognition system based on statistical moments. Nevertheless, the current system is not stable and not robust enough for a general use. In 2005, Macé *et al.* [10] proposed a generic method which recognizes the music score by context-free grammars and lots of recognizers. Unfortunately, the user must follow the writing orders and writing locations that are defined by professional musicians, and it is not friendly for the users that are not familiar with the music theory. Miyao and Maruyama [11] proposed a handwritten system based on time series data and image features. Their system uses dynamic programming and SVM algorithm to recognize handwritten music notations. However, only a small part of music notations is supported in the system. In other way, the system does not support modification operations, such as deleting or moving a notation, and this makes the system impractical.

In this thesis, we propose an online handwritten recognition system of music score which utilizes the input stroke properties, including the height, the shape and the direction. With these properties, we build a pen based system with high recognition rate. The system also supports enough music notations and intuitional modification operations for general use.

The rest of the thesis is structured as follows. Chapter 2 describes the strokes used in the system. The proposed recognition method is discussed in Chapter 3. The proposed method has been evaluated by experiments as reported in Chapter 4. The final chapter closes the thesis with conclusion and future researches.

CHAPTER 2

STROKE DATABASE

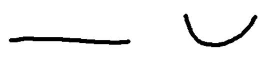
A stroke is a collection of points from pen-down to pen-up. A music notation or notation is the basic unit to record music, including staff, clefs, notes, rests, and signatures …, etc. When we are writing, there are some notations we cannot write in a single stroke, like natural or sharp. We have to write multiple strokes to represent a notation. In other way, some notations have innumerable dots, heads, or flags, and we cannot assure the exacted strokes in these notations. In a word, a notation could be considered as a collection of many strokes, and a stroke is a basic input unit in the system. In our system, as a new stroke is written, it would be recognized as a meaningful symbol. The system collect the symbols inputted in previous time, and try to convert them to a meaningful notation. Here, we divide the strokes into 17 kinds of symbol categories, as shown in Table 2.1.

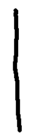
In Table 2.1(1), categories (1) to (6) are called "simple symbols, " which means that they could be recognized quickly by some extreme properties, like the stroke length. The others are called "complex symbols," which means they need to extract the features and be classified by the complex symbol classifier which will be elaborated in the next section.

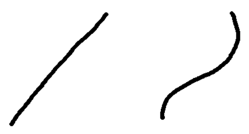
In our database, we obtained the strokes using a WACOM digital tablet written by 14 users. The users are not expert musicians and do not have any knowledge about the music theory. They write the strokes for all symbols on the digital tablet and the staff is pre-drawn on the screen. Each gap on staff is 25 pixels. For robustness, the procedure would be carried out at least 1000 times to each user.

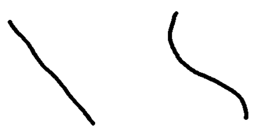
Table 2.1 Supported symbols (*continued*).

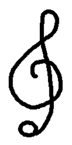
(1) Dot

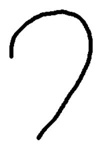

(2) HLine

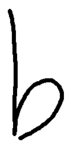

(3) VLine

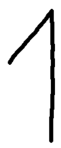

(4) Slash (Flag)

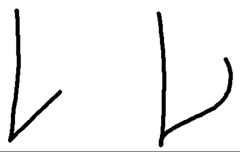

(5) UHook (Flag)

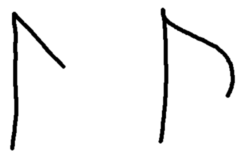

(6) GClef

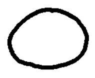

(7) FClefArc


(8) Flat


(9) NaturalRt


(10) LCheck


(11) StUHook


(12) WHead

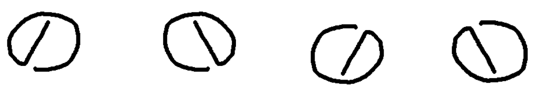

(13) BHead

Table 2.1 Supported symbols.

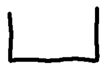
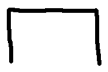
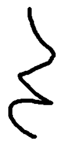
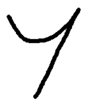
			
(14) WRest	(15) HRest	(16) QRest	(17) 8Rest

Table 2.2 shows all the supported music notations in this system. There are four types of notations supported. Table 2.2(a) is bar line which is used to separate the bars. Table 2.2(b) shows two examples of group which is a beam note formed by grouping several notes with filled head using a horizontal beam. Table 2.2(c) is the determinable note which consists of fixed number of symbols. Table 2.2(d) shows some uncertain notes with innumerable symbols, such as dot, flag...etc.

Table 2.2 Supported music notations (*continued*).

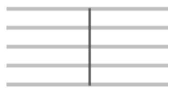
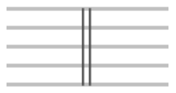
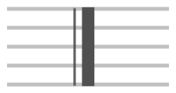
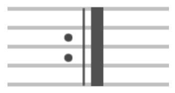
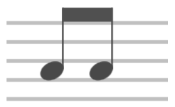
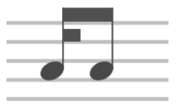
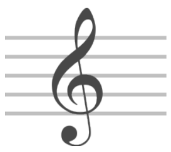
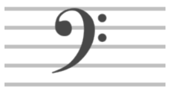
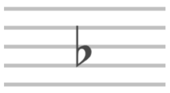
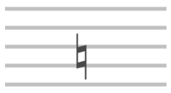
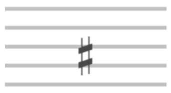
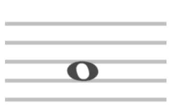
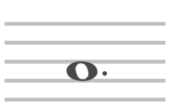
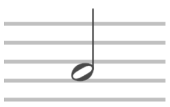
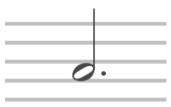
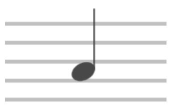
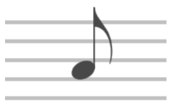
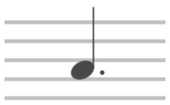
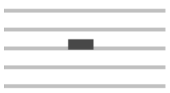
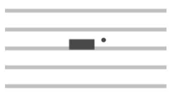
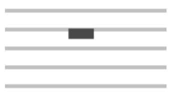
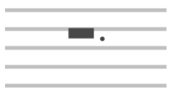
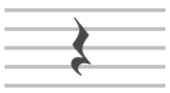
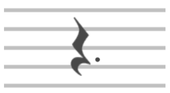
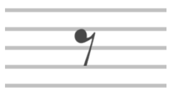
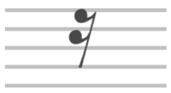
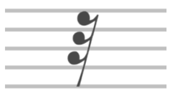
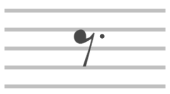
(a) Bar line			
			
(b) Group			
			

Table 2.2 Supported music notations.

(c) Determinable note				
				
(d) Uncertain note				
				
				
				
				

CHAPTER 3

THE PROPOSED METHOD

In this system, we recognize the input stroke as a symbol and then combine the symbol with other symbols to form a music notation.

The flow diagram of the symbol recognition is shown in Fig. 3.1. The whole process consists of 4 major phases: preprocessing, simple symbol classifier, feature extraction and complex symbol classifier. In the preprocessing phase, the noise and variety in the stroke would be eliminated. In the simple symbol classifier, the stroke with extreme properties would be recognized as a symbol and output as the result. If the stroke in the simple classifier is not recognized, it would be processed in the next phase. In the feature extraction phase, height, shape and direction are extracted from the stroke as features. In the complex symbol recognition phase, based on the extracted features, the stroke would be recognized as a symbol by the decision tree. In the decision tree, some similarity measures are provided to determine the most similar symbol in the database.

