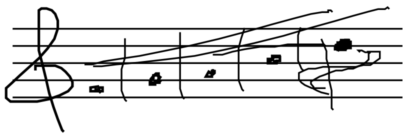
MusicHand: A Handwritten Music Recognition System

Gabriel Taubman [*] Brown University
Advisor: Odest Chadwicke Jenkins [†] Brown University
Reader: John F. Hughes [‡] Brown University



1 Introduction

Current methods of digitizing handwritten music for typesetting remain far from ideal despite many years for music typesetting programs to mature. There are essentially two categories of programs in this area. The first of these categories contains programs such as Sibelius [Sibelius 2005] which make no distinction between note pitch and note duration, forcing the user to enter both pitch and duration with the same mouse click. This method is time consuming and frustrating, especially if notes of differing durations must be entered often such as the rhythm in Figure 1.

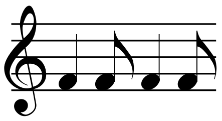


Figure 1: A difficult rhythm to enter in standard programs.

The second category consists of gesture based interfaces such as Music Notepad [Forsberg et al. 1998]. These programs require the user to learn a set of simple gestures which correspond to different musical symbols. After the user learns these gestures, the program is usually able to achieve a high recognition percentage. However such gestures are not ideal as they must be learned and practiced by the user. Learning gestures is an unnecessary task for music input because music notation is already itself a set of gestures.

MusicHand aims to achieve both the accuracy and ease of use of the gesture based programs, while not forcing the user to learn any new gestures. It does this by using an optical character recognition based approach [Parker 1997]. The system is trained with strokes of musical symbols and for each one a statistical moment is calculated. Upon entry of an unknown stroke, the database of moments is queried to find a closest match. Depending on what this entry is classified as, further processing may be done to find, for instance, its pitch in the case it is recognized to be a note. This method leads to a quick and easy method of inputting music in which users do not have to learn a set of gestures given they already know how to write music.

2 Musical Background

To properly discuss the choices made in the development of the system and to avoid confusion, a proper nomenclature for musical terms is first discussed in this section.

2.1 Staff

The height of a note on the page determines its pitch. Because people are unable to accurately judge relative heights over great distances, lines are drawn on the page to show the distinctions between heights. It is standard to draw 5 lines and they are called the *staff* or *stave*.

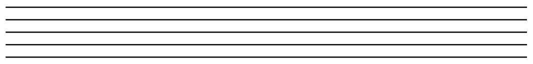


Figure 2: The musical staff.

2.2 Notes

Notes are a double encoding of pitch and the duration for which that pitch should be played. Note duration is indicated by the number of flags a note has. If a whole note is said to be played for one beat, then each successive note is played for half the duration. The english names of the notes are derived from this system.

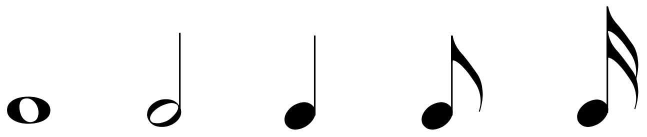


Figure 3: Notes of decreasing duration (left to right): Whole note, half note, quarter note, eighth note and sixteenth note.

Note pitch is heavily tied to the idea of clefs, as discussed in the next section.

2.3 Clefs

The staff alone does not indicate which lines correspond to which pitches. A *clef* is used to indicate which lines correspond to which pitches. Clefs are symbols placed at the beginning of the staff that define a pitch on a line. There are three types of clefs: G, C, and F, which show where G, C, and F lie on the staff respectively. The clefs their pitch-defining areas are shown in Figure 4. The G-clef defines a line to by G by circling it, the C-clef centers itself on the line to be designated C, and the F-clef places two dots on either side of the line to be considered F. Notes, much like G-clefs, distinguish their pitch by placing their round part on the line or space to which they are associated.

[*]e-mail: gtaubman@cs.brown.edu
[†]e-mail: cjenkins@cs.brown.edu
[‡]e-mail: jfh@cs.brown.edu

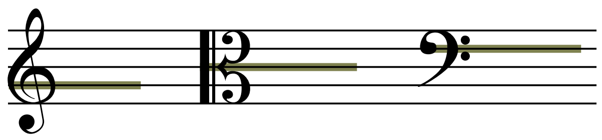


Figure 4: G, C and F clefs and the lines on which they define pitch.

