

CERTIFICATION OF APPROVAL

**Songs Search Using Human Humming Voice**

by

Mohamad Ariff Bin Abdul Rahman

A project dissertation submitted to the  
Computer and Information Science Programme  
Universiti Teknologi PETRONAS  
in partial fulfillment of the requirement for the  
BACHELOR OF TECHNOLOGY (HONS)  
(INFORMATION AND COMMUNICATION TECHNOLOGY)

Approved by,

---

(Mr Jale Ahmad)

UNIVERSITI TEKNOLOGI PETRONAS  
TRONOH, PERAK  
July 2007

## CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.



---

(MOHAMAD ARIFF BIN ABDUL RAHMAN)

## **ABSTRACT**

The system is developed to find songs stored in the database using human humming voice, whereby a sample of the humming voice is compared to songs stored in the system. The main function of the system is to find songs only by humming to the melody of the song. The scopes for this project are human humming voice, voice capture in WAV format, songs database, and MIDI file comparing algorithm. Methodologies used in this system are based on system analysis and design methodology comprising planning, analysis, design and implementation. Java programming language is used to build the system. The system has the functionality of humming voice recording and algorithms comparing both humming voice and song files in the system to find the right song. The intended result of this system is to display the titles of the song and similarity percentage between humming voice melody and songs in the system.

**Keyword:** Java programming language, human humming voice, voice recording, comparing algorithm, user interface.

## **ACKNOWLEDGMENT**

Greatest praise is to Allah for his blessing, the Final Year Project entitled Songs Search Using Human Humming Voice has completed as planned successfully. High gratitude to supervising lecture Mr. Jale Ahmad for his wonderful guidance, undivided support, generous help and enormous effort in making sure this project is a success. Appreciations to internal and external examiners who were involved during this two-semester project and lectures from Computer and Information Science programme for their best support.

Not to forget the support from Universiti Teknologi PETRONAS and colleagues and friends who were always there to help when needed with undivided attention. Lastly to any individuals who are involved directly or indirectly towards the process of completing the project. May all your help and supports be blessed.

## TABLE OF CONTENT

<b>CERTIFICATION.</b>		<b>i</b>
<b>ABSTRACT .</b>		<b>iii</b>
<b>ACKNOWLEDGEMENT .</b>		<b>iv</b>
<b>CHAPTER 1:</b>	<b>INTRODUCTION .</b>	<b>1</b>
	1 Background of Study .	1
	2 Problem Statement .	1
	3 Objectives .	3
	4 Scope of Study .	5
<b>CHAPTER 2:</b>	<b>LITERATURE REVIEW .</b>	<b>7</b>
	1 Introduction .	7
	2 Audio Capturing in Java .	7
	3 MIDI File .	8
	4 JFugue: Java Music Programming Api Features	10
	5 Akoff Sound Lab .	13
	6 Midomi .	15
	7 Singer Identification in Popular Music Recordings Using Voice Coding Features .	15
	8 Music Recognition .	15
	9 WAV and MIDI Format .	16
	10 Levenshtein Distance .	19
<b>CHAPTER 3:</b>	<b>METHODOLOGY/PROJECT WORK .</b>	<b>23</b>
	1 Methodology .	23
	1.1 Planning Phase.	23
	1.2 Analysis Phase.	24
	1.3 Design Phase .	27
	1.4 Implementation Phase.	28
<b>CHAPTER 4:</b>	<b>RESULTS AND FINDINGS.</b>	<b>33</b>
	1 System Process Flow .	33
	2 User Interface .	33
	3 Using the System .	34
<b>CHAPTER 5:</b>	<b>CONCLUSION AND RECOMMENDATION .</b>	<b>36</b>
<b>REFERENCES .</b>		<b>38</b>
<b>APPENDICES .</b>		<b>40</b>

## **LIST OF FIGURES**

Figure 1.1	Process Flows in Song Search	4
Figure 2.1	AKoff Music Composer - Recognition Dialog	13
Figure 3.1	System Use Case Diagram	26
Figure 3.2	System Main User Interface	27
Figure 3.3	System Result Display	32
Figure 4.1	System Interface	34
Figure 4.2	Record Humming Field	35
Figure 4.3	Result Display	35

## **LIST OF TABLES**

Table 2.1	Levenshtein Distance Result	20
Table 3.1	Levenshtein Distance Operation Result	31

# **CHAPTER 1**

## **INTRODUCTION**

### **1. BACKGROUND OF STUDY**

This is the dissertation for Final Year Project (FYP) of January and July 2007 Semester entitled “Songs Search Using Human Humming Voice”. The purpose of this project is to create a Java application system that finds a song stored in the database by using human humming voice, whereby a melody sample of the humming voice is compared to a portion of the song to find a match, regardless of knowing the song information for example title and artist.

### **2. PROBLEM STATEMENT**

#### **2.1. Problem Identification**

Finding the right song in a group of songs would not be difficult if the right title or the singer or composer of the song is known. As all the songs are catalogued with their own title and singer or composer name, it is easy to find the right song requested.

Imagine having to have a tune of a song keeps playing in the mind with knowing what is the title and who is the singer. It would be very hard to find the song as no information of the song is known except for the partial tunes of the song.