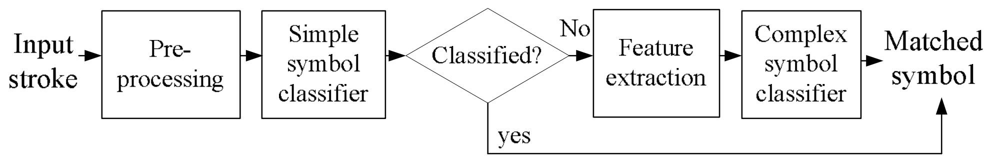


Fig. 3.1 Flow diagram of the symbol recognition.

After the symbol recognition, the notation recognition is conducted. Based on the semantic information, the output symbol would be combined with other existed symbols to form a notation. Finally, the system outputs the printed music notation and puts it at the exact location on staff.

3.1 Preprocessing

There are some problems after sampling the stroke. First, the stroke captured by the digital tablet tends to contain some noises which make the stroke jagged. Second, the stroke is sampled from the digital tablet by the fixed time interval, so the writing speed would affect the captured stroke. Finally, as pen-up and pen-down, there are some slips occurred which is caused by the user. In order to reduce the noise and variety in the stroke, we apply the preprocessing, including smoothing filter, gap filter and slipped segment remover.

3.1.1 Smoothing filter

The reason why a stroke jagged is that some errors occurred in the digital tablet or the unstable state the user is writing in. In order to eliminate these jags, we apply Gaussian filter [12] to make the stroke more smooth and keep the global information of corners in the stroke.

For every point ($x(t)$,$y(t)$), we smooth the stroke by

$$x(t)' = \sum_{i=-3\sigma}^{3\sigma} w_i x(t), \quad y(t)' = \sum_{i=-3\sigma}^{3\sigma} w_i y(t), \tag{1}$$

where

$$w_i = \frac{e^{-\frac{i^2}{2\sigma^2}}}{\sum_{j=-3\sigma}^{3\sigma} e^{-\frac{i^2}{2\sigma^2}}}, \ \sigma=11$$

3.1.2 Gap filter

Because the digital tablet samples points with a fixed time interval, the writing speed makes the distances between two points to be different. There would be some gaps in the stroke. These gaps would affect the curvature detection in later process. Fig. 3.2(a) is an example of writing with different speed. In order to compensate these gaps, we interpolate some new points between two adjacent points.

For each two adjacent points, let *dx* be the *x* difference between the two points, *dy* be the *y* difference between the two points. Then if *max*(*dx*,*dy*)>1, we interpolate *max*(*dx*,*dy*) points between them by linear interpolation. Fig. 3.2(b) shows the result of applying the process of the interpolation to Fig. 3.2(a).

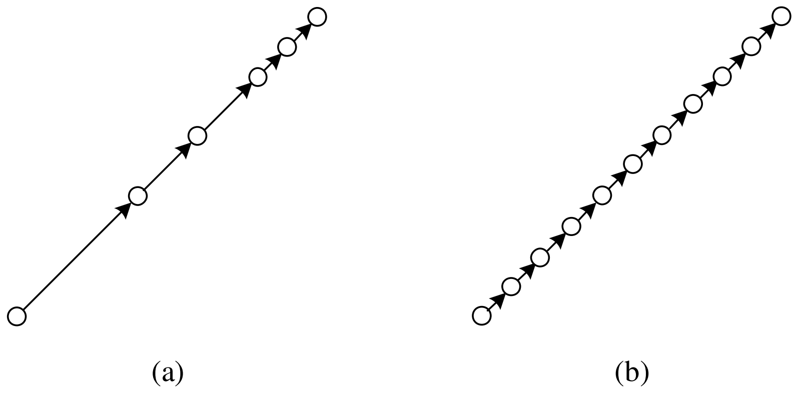


Fig. 3.2 An example of gap filter. (a) The original stroke with writing speed from fast to slow. (b) The stroke after the process of the interpolation.

3.1.3 Slipped segment remover

Slips are the action that user's pen move to the unexpected direction on the digital tablet. In the beginning and ending to write a stroke, it is easy to generate surplus slipped segments. The circles in Fig. 3.3 show the slipped segments of a stroke. We could remove slipped segments by detecting whether the length of the first segment or the last segment in a stroke is shorter than a given threshold.

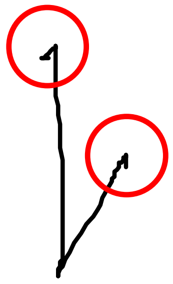


Fig. 3.3 An example of slipped segments in a stroke.

In order to eliminate the slipped segments, the first step is to find the candidates of slipped segments. Along a stroke, there usually exist some points with local extrema of curvature between the slipped segment and the others. To get these points, we first define dominant points as follows: (a) points corresponding to the local extrema of curvature; and (b) pen-up and pen-down points. Li and Hall proposed a method [13] to find dominant points in a stroke using a support region based on 8 ways chain codes which is shown in Fig. 3.4. Here, this method is adopted to find all dominant points in a stroke. Then we divide the stroke into several segments by dominant points. The first segment and last one are the candidates of the slipped segments. If the length of the candidate is less than a threshold, it is a slipped segment and would be removed. The threshold is set as half of the gap's height on staff in music score. Fig. 3.5 shows that there are five dominant points found in Fig. 3.3 and four segments are obtained. The first and the last segments are slipped ones.

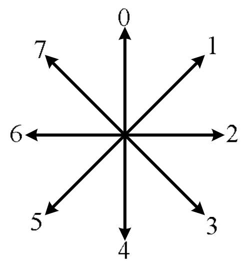


Fig. 3.4 8 ways chain codes.

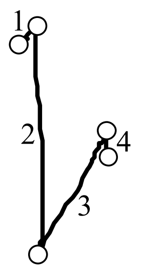


Fig. 3.5 An example of five dominant points in a stroke.

3.2 Simple symbol classifier

By observing the 17 kinds of symbols, we find that some symbols can be classified quickly by the extreme properties. We call these symbols as simple symbols, including Dot, the straight line of HLine, VLine, the straight line of Slash, the straight line of UHook and GClef. Here, we will discuss how to classify simple symbols.

Among all symbols, the length of GClef is longest obviously. By this property, we could easily recognize a stroke as a GClef symbol if the length of the stroke is longer than the length threshold. The length threshold is set as 12 times gap's height.

By observing the width and the height of a symbol, the Dot symbol has the smallest width and the smallest height in symbols. Therefore, the stroke would be recognized as a Dot symbol if the width and the height of the stroke are both shorter than a given threshold. The threshold is set as half of gap's height on staff.

To classify if a stroke is a straight line, a linearity measure is defined as

$$linearity = \frac{L}{G(P(s), P(e))}, \tag{2}$$

where L denotes the length of the stroke. $G()$ denotes Euclidean distance. $P(s)$ denotes the starting point of the stroke and $P(e)$ denotes the ending point of the stroke. Fig 3.6 illustrates an example of the linearity. If a stroke is a straight line, the linearity should approach to 1. Thus, if the linearity is smaller than the threshold, 1.07, we consider the stroke as a straight line and recognize it as HLine, VLine, Slash or UHook according to its slope. Once the stroke is recognized as a simple symbol, it would be output and exit the symbol recognition.

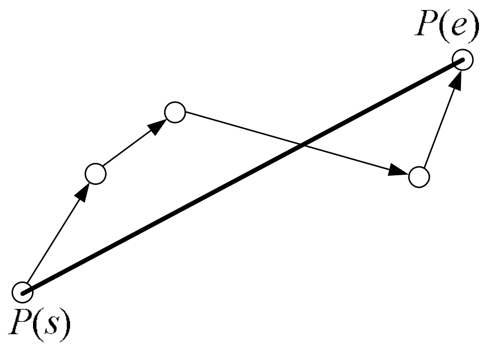


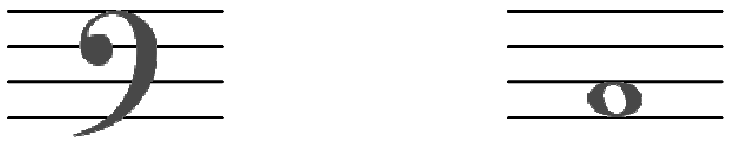
Fig. 3.6 An example of linearity of a stroke.