2.4 Beams

The beaming of notes into groups is technically an unnecessary practice, as all rhythms can be expressed without the need for beams. An example of this is seen in Figure 5. However, beams provide the composer with a method of describing groupings of notes. These groupings are helpful during sight-reading[1] and especially at quick tempos. Also, beams provide insight into where stress should be placed in a group of notes.

Notes typically must be eighth notes or shorter to be beamed together. There are certain abbreviations for rhythms that involve the beaming of half and whole notes but this are beyond the scope of the program. The number of flags on a note denote how many beams it will have when beamed. An example of this can be seen in Figure 5 as well. Notes of varying durations can also be beamed together as can be seen in Figure 6.

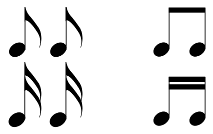


Figure 5: Two eighth notes and two sixteenth shown separated and beamed.

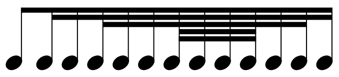


Figure 6: A set of notes with different durations all beamed together.

2.5 Barlines

Barlines provide separation of measures. These symbols are simply vertical lines that stretch the height of the staff. Because of their shape, they resemble the stems of notes. Special care must be taken to ensure that when the user enters a vertical line, it is recognized as the appropriate type.

3 Current Music Typesetting Programs

Sibelius and Finale [Finale 2005] are programs that enable the user to select the note duration from a palette, and then click the staff, inserting the selected note at the clicked pitch. These programs usually support input from a MIDI keyboard, but played rhythms must be quantized, making it very difficult for the program to distinguish between a quarter note followed by an eighth note all bracketed into a triplet, and a dotted quarter note followed by a sixteenth note.

Music Notepad [Forsberg et al. 1998] is a gestured based program for the input of music. While it is the first of its kind, it requires the user to learn a prescribed set of gestures that correspond to the different musical symbols.

4 Recognition Techniques

Recognition of the user's stroke input is achieved through the use of statistical moments [Shutler 2002]. These shape descriptors provide varying levels of invariance. Unfortunately the presentation of moments in existing literature tends to be variable so their implementation for this use will be discussed here.

4.1 Standard Moments

The standard equation for the moment of an image is:

$$M_{pq} = \sum_{x}^{X}\sum_{y}^{Y}(x^p y^q)\,P_{xy}$$

where $p,q \geq 0$, X and Y are the width and height of the image respectively, and P_{xy} is 0 when pixel x,y is white, and 1 when it is black. As can be seen from the binary requirement of P_{xy}, the image must usually be inverted and thresholded before its moments can be computed because the user is writing with black on white.

A fact often omitted from the moment literature is that these moments must be scaled back. If they are not, the higher order moments dwarf the lower order ones by many orders of magnitude. In the MusicHand system, all moments are divided by X^pY^q resulting in a roughly equal magnitude distribution of moments.

Since an infinite number of moments exist a subset must be chosen for recognition purposes. All moments from $p = 0, q = 0$ to $p = 3, q = 3$ inclusive were used. These standard moments change with image position and image scale. Because both of these parameters change often in handwritten music, moments that are position and scale invariant are desired.

4.2 Centralized Moments

Typically characters are recognized by humans to be invariant under translation. That is to say that the letter A is the letter A regardless of where it occurs on a page. The same is true for musical notes. Centralized moments provide such translation invariance and are calculated as:

$$\mu_{pq} = \sum_{x}^{X}\sum_{y}^{Y}(x-\bar{x})^p\,(y-\bar{y})^q\,P_{xy}$$

where

$$\bar{x} = \frac{M_{10}}{M_{00}} \quad \bar{y} = \frac{M_{01}}{M_{00}}$$

These moments are also scaled back by X^pY^q to ensure order of magnitude consistency. A fact about centralized moments used quite often in the MusicHand system is that $\bar{x} \times X$ and $\bar{y} \times Y$ give the centers of mass in the X and Y directions, relative to the $(0,0)$ point of the image. Using these provides a more accurate center for a stroke than the center of the stroke's bounding rectangle.

[1]Sight-reading is a term given to the playing of music that has not been practiced or rehearsed.

4.3 Normalized Moments

Normalized moments provide invariance under scaling. This scale invariance enables both a large G clef and a small G clef to be recognized as a G clef. Normalized moments are calculated as follows:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}}$$

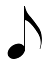
with

$$\gamma = \frac{p+q}{2} \text{ where } p+q \geq 2$$

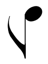
The normalized moments are what is used for recognition in the MusicHand system. Hu moments as described in the next section provide further invariance that unfortunately proves problematic for musical symbol recognition.