Therefore, this system provides the solution of finding a song in a database using partial tunes of the song that matches the song. A user would record the humming voice to the melody of the intended song to be search and the system will capture the voice and compare it with the songs that are in the database.

Apart from that, among the problem faced during the development of the systems are regarding the Java programming language and to identify how to use the packages provided with the language.

Furthermore, problems that occur during the development of the system are caused by humming voice. When humming voice is recorded, sometimes the musical notes produce would get off course with the right pitch of a note. This will cause the system to process the wrong melody and produce inaccurate result.

Other problems faced during the development of this system is the converting process of humming voice 'wav' format to 'midi' format as process to be done is very intricate. Therefore for the development of the system, the part of converting the humming voice will by done by a shareware system called "Digital Ear".

## **2.2. Project Significance**

This project is by mean to be developed to help the intended users by any group. This system will be easy to be used with functions and user interface that is simple. User would only need to hum to the melody of the song to be search and wait for the system to produce the result.

Furthermore, the system will show result of similar songs to the melody hummed by percentage making it easy for the user to get the best result and the right song. The produced system shows how latest technology can be exploited to help users in minimizing their effort and assist them in any way possible.

The system has achieved its objectives which are discussed in the next section and further discussion on the development and technologies used to develop the system can be found later in this dissertation report.



### **3. OBJECTIVES**

#### **3.1. Searching the Right Song**

Main objective of this project is to produce a system, where user hums for the song user would like to search in the song database. The system will then matches the humming voice with song that has the same melody and search for it.

The humming voice is recorded by the system into a 'wav' sound format. This 'midi' humming melody will be used by the system for comparison to the songs stored in the system in order to find a match of both files.

#### **3.2. Simplicity with Accurate Result**

The system would accurately find the right song user intended to search with minimal input from the user. User would only have to hum to song they would like to search.

Apart from that, the system will display results of songs that matched the humming melody by percentage to the similarity. This is to give the user the ability to choose which song has the similarity to the intended song.

#### **3.3. Effective and Less Time Consumption**

The system would work effectively in searching for the song by cutting the time taken by user with direct result instead of user scrolling to each of the songs to find the intended one.

The system will display the results to the user and this helps the user to get the information for example the title of the song with having to hear every song stored in the system and listen to each songs to find a match for the intended song.

### 3.4. How the System Works

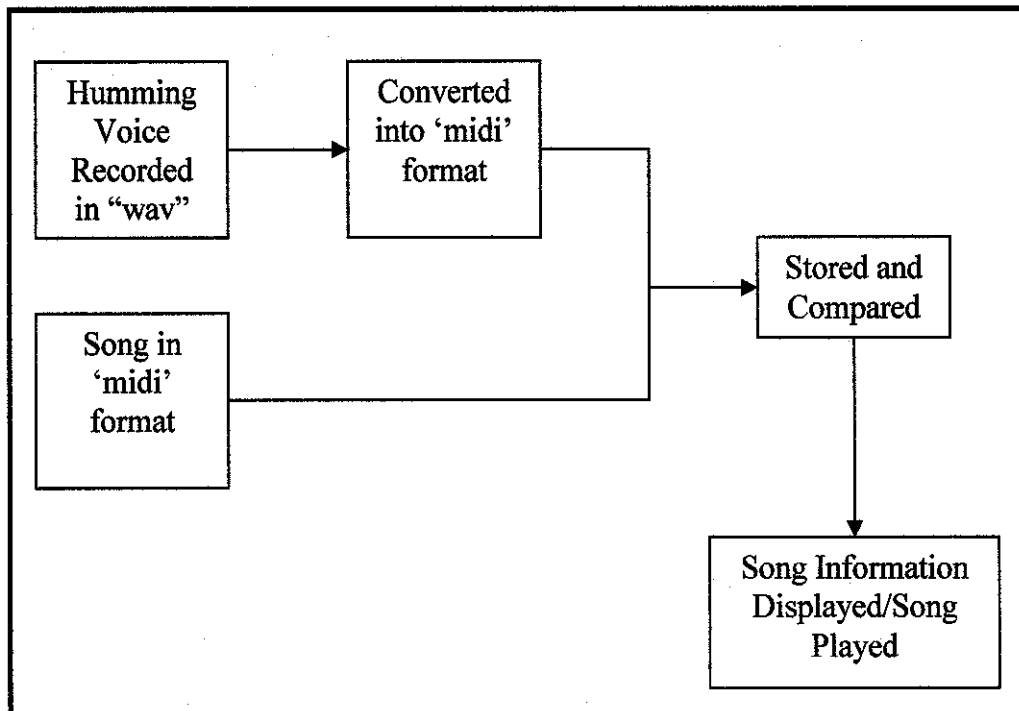


Figure 1.1: Process Flows in Song Search

Figure 1 shows the flow of the system. First, user of the system will hum to the tune of intended song to be searched. The hum voice, recorded in the form of 'wav' format will be converted into 'midi' for before being stored. In the song database, all the songs will be tagged to smaller portions and converted to 'midi' file and then be compared with the 'midi' file of the users voice for matching the right song.