3.3 Feature extraction

If a stroke is not classified as a simple symbol, we will do feature extraction from the stroke. Here, we take three kinds of features: height, shape and direction.

3.3.1 Height

Notations in music theory have height limitation. Since notations are formed by symbols, symbols also have the height limitation. In Fig. 3.7, the height of FClefArc must be at least 2 times gap's height on staff, and the height of WHead must be less than 2 times gap's height. We could extract the height of a stroke as a feature for rough classification.



(a) (b)

Fig.3.7 Two examples of heights of symbols on staff. (a) FClefArc. (b) WHead.

3.3.2 Shape

As described in Section 3.1.3, each stroke will be divided into several segments. Every segment has its special shape. Fig. 3.8 shows that a stroke consists of two segments with a vertical line and a slash. The number of shapes would be useful for classifying stroke. There are 7 kinds of basic shapes shown in Table 3.1. These shapes are horizontal line, vertical line, slash, backslash, clockwise curve, counter-clockwise curve and circle. Here, the horizontal line, vertical line, slash and backslash are roughly called "straight line." The clockwise curve, counter-clockwise curve and circle are roughly called “curve.”

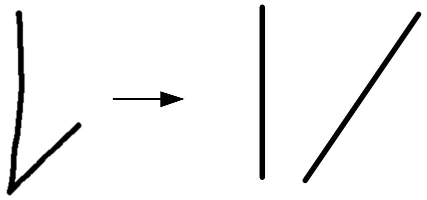
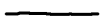
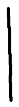
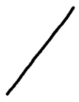
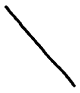
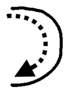
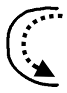
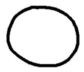


Fig. 3.8 An example for the shape feature of a stroke.

Table 3.1 The 7 basic shapes. (1) Horizontal line. (2) Vertical line. (3) Slash. (4) Backslash. (5) Clockwise curve. (6) Counter-clockwise curve. (7) Circle.

						
(1)	(2)	(3)	(4)	(5)	(6)	(7)

By the linearity and slopes mentioned in Section 3.2, we could determine if a segment is a horizontal line, vertical line, slash or backslash. If the segment does not belong to straight line, it may be a clockwise curve, a counter-clockwise curve, or a circle. Fig. 3.9 shows the clockwise curve, the counter-clockwise curve and the circle.

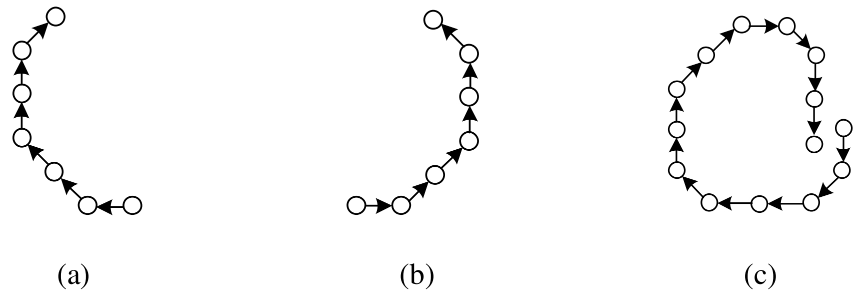


Fig. 3.9 Examples of curve segments. (a) Clockwise curve. (b) Counter-clockwise curve. (c) Circle.

The difference between the straight line and the curve is that the curve changes its direction very often. For each three adjacent points of a clockwise curve, the direction change from the previous point to the next point tends to be right direction. The direction change of the counter-clock curve tends to be left direction. The straight line has no direction change. Fig. 3.10 shows the direction changes in clockwise curve, counter-clockwise curve, and straight line. We could accumulate the direction change value to detect what kind of curve the segment is like. Based on the 8 way chain codes, the accumulated direction change value *v* is calculated by

$$v = \sum_{i=1}^{n} \mathrm{cost}(c(i), c(i-1)), \tag{3}$$

where *i* denotes the *i*th point in the segment. *n* denotes the number of points in the segment. *c*(*i*) denotes the chain code of the *i*th point in the segment. “cost” denotes the cost between two chain codes, which is shown in Table 3.2.

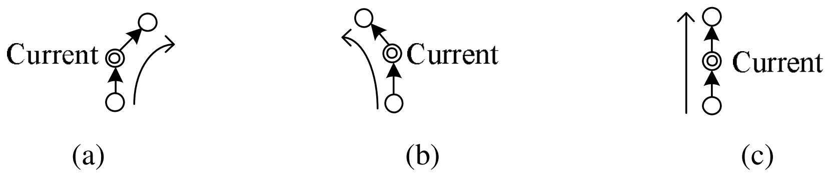


Fig. 3.10 Examples of the direction changes. (a) Clockwise curve. (b)Counter-clockwise curve. (c) Straight line.

Table 3.2 The cost between two chain codes.

c(i)\c(i-1)	0	1	2	3	4	5	6	7
0	0	-1	-1	-1	-1	1	1	1
1	1	0	-1	-1	-1	-1	1	1
2	1	1	0	-1	-1	-1	-1	1
3	1	1	1	0	-1	-1	-1	-1
4	1	1	1	1	0	-1	-1	-1
5	-1	1	1	1	1	0	-1	-1
6	-1	-1	1	1	1	1	0	-1
7	-1	-1	-1	1	1	1	1	0

In the cost table, the values are defined by 2 cases which are explained as follows:

1. If the current chain code is the same as the previous one, it means that they go toward the same direction and the direction change value is 0.

2. Otherwise, it means that the direction has been changed. We assign the direction change value +1 as they are toward the clockwise direction, and -1 as the counter-clockwise direction.

Then we classify each segment based on the accumulated direction change. If $v>0$, the segment is a clockwise curve. If $v<0$, it is a counter-clockwise curve. Furthermore, if $|v|>5$, it means that the curve changes its direction frequently and the segment is classified as a circle.

The last step is to calculate the number of every kind of shapes in the stroke. The vector of dimension 7 containing numbers of seven shapes is viewed as a shape feature.

3.3.3 Direction

The direction is the sequence of writing direction in time order, and it could reflect the writing style of a symbol. Fig. 3.11 shows the stroke with two writing directions, one is south and the other is east-north. The direction extracted from the stroke could help us clarify the difference among some symbols with similar shapes. For example, Fig. 3.12 shows two strokes with the same shape feature, but their direction features are different. We use the 8 way chain codes to represent the direction. We could extract the direction by eliminating the duplicate chain codes in the same direction. For example, if the chain code sequence of a stroke is (444444400111), the (4, 0, 1) is the direction feature. Note that the dimensions of the direction features of different strokes may be different.

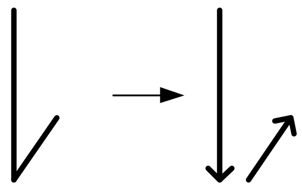


Fig. 3.11 An example of direction feature in a stroke.

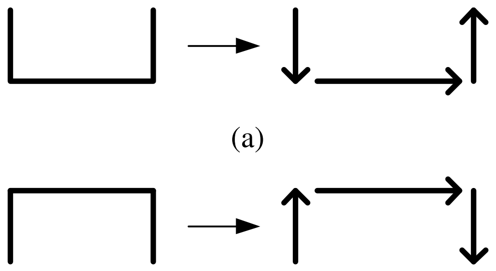


Fig. 3.12 An example of two strokes with the same shape feature, but different direction features. (a)WRest with direction feature (4, 2, 0). (b)HRest with direction feature (0, 2, 4).