4.4 Hu Moments

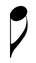
The seven Hu moments described as follows provide invariance under rotation and with the addition of I_7, invariance to skew as well. Rotational invariance, while providing users with the flexibility to write their characters slightly rotated, would permit the recognition of illegal strokes. For instance an eighth note drawn correctly with the stem up looks like:



however, when rotated 180 degrees looks like:



which does not exist in musical notation. The correct drawing of an eighth note with its stem going down is:



It is for reasons such as this that the Hu moments are not used for recognition. The Hu moments can be calculated by:

$$\begin{aligned}
I_1 &= \eta_{20}+\eta_{02} \\
I_2 &= (\eta_{20}-\eta_{02})^2+4\eta_{11}^2 \\
I_3 &= (\eta_{30}-3\eta_{12})^2+(3\eta_{21}-\eta_{03})^2 \\
I_4 &= (\eta_{30}+\eta_{12})^2+(\eta_{21}+\eta_{03})^2 \\
I_5 &= (\eta_{30}-3\eta_{12})(\eta_{30}+\eta_{12})\left[(\eta_{30}+\eta_{12})^2-3(\eta_{21}+\eta_{03})^2\right]+ \\
&\quad (3\eta_{21}-\eta_{03})(\eta_{21}+\eta_{03})\left[3(\eta_{30}+\eta_{12})^2-(\eta_{21}+\eta_{03})^2\right] \\
I_6 &= (\eta_{20}\eta_{02})\times \\
&\quad \left[(\eta_{30}+\eta_{12})^2-(\eta_{21}+\eta_{03})^2+4\eta_{11}(\eta_{30}+\eta_{12})(\eta_{21}+\eta_{03})\right] \\
I_7 &= (3\eta_{21}-\eta_{03})(\eta_{30}+\eta_{12})\left[(\eta_{30}+\eta_{12})^2-3(\eta_{21}+\eta_{03})^2\right]+ \\
&\quad (\eta_{30}-3\eta_{12})(\eta_{21}+\eta_{03})\left[3(\eta_{30}+\eta_{12})^2-(\eta_{21}+\eta_{03})^2\right]
\end{aligned}$$

4.5 Recognition from Moments

To determine which musical character most closely resembles the user's stroke, the system computes the moments of the unrecognized stroke and queries them against the database of normalized moments. The closest match is found using the *n*-nearest neighbor technique which involves computing the Euclidean distance from the unknown moment to all others in the database. Then, the *n* nearest moments are examined and whichever musical stroke is associated with the majority of those moments is decided to be the stroke entered by the user. An example of this situation can be seen in Figure 7. For a further discussion of recognition from moments see Section 9.3.

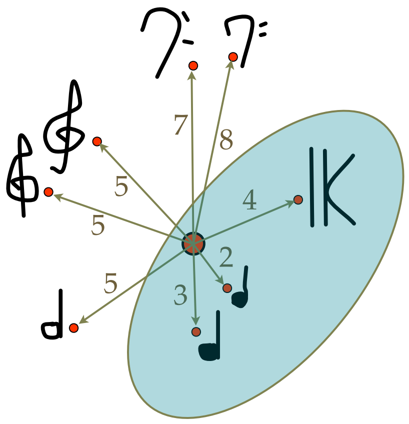


Figure 7: An illustration of *n*-nearest neighbor techniques. The unrecognized entry's moment is represented by the red dot at the center. Euclidean distances to all other moments are shown. The 3 nearest neighbors are circled. This situation would result in the unknown entry being classified as a quarter note.

5 Program Use

5.1 Training

The training step of MusicHand is to provide the system with the database of moments to be used for recognition. Training does not necessarily have to be done by every user. Assuming that the current user's music handwriting resembles to some degree[2] another user's who has trained the system, the other user's database of moments can be loaded to be recognized from.

In the current implementation of the MusicHand system, training is achieved by selecting the preferred symbol to train from a pull-down menu. After the stroke is drawn, either a button marked "Add Moment" or the return key are pressed to compute the moments of the stroke and add them to the database. When the user is finished training, the pull-down menu is set to "none" which signals to the system that it should attempt to recognize any inputted strokes.

5.2 Input

Initially the user is presented with a set of empty staves on which to write music. The number of parts can be changed and parts are group into systems. Inter-staff and inter-system spacing is user configurable. Input strokes are currently given to the system through by drawing on a Wacom[3] tablet.