The objectives set has been made as a guideline during the development of the system so that the system function as it is intended to be and produce the right result. These objectives also acts as a list of main functionality that the system has where functions of the system are mainly about helping user in finding the right song just by using humming melody that they know of.

## **4. SCOPE OF STUDY**

The scopes of study for this project are the areas that will be studied to build the system and these scopes are elaborated below

### **4.1. Voice (Human Humming Voice)**

In this area, research were done on understand human voice especially humming. This is to understand the ability of user to hum and the accuracy the humming voice to produce the right pitch for a series of several music notes. In this system, user is required to hum a portion of melody of a song that comprises of several music notes.

The humming voice produced can just be considered as noises and when recorded into the system, processes are needed to be done on how to recognize the humming melody into a series of real musical notes for further processing. Therefore, the accuracy of the humming voice is crucial in order to get accurate result for the humming melody.

In producing the humming melody, which is the series of recognized musical notes from the humming voice, the system uses a simple technique where it will record the humming voice into a 'wav' sound format and converts it into 'midi' sound format. Further discussion on what is 'midi' format and why 'midi' format is used can be found in Chapter 2 and Chapter 3 of this report consecutively.

### **4.2. Songs**

The formats of the songs stored in they are all in 'midi' format and the system will be as simple as only with one MIDI channel of musical instrument involved. This format is used as the system is at the beginning of its development and this is proven by the system that the functionality works by using the format.

### **4.3. Sound Files Conversion**

Sound files conversion method will be used to convert both humming voice which is 'wav' format sound file 'midi' format sound file type. As for the development of the system, this part of functions is done by an outsource system as developing the system with this function involves lot of intricate multimedia and sound programming.

### **4.4. Comparison Algorithm**

The scope for this area is to study and find an algorithm that will compare two 'midi' files of the humming voice and the stored songs and find the similar portion musical notes in order to have successful search of the song.

The process is done where the user humming voice is a 'midi' humming melody which comprises of a series of musical notes. Next, the songs that are already available in the system which are in 'midi' format will be compared to the humming melody.

Since both the humming melody and songs stored in the system are in 'midi' format, comparison algorithm is implemented to compare both melodies. This is done by comparing every musical note of both melodies to find similarity. Further discussion of what comparison algorithm used and how it is implemented is in Chapter 2 and Chapter 3 of this report.

## **CHAPTER 2: LITERATURE REVIEW**

### **1. INTRODUCTION**

The main programming language that will be used in this project is Java programming language. Therefore this chapter will be discussing the relevant works there were done with similarities to the system and the technologies that can be implemented. The available projects and systems are reviewed and analyzed in order to understand the technologies. These technologies and systems will be reviewed to help in development of the system.

### **2. AUDIO CAPTURING IN JAVA**

Capturing refers to the process of obtaining a signal from outside the computer. A common application of audio capture is recording, such as recording the microphone input to a sound file. However, capturing isn't synonymous with recording, because recording implies that the application always saves the sound data that's coming in. An application that captures audio doesn't necessarily store the audio. Instead it might do something with the data as it's coming in, such as transcribe speech into text and then discard each buffer of audio as soon as it's finished with that buffer.

Typical audio-input system in an implementation of the Java™ Sound API consists of:

- a) An input port, such as a microphone port or a line-in port, which feeds its incoming audio data into:
- b) A mixer, which places the input data in:

- c) One or more target data lines, from which an application can retrieve the data.

Commonly, only one input port can be open at a time, but an audio-input mixer that mixes audio from multiple ports is also possible. Another scenario consists of a mixer that has no ports but instead gets its audio input over a network.

### **3. MIDI FILE**

A MIDI file is a data file. It stores information, just like a text (ie, ASCII) file may store the text of a story or newspaper article, but a MIDI file contains musical information. Specifically, a MIDI file stores MIDI data, the data (ie, commands) that musical instruments transmit between each other to control such things as playing notes and adjusting an instrument's sound in various ways.

MIDI is binary data, and a MIDI file is therefore a binary file. It is not possible to load a MIDI file into a text editor such as Notepad and view it. However, MIDI File Disassembler/Assembler utility can be used to convert a MIDI file to readable text, edit it in Notepad, and then convert it back to a MIDI file using the same software).

The MIDI file format was devised to be able to store any kind of MIDI message, including System Exclusive messages, along with a timestamp for each MIDI message. A timestamp is simply the time when the message was generated. Using the timestamps, a sequencer can playback all of the MIDI messages within the MIDI file at the same relative times as when the messages were originally generated. In other words, a sequencer can playback all of the note messages, and other MIDI messages, with the original "musical rhythms". A MIDI file can also store other information relating to a musical performance, such as tempo, key signature, and time signature. A MIDI file is therefore a generic, standardized file format designed to store a "musical performance", and is used by many sequencers. A MIDI file even has provisions for storing the names of tracks in a sequencer, and other sequencer settings.

There are 3 different "Types" (sometimes called "Formats") of MIDI files. A Type 0 file contains only one track, and all of the MIDI messages (ie, the entire performance) are placed in that one track, even if this represents many musical parts upon different MIDI channels. A Type 1 file separates each musical part upon its own track. Both Type 0 and 1 store one "song", "pattern", or musical performance. A Type 2 file, which is extremely rare, is akin to a collection of Type 0 files all crammed into one MIDI file. Type 2 is used to store a collection of songs or patterns, for example, numerous drum beats. (If you need to convert a MIDI file to the various Types, use my Midi File Conversion utility).