3.4 Complex symbol classifier

Symbols except simple symbols are complex symbols. Features extracted from a stroke, including height, shape and direction, are taken for complex symbol matching at this phase.

Based on these three features, we construct three classifiers separately, including height classifier, shape classifier and direction classifier. Because some symbols in some classifiers are similar and are hard to separate them, we build a three level decision tree, which is shown in Fig. 3.13, to deal with this problem. The first level is the height classifier which roughly classifies symbols by the height feature. The second is the shape classifier which classifies symbols by the shape feature. The third level is the direction classifier which classifies symbols by the direction feature and outputs the recognized result.

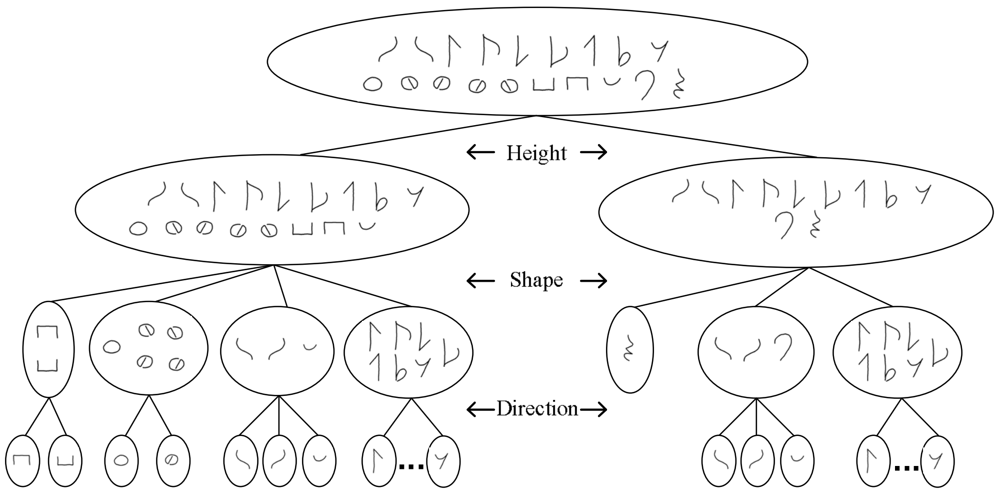
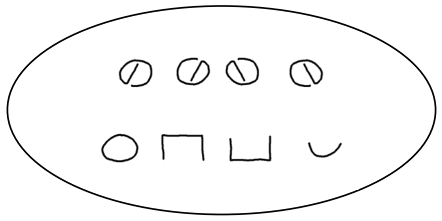
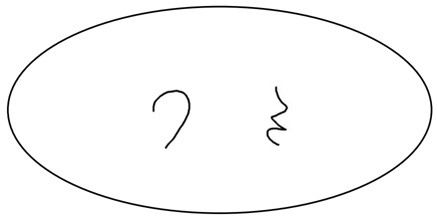


Fig. 3.13 A decision tree.

3.4.1 Height classifier

The heights of printed music notations are ruled by the music theory. Because the music notations are formed by symbols, symbols also have the height limitations in writing. By a height threshold, some symbols will be considered as high and some will be considered as low. However, due to the writing distortion, some symbols sometimes will be considered as in the high, sometimes low. We consider these symbols with unsure heights as variant. Fig. 3.14 shows the symbols in the low group, the high group, and the variant group.


(a) Low


(b) High

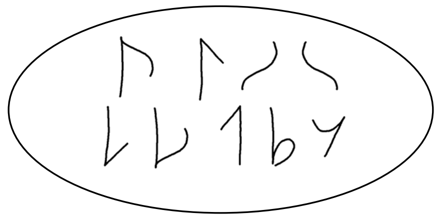
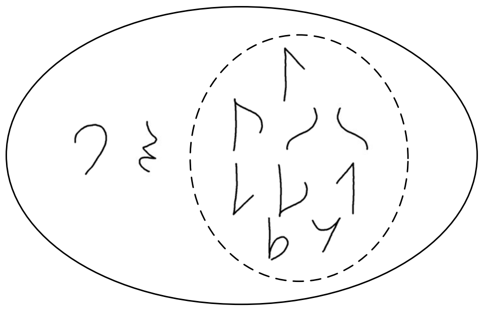

(c) Variant

Fig. 3.14 Three groups in height. (a) Low group. (b) High group. (c) Variant group.

In the height classifier, we use 2 times gap's height on staff as a threshold to classify symbols into the high group and the low group. Fig. 3.15(a) shows the low group in the height classifier. Fig 3.15(b) shows the high group in the height group.

The symbols surrounded by the dotted line and originally belonging to the variant group will be handled later.


(a) Low group

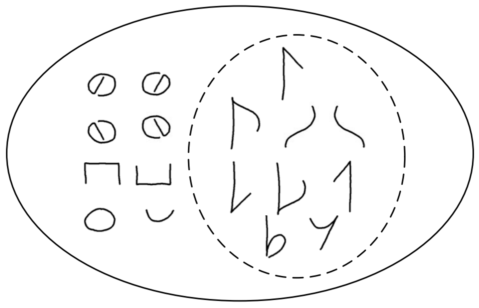

(b) High group

Fig. 3.15 Two groups in the height classifier. (a) Low group. (b) High group.

When a new stroke is coming, this classifier classifies the stroke to the high group or low group based on the height feature.

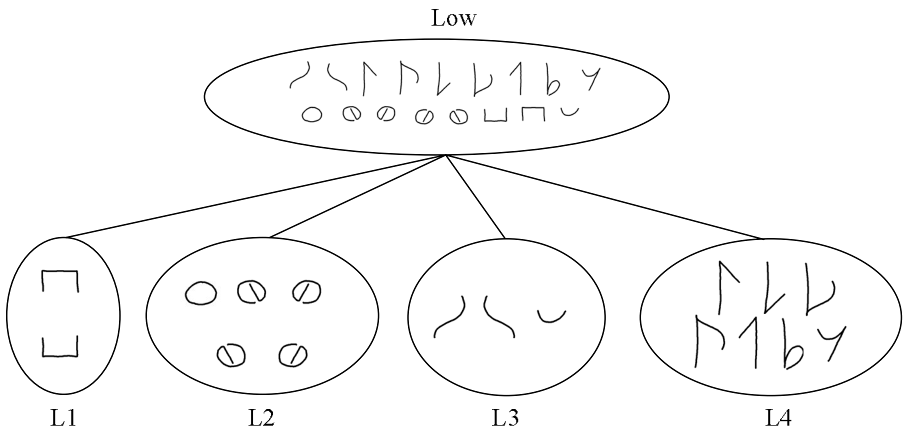
3.4.2 Shape classifier

In this stage, we group symbols with similar shape features. Fig. 3.16 shows the groups with similar shape features. The shape difference, *SD*, between two shape features is defined as follows:

$$SD = \sum_{i=1}^{7} \sqrt{S1(i)^2 - S2(i)^2}, \qquad (4)$$

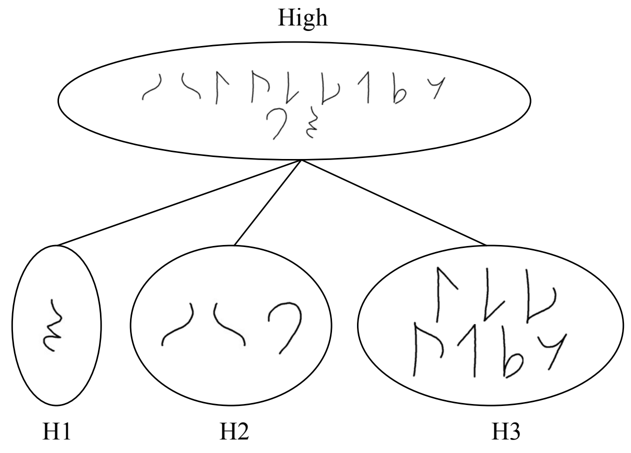
where *S1* is the vector of the shape feature 1. *S2* is the vector of the shape feature 2.