[2]See Section 9.1.2 for more discussion of interoperability of moment sets between users.

[3]http://www.wacom.com

At this point pen timing because an issue to contend with. Pen timing is the delay between when the user picks up the pen and when the system begins to process their input. Ideally, the instant the user picks up their pen the system would begin processing. However, this is not realistic as many musical symbols require more than one stroke to complete, such as a C-clef. If the user does not place the pen down again after a set time has elapsed, the system recognizes the stroke and places it on the staff in an appropriate location. For more information on pen timing, see Section 7.

6 Recognition Process

The full recognition process is split over many sections through the use of inheritance. This entire process is best viewed as a flowchart seen in Figure 8. The following sections will document the recognition processes on a smaller scale.

6.1 Note Recognition

The following is an example of the chain of events that occur when the user draws a quarter note with a stem up.

1. The user draws a note.
2. The standard moments, centralized moments, and finally normalized moments are computed for the user's stroke.
3. The database of normalized moments is queried to find the closest match using *n*-nearest neighbor techniques as previously described in Section 4.5. Since the user drew a quarter note the closest moment recognized should also be a quarter note.
4. The pitch of the note must be computed. This step is achieved through by splitting the user's stroke into all sub-strokes. Sub-strokes are delineated by breaks in the user's stroke path. Moments are computed for each sub-stroke and each one is queried against the moment database individually. The first sub-stroke to be recognized as either a closed dot or a whole note is considered to be the round part of the note by which the note will be centered.
5. Now that the round part or "dot" of the note has been located, its center of mass is computed. This center of mass is then snapped to the nearest staff line or staff space.
6. Finally the stem direction of the note must be computed. First, the center of mass of the entire user's stroke is computed. This position is then compared to the center of mass of the dot alone. If the dot is concluded to be below the center of mass of the rest of the stroke, the note is decided to have an up stem. If the dot is above the center of the stroke, a down stem is given to the note.

6.2 Multiple Dot Entry

One of the main advantages of the MusicHand system is that it allows the user the ability to enter multiple dots (filled or open) and add stems and flags at a later time. This freedom results in an input method that is more akin to the natural way of writing music by hand. It is easy to enter many sequential notes of different durations like those seen in Figure 1 with this method.

The entry of a vertical line can be quite ambiguous as can be seen in Figure 9. In this example the user may have intended a stem going up on the left note, or a stem going down on the right note, or a bar line in between.

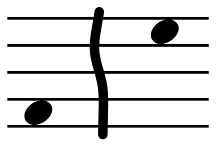


Figure 9: An ambiguous entry

The following method is employed to determine the user's intentions:

1. The stroke is recognized to be a vertical line.
2. Compute the center of mass of the line.
3. Find the entries to the immediate left and to the right of the x position of the center of mass of the line. If neither a left hand nor right hand entry exist, the stroke is recognized as a bar line.
4. First the left hand note is examined. The reason for this is that in practice people tend to draw stems on notes from left to right, therefore there exists a higher chance of their intention being a stem on the left note rather than right note. First, the horizontal distance between the line and the dot is computed. If it is not less than a set threshold (set to 30 pixels in the system) the system moves on to the next step. If however it is less than the threshold, then the note itself is inspected. If it is either a whole note, or a closed dot the system continues analyzing. Finally, if the center of the vertical line in the y direction minus the center of the dot in the y direction yields a positive number, then the line is considered to be an up stem on the left hand note and the recognition process is finished. If any of these requirements are not fulfilled however, the system moves to the next step.
5. The system examines the right hand note. The same recognition steps take place with slight changes. The system ensures that the y center of the vertical line must be *below* the note's center this time. This is because a stem going up can not exist on the left side of a note. If any of the requirements are not fulfilled, the user's stroke assumed to be a bar line.

6.3 Beams

As previously mentioned beams technically provide no extra flexibility in the ability to notate rhythms. However, they are a necessity in the reading of music. Beam recognition begins at the beginning of the recognition chain in Figure 8.

The user's stroke is decided to be a beaming stroke without actually querying the moment database. This classification is achieved by defining a beaming stroke to be any stroke that touches at least two notes. Recognizing a beaming stroke through the use of statistical moments would be difficult because a moment for a line at every possible rotation would need to be entered into the database. This is an unrealistic need and the use of context-based recognition performs much better.