Besides sequencers, other software and hardware devices may use MIDI files. For example, a patch editor may store an instrument's patch settings in a MIDI file, by storing System Exclusive messages (received from the instrument) within the MIDI file. In this case, the patch editor may not care about the timestamp associated with each SysEx message.

## **4. JFUGUE: JAVA MUSIC PROGRAMMING API FEATURES**

JFugue is full of interesting and useful features that let user experiment with music like no other programming tool. It is a Java Music API that helps in programming musical notes into coding. Among the features of JFugue are:

### **4.1. Create music using JFugue "MusicStrings", an easy and quick way to specify musical notes and other events**

- JFugue supports for all audible MIDI events and files. Any file that can be played in MIDI, can be played in JFugue.
- Easily specify notes, chords, ties, instruments, key signatures, controller events, percussion sounds, and more.
- Play the Music Strings directly from a program and save Music Strings as text files, or save them into MIDI files.

### **4.2. Convert existing MIDI files into JFugue MusicStrings**

- Read and understand any of created MIDI files.
- Manipulating musical patterns and snippets from existing MIDI files.

### **4.3. Manipulate Patterns of music**

- JFugue's Pattern class lets several processes for example to change, transform, or measure pieces of music
  - Change octaves, scales, durations
  - Replace notes, instruments, chords
  - Mutate musical patterns to come up with a variety of related sounds
- Patterns can be easily recombined, looped, and otherwise manipulated
- Create factories of patterns to produce well-known rhythms, like Rock or Swing beats



#### **4.4. Easily and intuitively specify rhythms**

- Type or generate rhythm sections easily. For example: "ooO' ooO' ooO' OOO", where "o" might represent a snare drum, "O" a bass drum, and ' a hi-hat
- Create layers of percussion sounds to be played on the same MIDI channel

#### **4.5. Support for Microtonal Music**

- Easily create music like an Indonesian gamelan, or in an Indian classical or Turkish style - all need is the frequency of the notes to be played. JFugue manipulates MIDI events to do the rest

#### **4.6. Send music to, or receive music from, other MIDI devices**

- Send the Patterns or any MIDI files to MIDI keyboard or sequencer
- Record music into Patterns as played on the keyboard
- Develop music training software, or games that involve playing notes on the MIDI device

#### **4.7. Experience simplified use of MIDI**

- One line of code to play any MIDI file (well, you'll have to manage any exceptions)
- MIDI generated from JFugue Music Strings can be worked with like any other MIDI sequence - add lyrics, track names, etc.

#### **4.8. Anticipate musical events before they happen**

- Easily set up JFugue to alert you before a musical event is fired
- Develop your own virtual instruments or interactive characters that need to begin some animated action before the notes actually sound

#### **4.9. Excellent architecture maps any Parser to any Renderer, allows for easy and powerful reconfiguration**

- A Parser converts data into musical events. Examples: MusicStringParser, MidiParser
- A Renderer converts musical events into something that can be seen or heard. Examples: MidiRenderer, MusicStringRenderer
- Create your own Parsers and Renderers, and easily use them in your program!
  - Create your own Parser to convert music from ABC format
  - Create your own Renderer to turn musical events into sheetmusic, or create a music-driven light show
  - Create your own Parser and Renderer to support MusicXML
- Multiple Renderers can listen to an individual Parser

## 5. AKOFF SOUNDS LAB

### 5.1. AKoff Music Composer Version 2.0

This software is designed for recognition of polyphonic music from audio source and its conversion to MIDI score. Now one can compose music without MIDI keyboard and even without knowledge of notes or how to play musical instruments. It is as easy as humming a tune into a microphone and Composer will do the rest. Composer will take into consideration individual performance style tracking note dynamics and pitch bends.

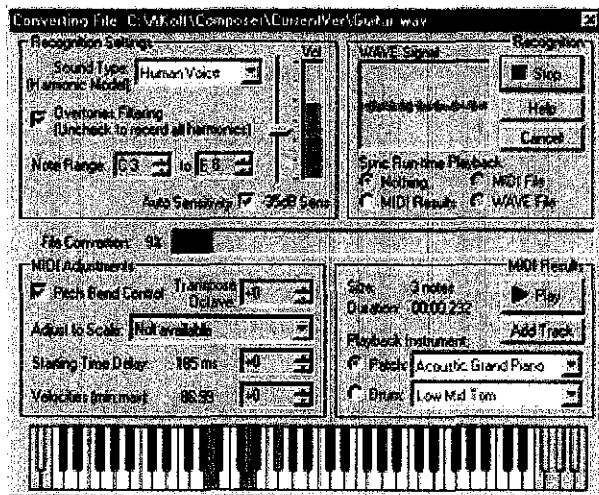


Figure 2.1: AKoff Music Composer - Recognition Dialog

### 5.2. How AKoff Music Composer Works

Composer analyzes a stream of audio signals from pre-recorded WAVE files or directly from audio input of your sound card in real-time. It can be sound from microphone, linear input or audio CD.