As a new stroke is coming, the classifier could measure the shape distance between the stroke and the shape templates in each group and applies KNN to find the group with the nearest distance. In H1 of Fig 3.16(b), there is only one possible symbol in the group, which will be output directly without further processing.



(a)

Fig. 3.16 Groups in the shape classifier. (a) For low group. (b) For high group. (*continued*).



(b)

Fig. 3.16 Groups in the shape classifier. (a) For low group. (b) For high group.

3.4.3 Direction classifier

In the third level of the decision tree, we would find most likely symbol according to the direction feature. We measure the distance between the direction feature of the stroke and the direction templates in database. Because the direction features are variable in dimension, the distance measure could be considered as the string matching problem. We apply the dynamic programming to obtain the distance.

Let $\{a_1, a_2,.., a_I\}$ denotes the direction feature and $\{b_{k1}, b_{k2},\ldots, b_{kJ}\}$ denotes the kth template in database. The accumulated distance $g_k(i,j)$ is calculated as follows:

Initial values:

$$\begin{cases} g_k(0,0) = 0 \\ g_k(i,0) = \infty \\ g_k(0,j) = \infty \end{cases}, \tag{5}$$

Recurrence formula:

$$\begin{aligned} g_k(i,j) = {} & \min\left(g_k(i-1,j-1), g_k(i-1,j), g_k(i,j-1)\right) \\ & + \text{differnece}(a_i, b_{kj}), \end{aligned} \tag{6}$$

where the difference is the chain code difference between chain code a_i and chain code b_{kj}. The difference is defined in Table 3.3.

Table 3.3 Difference between two chain codes.

$a_i \backslash b_{kj}$	0	1	2	3	4	5	6	7
0	0	1	2	3	4	3	2	1
1	1	0	1	2	3	4	3	2
2	2	1	0	1	2	3	4	3
3	3	2	1	0	1	2	3	4
4	4	3	2	1	0	1	2	3
5	3	4	3	2	1	0	1	2
6	2	3	4	3	2	1	0	1
7	1	2	3	4	3	2	1	0

Using above formula, the distance, $g_k(I,J)$, is calculated. After examining the distances with all templates, we find the symbol with the nearest distance and output the symbol as the recognized result.

3.5 Notation recognition

After a stroke is recognized as a symbol, the notation recognizer will be conducted by combining symbols. The output symbol will be combined with previously recognized and unused symbols to form a notation based on the semantic information. There are 3 levels in the notation recognition. In the bar level, those unused symbols would be combined to form a bar line. In the note level, those unused symbols would be combined to form a note. In the group level, some specific notes would be grouped together. The other part of notation recognition is modification operation. It provides the deleting, editing and moving operations which can be used in the three levels.

3.6.1 Bar level

In music theory, a bar is a container containing notes, and a bar line is used to separate bars. In our system, the bar would be constructed automatically to hint the user. We combine unused symbols to form a bar line in the bar level. In each bar, the pseudo borders of the bar are pre-drawn in our system. We define the head and the end of the bar as the reactive area for combining symbols separately. Fig 3.17 shows the reactive areas in a bar. By the shape of bar lines, we define the components in Table 3.4 to describe how to form a bar line. When a new coming symbol falls in one of these reactive areas, we would combine the new symbol and the existed symbols in the area and check whether the combined one matches the sets in Table 3.4. If yes, update the bar line; otherwise, the symbol would be sent to the note level.

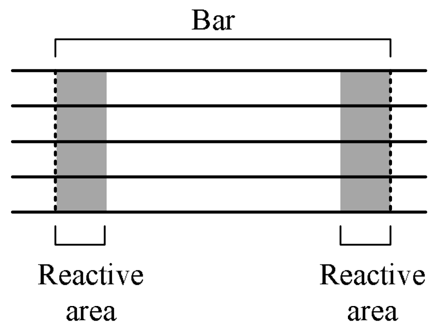


Fig. 3.17 The reactive areas in a bar.

Table 3.4 List of bar line with the set of components forming them.

Bar line name	Component
Single bar line (𝄀)	1 VLine
Double bar line (𝄁)	2 VLines
End bar line (𝄂)	3 VLines
Repeat sign line(𝄇)	1 Dot, 0 or more VLines

3.6.2 Note level

Notes are used to represent the relative duration and pitch of a sound in the music score. Symbols are combined to form a note in this level. By the composition of a note in music theory, there are three types of notes: determinable, uncertain and incomplete. Determinable note means that the numbers of symbols in it are fixed. The uncertain note means there are innumerable dots, heads, or flags in it. The incomplete note is a part of a certain note and recorded as a temporary note in this system. Fig. 3.18 shows that a sharp, which is a determinable note, has two VLines and two HLines exactly. Fig. 3.19 shows that a Note with filled head would be added by some hooks or dots. Fig. 3.20 shows an incomplete note which only has a filled head.

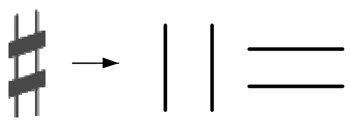


Fig. 3.18 An example of the determinable note.

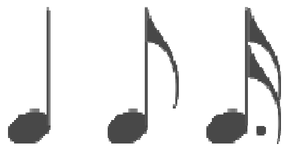


Fig. 3.19 Some examples of the uncertain notes.

Fig. 3.20 An example of the incomplete note.

When a new symbol is coming to this level, we would search the nearest uncertain or incomplete note. We do not have to search the determinable note, because it is impossible to add more symbols to it. If the distance to the nearest note is too large, we would construct a new empty incomplete note, and add the new symbol to it. Then check the symbols with rules in Table 3.5 [11], which consists of three cases as follows:

1. If we find a match in the table, then update the note and set its type.
2. If we could not find a match in the table, and the set of symbols is a subset of a note, then we add the symbol to the note and set its type to be incomplete.
3. If we could not find a match and the set of symbols is not a subset of a note, then the new symbol would be discarded.

Table 3.5 List of notes with the set of symbols forming them (*continued*).

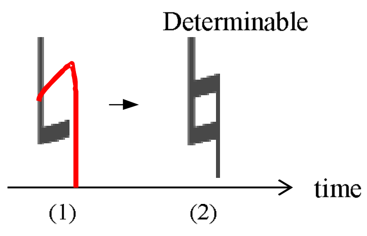
Note name	Type	Components
FClef (𝄢)	Determinable	2 Dots, 1 FClefArc
	Determinable	1 FClefArc
Sharp (♯)	Determinable	2 HLines, 2 VLines
	Determinable	2 Slashes, 2 VLines
	Determinable	1 HLine, 1 Slash, 2 VLines
	Determinable	2 UHooks, 2 VLines
	Determinable	1 HLine, 1 UHook, 2 VLines
	Determinable	1 Slash, 1 UHook, 2 VLines
GClef (𝄞)	Determinable	1 GClef
Natural (♮)	Determinable	1 LCheck, 1 NaturalRt
		1 LCheck, 1 8Rest
Flat (♭)	Determinable	1 Flat

Table 3.5 List of notes with the set of symbols forming them.