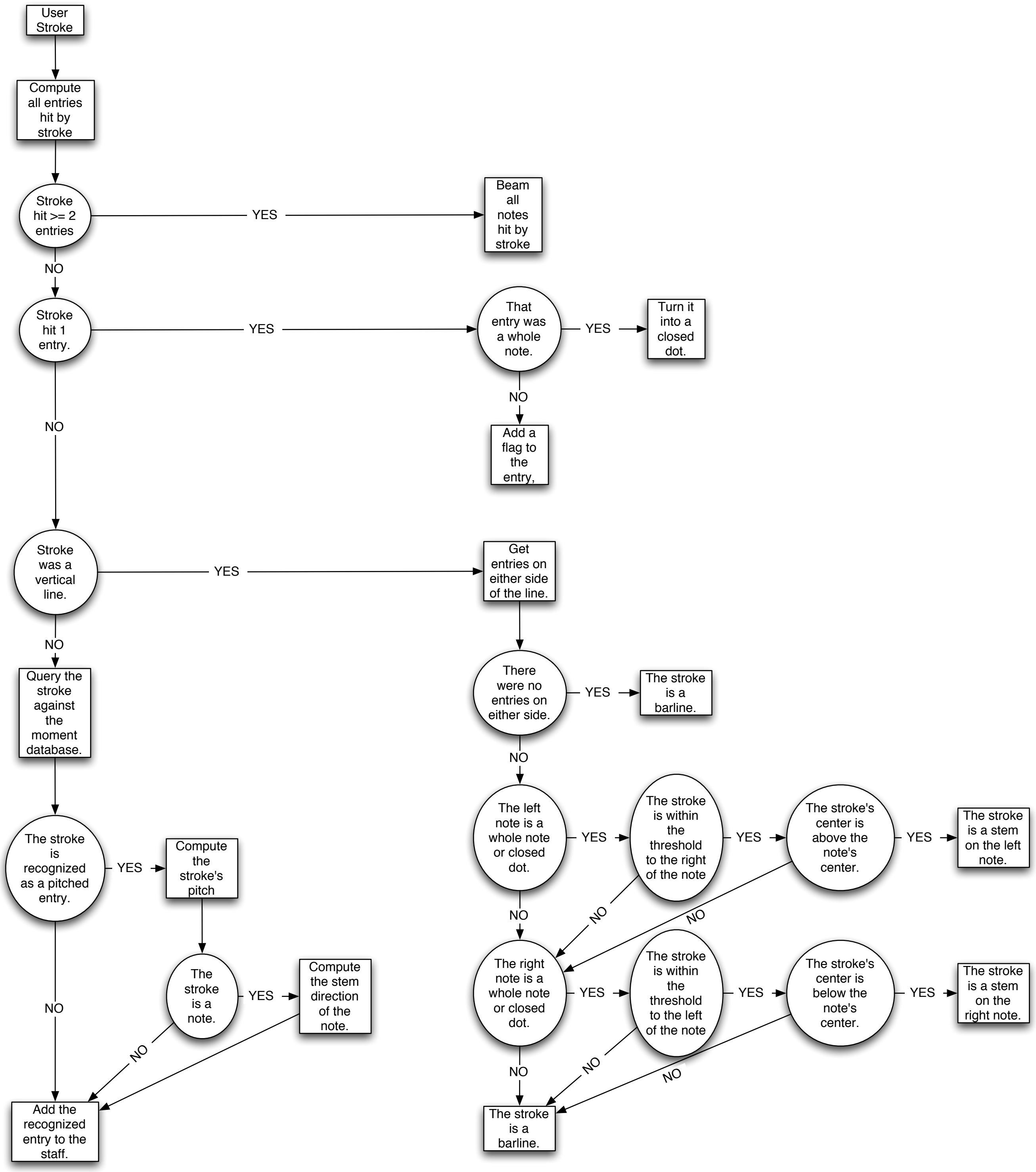

User Stroke
Compute all entries hit by stroke
Stroke hit >= 2 entries
YES
Beam all notes hit by stroke
NO
Stroke hit 1 entry.
YES
That entry was a whole note.
YES
Turn it into a closed dot.
NO
Add a flag to the entry,
NO
Stroke was a vertical line.
YES
Get entries on either side of the line.
There were no entries on either side.
YES
The stroke is a barline.
NO
The left note is a whole note or closed dot.
YES
The stroke is within the threshold to the right of the note
YES
The stroke's center is above the note's center.
YES
The stroke is a stem on the left note.
NO
NO
NO
The right note is a whole note or closed dot.
YES
The stroke is within the threshold to the left of the note
YES
The stroke's center is below the note's center.
YES
The stroke is a stem on the right note.
NO
NO
NO
The stroke is a barline.
NO
Query the stroke against the moment database.
The stroke is recognized as a pitched entry.
YES
Compute the stroke's pitch
The stroke is a note.
YES
Compute the stem direction of the note.
NO
NO
Add the recognized entry to the staff.