Composer normally recognizes polyphonic music with one instrument or voice. This means user won't get the appropriate results if they try to recognize many instruments playing at the same time especially with drums. Composer determines note dynamics and frequencies and translates this information into MIDI events.

Composer doesn't automatically recognize the types of sounding instruments. Moreover, human voice and instruments have various timbres and complicated harmonic components; therefore recognition accuracy depends on concrete instrument or singing style. Furthermore the recognition is influenced by quality of WAV recordings such as background noises and recording level.

The complicated mathematical algorithms of DSP (Digital Signal Processing) require a great amount of calculations and consequently a fast computer. When recognizing audio input in real-time, Composer only works reliably on Pentium 150 and higher processors. On weaker machines, the incorrect working or dead halt of the program may occur, which is removed through pressing Ctrl-Alt-Del keys.

During recognition the recorded notes are automatically shown on the graphic keyboard. After recognition is stopped, user can play and adjust the obtained melody. User can select the playback MIDI instrument, change octave and also try to correct music scale or use Pitch Bend Control. Further user can add the recorded track to the general list of MIDI tracks and then save the generated MIDI file.

## **6. MIDOMI**

Midomi is a system that is developed by Melodis Corporation to online user where they provide service of searching the list of songs they have in the database. Midomi is a music search tool powered by human voice. The favorite music in the songs database is search using singing, humming, or whistling. No development information on the system is provided by the company.

## **7. SINGER IDENTIFICATION USING VOICE CODING**

This is a study done by Massachusetts Institute of Technology on singer identification in popular music recording has a few similarities with the project. The research presented in the paper attempted to automatically establish the identity of a singer using acoustic features extracted from songs in a database of popular music. As a first step, an untrained algorithm for automatically extracting vocal segments from within songs is presented. Once these vocal segments are identified, they are presented to a singer identification system that has been trained on data taken from other songs by the same artists in the database.

## **8. MUSIC RECOGNITION**

In a few words music recognition is mathematical analysis of an audio signal (usually in WAV format) and its conversion into musical notation (usually in MIDI format). This is a very hard artificial intelligence problem. For comparison, the problem of recognition of scanned text (OCR - Optical Character Recognition) is solved with 95% accuracy - it is an average exactitude of recognition of the programs of the given class. The programs of speech recognition already work with 70-80% accuracy, whereas the systems of music recognition work with 60-70% accuracy but only for a single voice melody (one note at a time). For polyphonic music the accuracy is even lower.

To create a MIDI file for a song recorded in WAV format a musician must determine pitch, velocity and duration of each note being played and record these parameters into a sequence of MIDI events. Music recognition software must do the same things. Even for a single instrument song it is not a simple task, because a WAV recording contains waveform signals and doesn't contain any music specific data.

In general cases the variety of music timbres, harmonic constructions and transitions make it impossible to create a mathematical algorithm for precise reconstruction of a music score from the audio sources. It is hard to recognize audio data which contains many instruments, drums and percussions or clipping signals, unstable pitch sounds and background noises. Therefore it is a need to find a system that will produce a MIDI material that represents the basic melody and chords of recognized music.

## **9. WAV AND MIDI FORMATS**

The difference between WAV and MIDI formats consists in representation of sound and music. WAV format is digital recording of any sound (including speech) and MIDI format is principally sequence of notes (or MIDI events). The relations are approximately the same as between sounded speech and printed text.

A WAV file is the recording of a sound wave. It is the mix of all the given sounds (instruments, voices, background noises) that could have been heard at the moment of recording. Example of human voice recorded in WAV format, it could not be edited to any note or change any instrument in music recorded in a WAV file. The Standard Windows PCM WAV format contains only Pulse Code Modulation data without compression. PCM format is the only kind that saves the entire wave completely with no data loss.

There are many other formats for audio recording. They differ from each other by compression algorithms and can be referred to one group. The conversion from one

format into another is very simple. There are many sound editors which allow one to do this.

The following is a list of some audio formats with file extensions:

- Standard Windows PCM waveform (.WAV)
- Microsoft ADPCM waveform (.WAV)
- MPEG Layer (.MP2, .MP3)
- RealAudio (.RA)
- Sound Blaster voice file format (.VOC)
- MIDI format

MIDI files store MIDI messages, which are commands that tell a musical device what to do in order to make music. For example, there is a MIDI message that tells a device to play a particular note. There is another MIDI message that tells a device to change its current "sound" to a particular patch or instrument. Etc. The MIDI file also stores timestamps, and other information that a sequencer needs to play some "musical performance" by transmitting all of the MIDI messages in the file to all MIDI devices. In other words, a MIDI file contains hundreds (to thousands) of instructions that tell one or more sound modules (either external ones connected to your sequencer's MIDI Out, or sound modules built into your computer's sound card) how to reproduce every single, individual note and nuance of a musical performance.