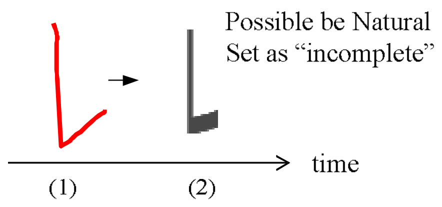
Note name	Type	Components
Whole note (e.g.,)	Uncertain	0 or more Dot(s), 1 or more WHead(s)
Half note (e.g.,)	Uncertain	0 or more Dot(s), 1 VLine, 1 or more WHead(s)
Note with filled head (e.g.,)	Uncertain	1 or more BHead(s), 0 or more Dot(s), 0 or more UHook(s), 1 VLine, 0 or more Slash(es)
	Uncertain	1 or more BHead(s), 0 or more Dot(s), 0 or more Slash(es), 1 VLine, 0 or more UHook (s)
	Uncertain	1 or more BHead(s), 0 or more Dot(s), 1 StUHook, 0 or more Uhook(s), 0 or more Slash(es)
	Uncertain	1 or more BHead(s), 0 or more Dot(s), 1 Lcheck, 0 or more Slash(es), 0 or more UHook(s)
Whole rest (e.g.,)	Uncertain	0 or more Dot(s), 1 WRest
Half rest (e.g.,)	Uncertain	0 or more Dot(s), 1 HRest
Eight rest (e.g.,)	Uncertain	1 8Rest, 0 or more Dot(s), 0 or more HLine(s)
Quarter rest (e.g.,)	Uncertain	0 or more Dot(s), 1 QRest

Fig. 3.21 shows the examples of the three cases. In Fig. 3.21(a), a NaturalRt is combined with an LCheck to form a Natural note which is determinable. In Fig. 3.21(b), a LCheck symbol is added to an empty note. Since it is a subset of Natural and may become a Natural in the future, the note is set as an incomplete note. In Fig.

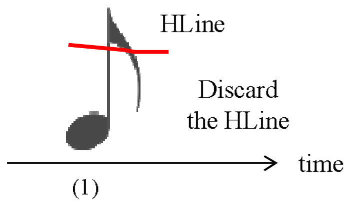
3.21(c), an HLine symbol is added to a Note with a filled head. This action is illegal and the set of symbols doesn't belong to any note. Since the HLine could not be used in the future, hence the HLine would be discarded.



(a)



(b)



(c)

Fig. 3.21 Some examples of combining symbols at the note level. (a) Case 1. (b) Case 2. (c) Case 3.

In order to keep the simplification of the system, when there are two incomplete notes existing in the meanwhile, we would only keep the latest note and delete the other. Fig 3.22 shows an example. At first, there is an incomplete note existed in the system. Then the user writes a new stroke, and the distance between the stroke and the incomplete note is too large. After the symbol recognition, the stroke is recognized as a new symbol and applies the notation recognition. In the note level, because of the large distance, the system constructs a new incomplete note containing the new symbol. At this time, there are two incomplete notes existing. We would consider that the user wants give up the old incomplete note and writes another new notation. Hence the old incomplete note would be deleted.

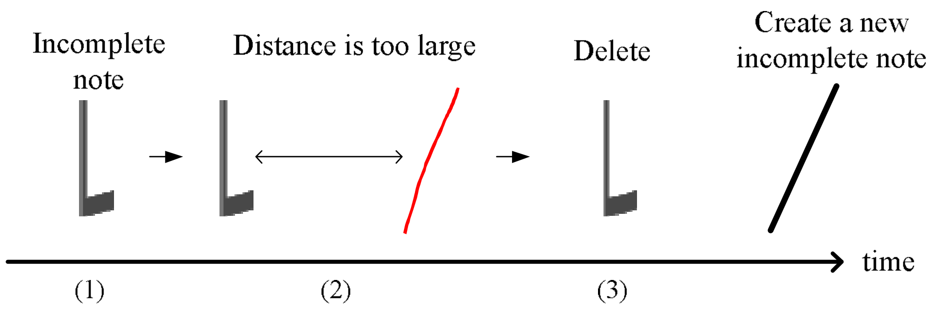


Fig. 3.22 An example of deleting an incomplete note.

For recording the pitch, some notes have to be located at the corresponding line or space on staff. These notes are Sharp, Natural, Flat, Whole note, Half note and Note with filled head. Fig. 3.23 shows some of these notes with pitch corresponding to line 2. We proposed a method to obtain the pitch of these notes. First, because the pitch on staff is related to the y-axis, we project all points of the note to the y-axis, and calculate the number of the projected points for each y-value. Fig. 3.24 shows a Sharp projected to y-axis. Then we apply the threshold which is defined for each note separately. The number of projected points on y which is below the threshold would be set to zero. Finally, we obtain an interval contains all non-zero values on y. The lines or spaces in the interval are the candidates for the pitch, and next we would determine the most possible one. We get the middle point, *my*, of the interval and define a window of size *t*, which is less than a half height of a space. If there is a line existing between *my-t* and *my+t*, the line is considered as the pitch of the note; otherwise, the space where *my* falls would be the pitch of the note. In Fig. 3.25(a), we could see that no line falls between *my-t* and *my+t*, so the pitch of the sharp is the space *i*. Fig. 3.25(b) shows another example that a line falls between *my-t* and *my+t*, and the pitch of the sharp is line *i*.

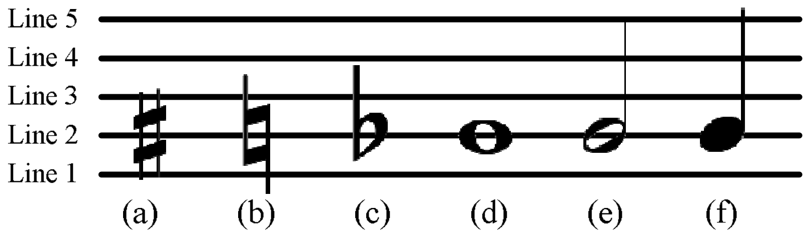


Fig. 3.23 Some examples for the pitches of notes corresponding to line 2. (a) Sharp. (b) Natural. (c) Flat. (d) Whole note. (e) Half note. (f) Note with filled head.

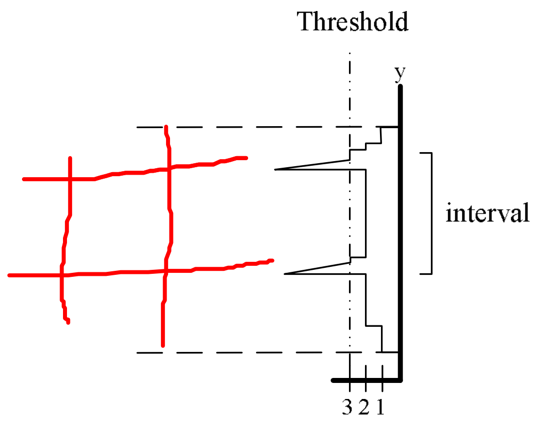
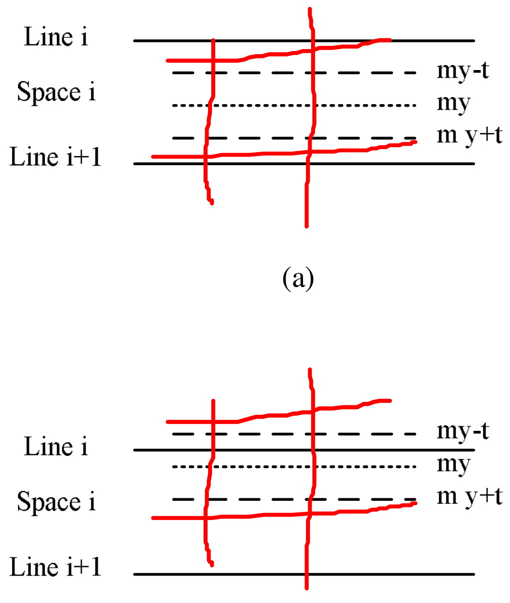


Fig. 3.24 An example of a sharp projected to y-axis.



(b)

Fig. 3.25 Some examples of pitch detection. (a) The pitch on the space. (b) The pitch on the line.