Figure 8: A flowchart of the recognition process. Circles are if statements

6.3.1 Beam Method

Assuming that rhythms can be inputted with complete accuracy and ease, the system still needs a method by which to draw them. As said before these rhythms can be either flagged or beamed. To transform from flag to beam representation, the following algorithm is applied:

```
for (int d = 8; d <= MAX_D; d *= 2){
   for (all notes i under the beam){
      if (note[i].d >= d){
         if (note[i].d == d && note[i].b){
             if (i == 0){
                  halfbeam right
             }
             else{
                  halfbeam left
             }
             continue
         }
         if (note[i+1].d >= d){
             fullbeam from note[i] to note[i+1]
             continue
         }
      }
   }
}
```

where `d` is the current duration and the member variable `d` is the duration of that note. Note duration is represented as the reciprocal of its beat length. For instance a quarter-note has duration 4 instead of $1/4$. `b` is a boolean specifying whether or not the note is a broken. A note becomes broken in one of two situations. The first is a note whose beam should connect to the next note, but due to rhythmic considerations such as time signature or note stresses, the beam is split. The second is a note of duration smaller than both of the adjacent notes. An example of this can be seen in Figure 10, and standard beams can be seen in Figure 11.

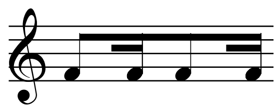


Figure 10: An example of beaming with broken beams.

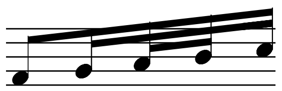


Figure 11: An example of a complicated beam.

The above algorithm for drawing beams is not ideal. Broken note's beams are assumed to go left unless the note is the first one in which case it goes right. This is an incorrect assumption because broken beans can go in either direction. However, as a preliminary reverse engineering of proper beaming rules it suffices.

6.4 Flags

Like beams, flags are also recognized through context based recognition without a moment query. This classification is achieved by examining the list of all notes hit by the user's stroke. If this list contains only one item, and it is a note, it is assumed that the user intended to draw a flag onto a note. There are two exceptions to this rule however which are when the note that the user drew over is a whole note or a closed dot. Neither of these glyphs have stems which leaves the user nothing to draw a flag onto thus it is assumed that drawing a flag was not the user's intent.

For a closed dot, it is assumed that the user was attempting to add a stem to the note to turn it into a quarter note. This adds extra strength to the programs ability to recognize stems.

For a whole note, it is assumed that the whole note was recognized in error and that the user is attempting to fill it in. This assumption provides the user with a quick method of turning whole notes into closed dots. Since the moments for the two symbols are similar, this extra ability to correct the system is a welcome addition.

If any other type of note is hit, it simply has its duration decreased to the next type of note. For instance, a half note turns into a quarter note, and a quarter into an eighth and so on. An example of the addition of a flag to a note can be seen in Figure 12.

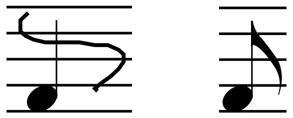


Figure 12: A quarter note during and after the addition of a flag.

7 Pen Timings

The subject of pen timing as previously mentioned is of great consequence in the MusicHand system. The question to ask is, "How long should the system wait after the user has picked up their pen before attempting to recognize their stroke." If musical symbols were all one stroke, then the obvious answer would be to have the system recognize the input as soon as the pen is lifted. However, many symbols for instance notes, clefs, and accidentals are often written in more than one stroke. It is for this reason that the user must be given time to put the pen down and continue writing before recognition takes place.

A recognition delay of 0.6 seconds is set as default. However, as the user begins to feel comfortable with the tablet they draw on, the delay can be lowered to 0.3 seconds before the input of multiple stroke symbols becomes infeasible.

Certain shortcuts can be taken however, for instance when the pen begins its path within the bounding rectangle of a note. If this is the case then the only possible user intentions are adding a flag to the note or joining that note with other notes into a beam. Both of these situations only require one stroke so the wait between lifting the pen and performing the recognition operation can be set very small.