A WAVE file stores a digital audio waveform. This data must be played back upon a device with a Digital To Analog Converter (ie, DAC) such as a MIDI sampler (ie, the AKAI S1000 for example) or a computer sound card's DAC. There are no timestamps, or other information concerning musical rhythms or tempo stored in a WAVE file. There is only digital audio data. Typically, a WAVE file is used to store a looped, single-pitched waveform (which a sampler transposes and plays back over a range of MIDI note numbers), or a short, non-looped percussive sound or sound effect, or often a digital audio recording (ie, stereo mixdown) of some musical performance stored upon your hard drive.

The only way to record and store a musical performance within a WAVE file is to digitize the audio output of all instruments while they play that performance. The result will be a typically large WAVE file that represents the digitized "sound" of all instruments playing the musical piece in realtime. The act of doing such is often referred to as "Hard disk recording" because the WAVE data usually has to be recorded directly to a large Hard Drive while the DAC digitizes the performance.

A MIDI file allows easy editing of the individual musical parts, because each part is usually assigned to its own MIDI channel, and it's easy to separate that part's MIDI data from the other parts' MIDI data, based upon the MIDI channel in each MIDI message. On the other hand, you can't easily separate the digital audio data of one instrument from the digital audio data of another instrument, if the two were digitized simultaneously into one WAVE file. For editing of individual musical parts, each part should be digitized and stored in its own WAVE file. Due to the difficulties in separating musical parts from a WAVE file, a conversion from WAVE format to MIDI format is not too feasible, although the reverse is not a problem.

One benefit of a WAVE file is that its "sound" is not usually dependent upon the playback device. A WAVE file should sound the same upon different equipment (with reasonably similar specs). On the other hand, a MIDI file can sound considerably different upon different MIDI gear. This is because the MIDI file doesn't dictate what means an instrument uses to produce its sound. The MIDI file may sound very different upon an instrument that uses FM synthesis than it will sound upon an instrument using "wavetable synthesis" (ie, looped, digital audio waveforms in ROM). But, standards such as *General MIDI* seek to alleviate some of the discrepancies between MIDI devices.

An MP3 file, like a WAVE file, stores digital audio data. So a MIDI file and an MP3 file are different in exactly the same way that a MIDI file and a WAVE file are different. Indeed, a WAVE and MP3 file are two different ways of storing the exact same type of data. The primary difference between a WAVE and MP3 file is that the



latter uses compression to squeeze the data down in size, resulting in a typically much smaller file size.

## 10. LEVENSHTEIN DISTANCE

In information theory and computer science, the Levenshtein distance is a string metric which is one way to measure edit distance. The Levenshtein distance between two strings is given by the minimum number of operations needed to transform one string into the other, where an operation is an insertion, deletion, or substitution of a single character. It is named after Vladimir Levenshtein, who considered this distance in 1965. It is useful in applications that need to determine how similar two strings are, such as spell checkers.

For example, the Levenshtein distance between "kitten" and "sitting" is 3, since these three edits change one into the other, and there is no way to do it with fewer than three edits:

1. kitten → sitten (substitution of 's' for 'k')
2. sitten → sittin (substitution of 'i' for 'e')
3. sittin → sitting (insert 'g' at the end)

It can be considered a generalization of the Hamming distance, which is used for strings of the same length and only considers substitution edits. There are also further generalizations of the Levenshtein distance that consider, for example, exchanging two characters as an operation, like in the Damerau-Levenshtein distance algorithm.

A commonly-used bottom-up dynamic programming algorithm for computing the Levenshtein distance involves the use of an  $(n + 1) \times (m + 1)$  matrix, where  $n$  and  $m$  are the lengths of the two strings. This algorithm is based on the Wagner-Fischer algorithm for edit distance.

This is pseudocode for a function *LevenshteinDistance* that takes two strings, *s* of length *m*, and *t* of length *n*, and computes the Levenshtein distance between them:

```

int LevenshteinDistance(char s[1..m], char t[1..n])

// d is a table with m+1 rows and n+1 columns
declare int d[0..m, 0..n]

for i from 0 to m
  d[i, 0] := i
for j from 1 to n
  d[0, j] := j

for i from 1 to m
  for j from 1 to n
    if s[i] = t[j] then cost := 0
    else cost := 1
    d[i, j] := minimum(
      d[i-1, j] + 1, // deletion
      d[i, j-1] + 1, // insertion
      d[i-1, j-1] + cost // substitution
    )

return d[m, n]

```

An example of the resulting matrix (the minimum steps to be taken are bold):

**Table 2.1: Levenshtein Distance Result Table**

		k	i	t	t	e	n
	<b>0</b>	1	2	3	4	5	6
s	1	<b>1</b>	2	3	4	5	6
i	2	2	<b>1</b>	2	3	4	5
t	3	3	2	<b>1</b>	2	3	4
t	4	4	3	2	<b>1</b>	2	3
i	5	5	4	3	2	<b>2</b>	3
n	6	6	5	4	3	3	<b>2</b>
g	7	7	6	5	4	4	<b>3</b>

The invariant maintained throughout the algorithm is that we can transform the initial segment  $s[1..i]$  into  $t[1..j]$  using a minimum of  $d[i,j]$  operations. At the end, the bottom-right element of the array contains the answer.

This algorithm is essentially part of a solution to the Longest Common Subsequence problem (LCS), in the particular case of 2 input lists.