3.6.3 Group level

In music theory, when two or more notes with filled head and flags appear successively, we could group them using a beam to replace the flags. When playing the music score, the notes with beam should be more connected than non-beamed notes. As writing, users always draw a horizontal line across the notes to represent the grouping action. In the group level, we group the notes to form a beamed note. Steps for grouping are presented as follows:

1. We define the HLine symbol as the signal to group the notes by the common practice. If the new coming symbol is HLine, go to step 2; otherwise, reject the symbol.
2. Only the Note with filled head could be grouped in the music theory, and the accidental notes, like Sharp, Flat and Natural could be ignored. In the range where the HLine symbol covers along the x-axis, check whether the notes are valid.
3. The system would detect the time duration of each Note with filled head and draw the appropriate beam. In the meanwhile, all the flags on the notes would be removed.

If the grouped notes are quarter notes, we would consider the user forces to group the notes, and they would be converted to eighth notes automatically. Fig 3.26

shows an example which indicates how notes are grouped by the three steps.

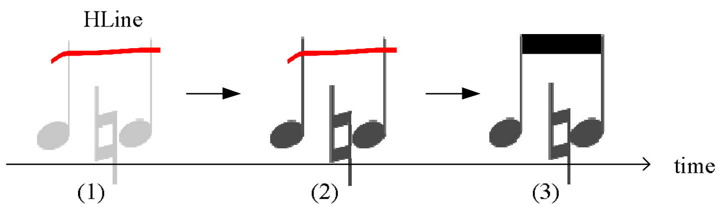


Fig. 3.26 An example of group note.

For those notes grouped, we also support to add more beams on these notes. By the common practice, the user only write a HLine on the notes where he would like to add a beam, the system would draw a new beam and update the time durations of these notes covered by the HLine. Fig. 3.27 shows an example of adding a new beam to a grouped note.

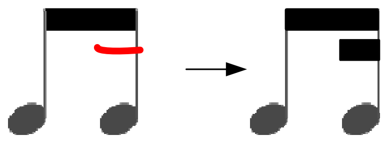


Fig. 3.27 An example of adding a beam to a grouped note.

3.6.4 Modification operation

In this stage, we introduce the modification operations for editing the music score. In the past, few systems provide the easy-using modification operations. They just use buttons as the modification operations on the screen for users to click. The input method is like the "point and click" based systems which is mentioned in previous chapter. For users, this is not intuitional at all. Instead of buttons, we take the advantage of pen based input method and provide some gestures for the modification operations.

We define two horizontal lines which are higher and lower than the music score, called "border lines." The border lines are the writing borders in the system. The area between two border lines is called "writing area," and the other areas are called "deleting area." Fig. 3.28 shows these lines and areas. Writing in the writing area is valid, or it is an illegal operation. The concept of the modification operations contains two points: (1) if we want to move the location or pitch of a note, we could drag parts of a notation or whole notation to the destination directly. (2) If we want to delete some parts of the notation or the whole one, just drag it to the deleting area.

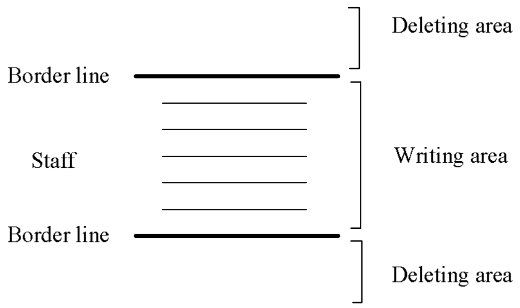


Fig. 3.28 The border line, writing area and deleting area in the system.

In Table 3.6, we list all the modification operations supported in the system and the details of these operations. The arrow line is the trajectory of the modification operation.

Table 3.6 List of the supported modification operation (*continued*).

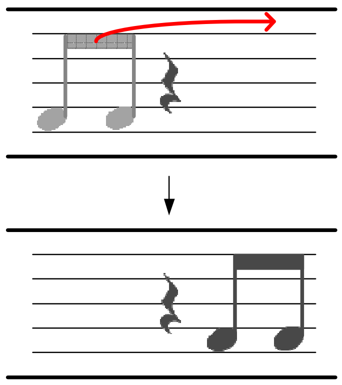
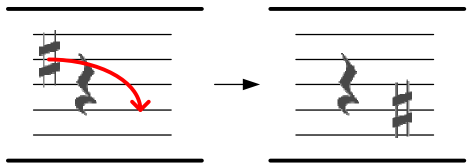
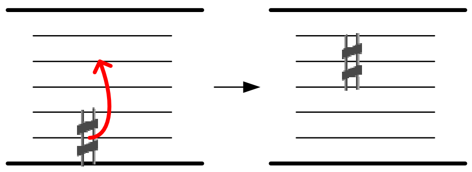
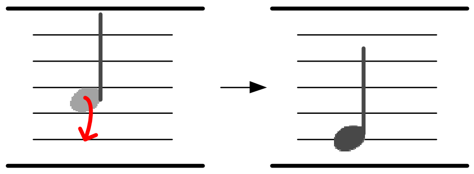
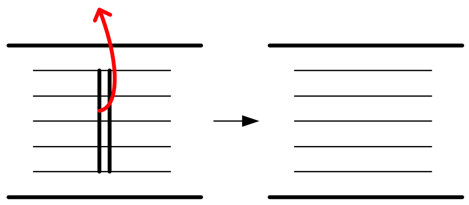
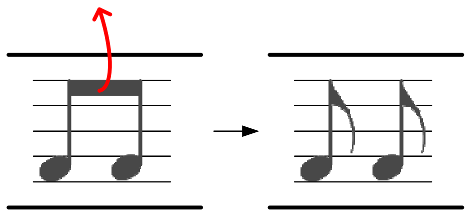
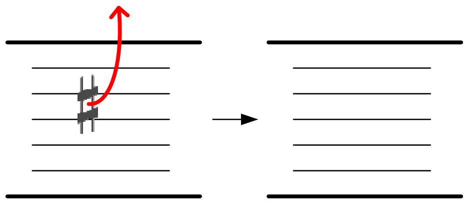
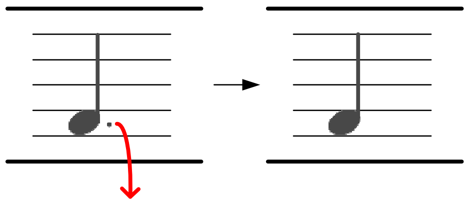
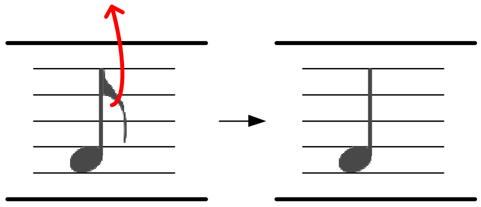
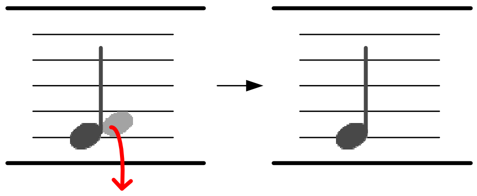
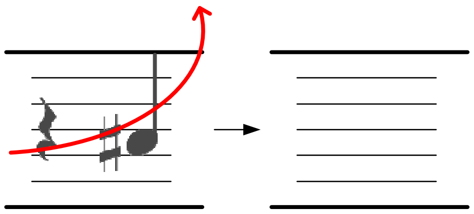
Modification operation	Example
Move a beam note: Drag the beam and move to the destination.	
Move a note: Drag the note and move to the destination.	
Modify the pitch of a note: Drag the note and move to the desired line or space.	
Modify the pitch of a note with heads: Drag the head of the note and move to the desired line or space.	
Delete a bar line: Drag the bar line to the deleting area.	
Separate a group of notes: Drag the beam to the deleting area.	