However it was noticed that setting the delay to zero seconds resulted in a surprisingly jarring behavior. It became difficult to distinguish what had just been written. For this reason, if the pen begins within the bounding rectangle of a note, the recognition delay is set to 0.2 seconds.

8 Results

8.1 Implementation

The current implementation of the MusicHand system is written in Objective-C to run under OS X. The implementation of Objective-C by Apple[4] provides libraries which facilitate the import of musical character fonts which are used to display a printed version of musical symbols after recognition has taken place. The font used in this implementation is called “Fughetta” and was created by Blake Hodgetts[5]. The Apple supplied libraries also greatly ease transition from vector to raster image formats.

No extra peripherals are required for the system. However, the use of a Wacom tablet greatly enhances and facilitates the experience of using the system.

While the current system is functional, it is still a proof of concept and thus has its shortcomings. Section 9 discusses further steps that can be taken to create a more robust system.

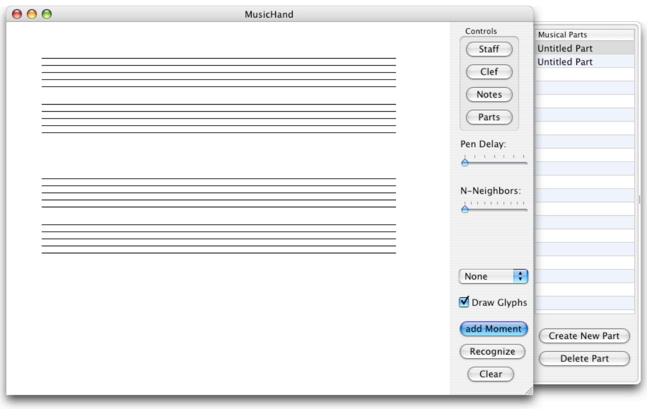
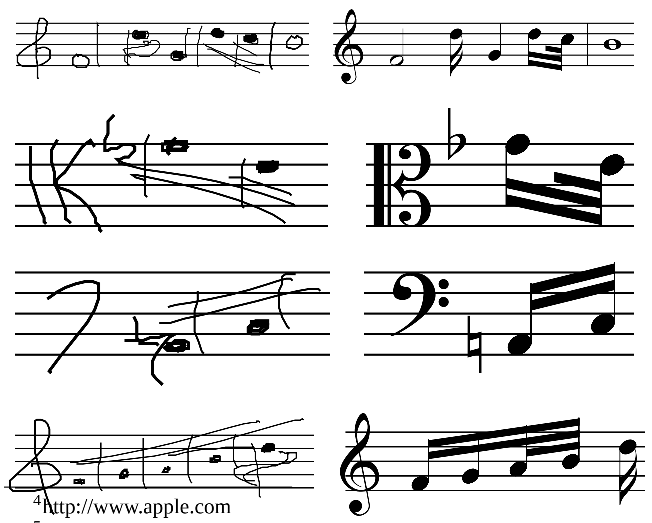


Figure 13: The current implementation of the MusicHand system.

8.2 Examples

The following are a series of examples of handwritten input and the system’s corresponding recognized and typeset output.



[4]http://www.apple.com
[5]http://www.efn.org/ bch/AboutFonts.html

9 Future Work

9.1 User Studies

The MusicHand system could benefit greatly from a series of user studies. The following are areas in which studies are most needed.

9.1.1 System Effectiveness

Primarily, a study needs to be done to determine whether or not the MusicHand system effectively accomplishes its goal as a compositional aid, and as an alternate method of inputting music to a computer. A possible test is to ask users to copy an excerpt of printed music with pencil and paper, and with the MusicHand system. Timing both tasks will yield information as to whether or not the system is robust enough to accommodate user error and writing style as well as a pencil and eraser do.

Another possible test is to ask current composers to attempt to use the MusicHand system instead of their traditional means of writing music. The use of a Wacom Cintiq tablet would be more fitting for this test because it provides the user with essentially an electronic version of the pencil and paper input device they are already used to. This is opposed to the standard Wacom tablets which require time before their use feels natural.

9.1.2 Moment Compatibility

A study of how well one user’s library of moments can be used by another would be of great importance. If the case is that twenty people’s combined training data is sufficient for a high percentage of user’s strokes to be recognized then individual training would only be an improvement rather than a requirement. If this fact is true it would present an ideal situation where the average user is not required to provide the technical training steps.

9.1.3 N-Nearest Neighbor

The value of *n* in the *n*-nearest neighbor system changes recognition results greatly. Finding an optimal value for *n* based on the number of moments in the database would greatly improve the recognition percentage.