### 10.1. Proof of Correctness

As mentioned earlier, the invariant is that we can transform the initial segment  $s[1..i]$  into  $t[1..j]$  using a minimum of  $d[i,j]$  operations. This invariant holds since:

- It is initially true on row and column 0 because  $s[1..i]$  can be transformed into the empty string  $t[1..0]$  by simply dropping all  $i$  characters. Similarly, we can transform  $s[1..0]$  to  $t[1..j]$  by simply adding all  $j$  characters.
- The minimum is taken over three distances, each of which is feasible:
  - If we can transform  $s[1..i]$  to  $t[1..j-1]$  in  $k$  operations, then we can simply add  $t[j]$  afterwards to get  $t[1..j]$  in  $k+1$  operations.
  - If we can transform  $s[1..i-1]$  to  $t[1..j]$  in  $k$  operations, then we can do the same operations on  $s[1..i]$  and then remove the original  $s[i]$  at the end in  $k+1$  operations.
  - If we can transform  $s[1..i-1]$  to  $t[1..j-1]$  in  $k$  operations, we can do the same to  $s[1..i]$  and then do a substitution of  $t[j]$  for the original  $s[i]$  at the end if necessary, requiring  $k+\text{cost}$  operations.
- The operations required to transform  $s[1..n]$  into  $t[1..m]$  is of course the number required to transform all of  $s$  into all of  $t$ , and so  $d[n,m]$  holds our result.

This proof fails to validate that the number placed in  $d[i,j]$  is in fact minimal; this is more difficult to show, and involves an argument by contradiction in which we assume  $d[i,j]$  is smaller than the minimum of the three, and use this to show one of the three is not minimal.

## 10.2. Possible Improvements

Possible improvements to this algorithm include adapting the algorithm to use less space,  $O(m)$  instead of  $O(mn)$ , since it only requires that the previous row and current row be stored at any one time. It can store the number of insertions, deletions, and substitutions separately, or even the positions at which they occur, which is always  $j$ . Further is by giving different penalty costs to insertion, deletion and substitution. The initialization of  $d[i,0]$  can be moved inside the main outer loop. This algorithm parallelizes poorly, due to a large number of data dependencies. However, all the cost values can be computed in parallel, and the algorithm can be adapted to perform the minimum function in phases to eliminate dependencies.

## 10.3. Upper and Lower Bounds

The Levenshtein distance has several simple upper and lower bounds that are useful in applications which compute many of them and compare them. These include:

- It is always at least the difference of the sizes of the two strings.
- It is at most the length of the longer string.
- It is zero if and only if the strings are identical.
- If the strings are the same size, the Hamming distance is an upper bound on the Levenshtein distance.
- If the strings are called  $s$  and  $t$ , the number of characters (not counting duplicates) found in  $s$  but not in  $t$  is a lower bound.

## **CHAPTER 3**

### **METHODOLOGY/PROJECT WORK**

#### **1. Methodology**

The project is developed by several phases until the end of its completion. The phases cover the research on available technologies to develop the system, analysis and understanding of human voice, music pattern recognition and comparing algorithm and the development of the system itself. The methodology used in the development of the system is based on four phases of system analysis and design which are planning, analysis, design and implementation.

##### **1.1. Planning Phase**

During the planning phase of the system development, objectives of the system were identified to plan on what the functionalities that the system will have and area to be focused on during the analysis.

As discussed in Chapter 1, the system objectives are:

a) **Searching the Right Song**

The functionality of the system is to search the right song from group of songs by comparing to user's humming voice melody

b) **Simplicity with Accurate Result**

The system would be easy to use for user and the result produced by the system is accurate by displaying the right song search by the user.

c) **Effective and Less Time Consumption**

User will find the system effective as it cuts users time instead of listening to all songs intended to search manually, user would just let the system do the work.

Once the objectives were identified, the next step was to analyze the feasibility of the system through several areas technically, economically and development. Analysis phase follows as the next stage of the development of the system.

## **1.2. Analysis Phase**

In this phase, areas involving the system are identified where these are the focus of the system. Information gathering were also done as discussed in literature review. The areas involve the functionality that the system has and process needed for the development of the system. These areas are discussed below.

### **1.2.1. Technology**

The system is developed using Java language which covers every aspect and function that the system has. An example of the functions is the recording of humming voice into 'wav' format and the displaying the songs result. Apart from that, an algorithm is also being implemented for the system's function of comparing the similarity of the humming voice and the correct song.

Technologies implemented in this system are the technologies used to develop a Java application. Among the packages of Java programming used is the Java Sound API package which is used to record to user's humming voice melody and Java JFugue package for the manipulation of songs musical notes for comparison algorithm.

All of the technologies used doing the development of the program involves writing source codes in Java programming language. The beauty of this system is that, to implement the system to any computer, hardware needed are a microphone, speaker and the Java enabled computer.

### **1.2.2. Conversion 'wav' Format File to 'midi' Format File**

Currently, the develop system is only capable of capturing humming voice from an in-line microphone and save it into 'wav' form. The next step of converting 'wav' file to 'midi' file is not possible at the moment as it involves intricate process of sound and multimedia programming. As discussed in Problem Statement of Chapter 1, this function will be done by already available program.