Table 3.6 List of the supported modification operation.

Delete a note: Drag the note to the deleting area.	
Delete a dot of a note: Drag the dot of the note to the deleting area.	
Delete a flag of a note: Drag the flag of the note to the deleting area.	
Delete a head of a note: Drag the head of the note to the deleting area.	
Delete some of notes: Draw a line across these notes which will be deleted and drag to the deleting area.	

3.5 Database reduction

The database consists of a lot of templates. To compare the feature with every template is very inefficient. If we could reduce the number of templates, the consuming time would be reduced. The problem is how to obtain the representative templates and discard the others. The clustering methods, like K-Means, are proposed to solve this problem. However, we do not know the distribution of the templates and could not assign the number of initial seeds exactly. So, we applied MBSAS (Modified Sequential Algorithmic Scheme) [14] to clustering templates which automatically determinates the number of initial seeds.

In our databases, we train the shape feature and the direction feature separately. The training process is applied to every symbol separately. Finally, for each symbol, we obtain the representative templates.

CHAPTER 4

EXPERIEMENT RESULT

Experiments are conducted to evaluate the performance of the proposed method. 13801 strokes, collected form 14 distinct writers, are used to test our algorithm. 6509 out of 13801 are taken as the training data. The remaining 7292 strokes are the testing data. Every stroke in the testing data is examined by symbol recognition. Finally, we could get the most similar symbol of the stroke as the output. In our experiments, a notebook (Intel T2300 CPU; only single cpu used; 1.66GHz; 1GB memory) and a digital tablet are used.

In order to measure the performance, we define the "precision" as follows:

$$\text{Precision} = \frac{\text{Correct}}{\text{Correct} + \text{Incorrect}}, \tag{7}$$

The precision for each symbol is shown in Table 4.1. The average precision for the symbols of our method is 98.35%, which is better than 97.54% of Miyao-Maruyama's method [11].

Table 4.1 Precision of each symbol.

Symbol name	Our method (%)	Miyao- Maruyama's method(%)
Dot	100.00	99.73
HLine	97.73	87.31
VLine	100.00	100.00
Slash	96.52	96.52
UHook	100.00	93.85
GClef	98.80	99.71
FClefArc	98.55	93.68
LCheck	99.71	90.81
NatureRt	97.87	100.00
Flat	98.69	100.00
WHead	98.46	97.49
BHead	96.70	99.85
StUHook	96.90	99.78
WRest	99.72	99.72
HRest	100.00	100.00
QRest	96.41	99.70
8Rest	95.88	100.00
Average	98.35	97.54

From the misclassified strokes, we find that the misclassification is due to that some users do not have any domain knowledge about the music theory, and they are not familiar with writing music notations. Sometimes they ignore the detail about the difference between symbols, like the curvature or the corners in a stroke. It makes some strokes ambiguous as trying to recognize. For example, if the user ignores the curvature between the slash and circle in BHead, the stroke is easily to be recognized as a WHead.

For the misclassified strokes, we provide the semantic correction to correct the mistakes. There are two rules defined in note level of notation recognition. First, while a WHead is misclassified to BHead and combine with a Half note, the system would convert BHead to WHead and do the combination. Second, while a BHead is misclassified to WHead and combine with Note with filled head, the system would convert WHead to BHead and do the combination. By the semantic correction, the precisions of WHead and BHead raise to 99.48% and 99.38%.

The total time of processing the 7292 testing data is about 157.38 seconds. Thus, the average processing time is about 0.0216 seconds per stroke. This is faster than Miyao-Maruyama's method which takes 0.0731 seconds per stroke by a PC (Pentium 4 CPU; 1.8GHz; 512MB memory). Thus, a user takes less waiting time while writing. Furthermore, our method is more suitable to migrate to the handheld devices with touched screen which have low computing power, and the user could compose a music score everywhere.

CHAPTER 5
CONCLUSION

The study proposed a method for recognizing music score by the properties of strokes. A stroke is recognized as a symbol and the symbol is combined with other symbols to form a music notation. Firstly, the preprocessing is applied to eliminate distortion in the stroke. Next, the stroke could be recognized as a simple symbol by the simple symbol classifier. If not, three feature extraction methods are performed on the stroke, and then the complex symbol classifier is applied. A decision tree with three classifiers is used to recognize the stroke to a complex symbol. Finally, the output symbol is combined with nearby symbols by rules in three levels and output a music notation. Both recognition rate and recognition speed of our method is better than those of existing method.

This system is robust enough for a general use. It provides all the common music notations and easy-using modification operations. Furthermore, music score playing function is supported for users to listen to the melody while they are editing the music score. Users are able to compose a complete music score by this system.

The future works are as follows:

- Multi-strokes input: in the proposed system, a stroke is considered as an input at a time. If the system supports the multi-strokes in a single input, the user could take less time waiting for the recognition.
- More symbols supported: The symbols are related to the writing styles of the notation. In this system, we support 17 kinds of symbols. The more symbols supported means the more ways to write a notation.
- More uncommon used notations supported: The common used music notations are supported, but some notations only used in a specific purpose are not included in the system, like C Clef ..., etc. In the future, these notations would be added to the system, and the system would be suitable for professional use.
- Semantic hint: When the music score is illegal to the music theory, the system would show hints to the user. It is very useful for the users who are not familiar with music theory.

REFERENCES

[1] J. Anstice, T. Bell, A. Cockburn and M. Setchell, "The Design of a Pen-Based Musical Input System," *In Proceedings of the 6th Australian Conference on Computer-Human Interaction* (OZCHI 1996), Hamilton, New Zealand, pp. 260-267, Nov. 1996.

[2] MagicScore Maestro software, DG software. (http://www.dgalaxy.net/)

[3] Allegro, finale software. (http://www.finalemusic.com/)

[4] S. E. George, "Online Pen-Based Recognition of Music Notation with Artificial Neural Networks," *Computer Music Journal*, vol. 27, no. 2, pp. 70-79, Jun. 2003.

[5] A. Forsberg, M. Dieterich, and R. Zeleznik, "The music notepad," *In Proceedings of the 11th annual ACM symposium on User interface software and technology*, San Francisco, CA, USA, pp. 203-210, Nov. 1998.

[6] Calligrapher, ParaGraph International, Inc. (http://www.paragraph.com/)

[7] D. Rubine, "Specifying Gestures by Example," *In Proceedings of ACM SIGGRAPH '91*, New York, USA, pp. 329-337, Jul. 1991.

[8] E. Ng, T. Bell and A. Cockburn, "Improvements to a Pen-Based Musical Input System," *OzCHI'98: The Australian Conference on Computer-Human Interaction*, Adelaide, South Australia, pp. 178-185, Dec. 1998.

[9] G. Taubman, "MusicHand: A Handwritten Music Recognition System,"*Honor*

thesis, Brown University, 2005.

[10] S. Macé, E. Anquetil and B. Coüasnon, “A generic method to design pen-based systems for structured document composition : Development of a musical score editor,” *In Proceedings of the 1st Workshop on Improving and Assessing Pen-Based Input Techniques*, Edinburgh, Scotland, pp. 15-22, Sep. 2005.

[11] H. Miyao and M. Maruyama, “An Online Handwritten Music Score Recognition System,” *In Proceedings of the 17th International Conference on Pattern Recognition* (ICPR 2004), Cambridge, United Kingdom, pp. 461-464, Aug. 2004.

[12] S. Connell and A.K. Jain, "Template-based Online Character Recognition," *Pattern Recognition* 34(1), pp. 1-13. 2001.

[13] X. Li and N. S. Hall, “Corner detection and shape classification of on-line handprinted Kanji strokes,” *Pattern Recognition* 26(9), pp. 1315-1334. 1993.

[14] S. Theodoridis and K. Koutroumbas, *Pattern recognition*, Academic Press. 2006.