9.2 Pen Timing

Currently pen timing is set to a default value. A more comprehensive method of learning pen timing on a per-user basis would be to compute initial pen timing for a user by example. The user can be asked to perform a series of note entries, where between each entry the user hits the space bar as soon as they’ve finished. This gives the system an initial idea of how long to give the user to create multi-stroke input.

9.3 Recognition

There exist many expansions to the current recognition scheme for instance a probabilistic or Bayesian method. A natural extension however would be computing inverse-covariance matrices and means for every type of symbol. This method allows each moment to have its components scaled appropriately given the distribution of all moments corresponding to the same musical symbol which will provide a more accurate measure of distance between unknown and known moments.

Another possible addition is the ability for the program to search for patterns in music already entered on a per-piece basis. These patterns can then be used to provide a form of auto-completion of commonly entered rhythms and motives. The usefulness of this feature may vary however, depending on the style of music being entered.

9.4 Semantics

The current incarnation of the MusicHand system attempts to allow the user to enter any symbol at any place on the score. However, incorporating a semantic checking system beyond what is used to recognize the addition of stems onto note heads may provide further benefits to recognition abilities. For example if the user draws a flat, an entry entered directly to the right and centered on the same line or space will most likely be a note of some kind. This form of semantic information could greatly expand the system's ability to recognize accurately. However, it would add the sense of rigidity present in programs such as Sibelius that make them potentially undesirable for composition.

9.5 Beams

The current rules for creating beams are simple and somewhat crude. Investigation into how beams are typically handled by human typesetters could provide the system with greater ability to produce pleasing looking beams.

Also, the current system is unable to conjoin sets of already beamed notes. Small functionalities like this, while not necessary, would provide the user with a more pleasant experience.

9.6 User Interface

9.6.1 User Mistake Correction

Currently if the user wants to remove a recognized symbol, the "E" key is pressed which switches the system to erase mode. While in erase mode the drawing color is set to red as opposed to black. Any typeset symbol can be drawn over with red to be erased. However, the use of the Wacom pen's built in eraser would be more desirable because it would alleviate the need for the keyboard entirely.

9.6.2 Recognition Mistake Correction

The *n*-nearest neighbor recognition technique provides feedback about all symbols that resemble an unknown one, and the percentage the system is confident in its classification choice. If a user's stroke is recognized incorrectly, the ability to have the user then tap on that symbol to bring up a menu of other symbols to change it to, ordered by recognition confidence percentage, would be a highly intuitive and quick method of telling the system that a misclassification took place. Also, the system currently does not add recognized input to the database of moments. However, if the user tells the system through the use of this menu that the input was recognized incorrectly, then that stroke could be added to the database to lessen the chance of future misclassifications.

9.6.3 Note Layout

Currently the system enforces no semantic rules of music such as the number of beats per measure. A possible extension would be once a measure has been completed to have the system then run a relaxation to space the notes in that measure appropriately regardless of their original locations as entered by the user.

9.6.4 User Trainable Macros

Because arbitrary strokes can be recognized using statistical moments, the option exists for the user to train the system to recognize any unused stroke to be any set of symbols. For instance the system could be trained to recognize a wavy horizontal line to mean four eighth notes of the pitch the line is centered on, all beamed together. These sorts of user definable macros could greatly increase input speed.

10 Conclusion

The goal of the curent MusicHand system is to prove that handwritten music recognition is a viable method for music input to computers. In this regard it is a success. The examples shown in Section 8.2 were written without hesitation or coaxing of the system. However, the current system is not robust enough for general use. With the further enhancements documented in Section 9 it may provide an intuitive and possibly more enjoyable music input interface than any system currently available.

References

FINALE. 2005. *Finale 2005*. MakeMusic! Incorporated. http://www.finalemusic.com.

FORSBERG, A., HOLDEN, L., MILLER, T., AND ZELEZNIK, R. 1998. Music notepad. In *Proceedings of the 11th annual ACM symposium on User interface software and technology.*, ACM, 203–210. http://graphics.cs.brown.edu/research/music/tpc.html.

PARKER, J. R. 1997. *Algorithms for Image Processing and Computer Vision*. John Wiley and Sons.

SHUTLER, J., 2002. Statistical moments. http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/SHUTLER3/.

SIBELIUS. 2005. *Sibelius version 3.0*. Sibelius Inc. http://www.sibelius.com/.