### **1.2.3. Human Voice**

Analysis and studies were done in understanding the human voice characteristic in the form of humming. Humming voice is studied to find out the character of humming voice in order to understand and identify the right pitch that is producing the right note. In the development of the system, studies were done on how to capture the humming voice melody recorded using a microphone connected to a computer.

For this project, the system records the humming voice melody from the microphone and stores inside the system as a 'wav' humming melody. This 'wav' melody will then be passed to other process in the system.

### **1.2.4. Songs Melody**

In this part, songs melodies are the songs that are stored in the system for comparison to the humming voice melody. These songs are in 'midi' file format which in a 'midi' file, it contains a stream of musical notes which are combined together to produce the song file.

The 'midi' format is chosen for the songs stored in the system because 'midi' file consists of musical notes and the arrangements of the musical notes are what produce the song. Furthermore, this 'midi' file is easier to be processed during the comparison process of the system.

### 1.2.5. Comparing Algorithm

It is vital to understand and find the algorithm of comparing the 'midi' file of the humming voice and the 'midi' files of the songs, in order to have a successful search of songs in the database.

As discussed in the Literature review, the comparing algorithm used for the system is the Levenshtein Distance. Further discussion on how the algorithm is implemented is in the next section.

### 1.2.6. System Use Case Diagram

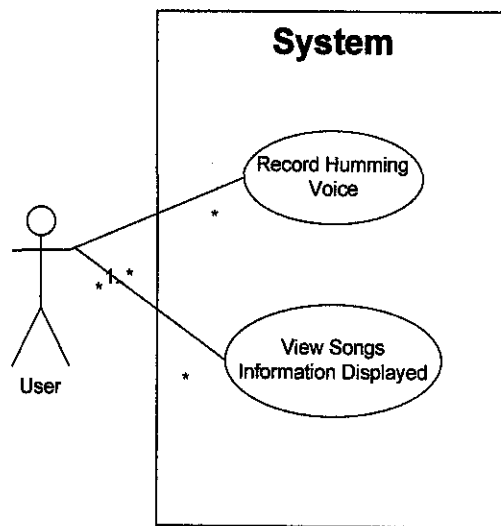


Figure 3.1: System Use Case Diagram

The use case diagram shown depicts how user interacts with the system. To operate the system, user needs to record the humming voice melody using the system and click the button for viewing the result.



### 1.3. Design Phase

The design phase of the system development involves the system design, the user interface and what technologies are used for doing the design.

#### 1.3.1. System User Interface

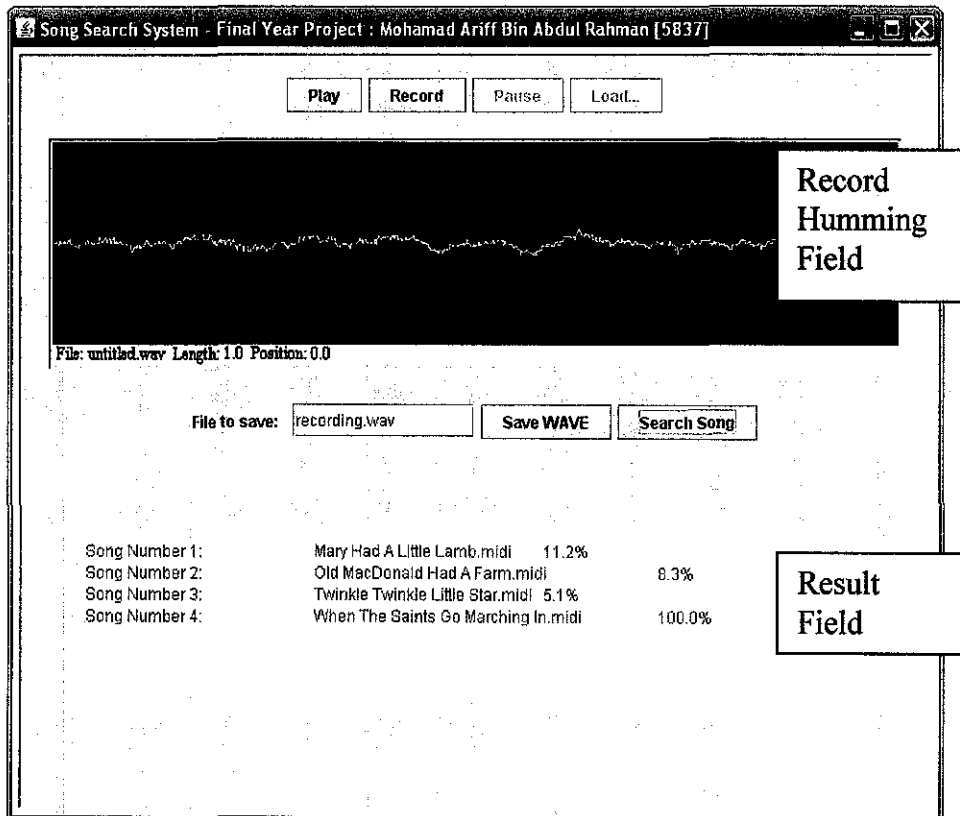


Figure 3.2: System Main User Interface

The user interface of the system is divided into two parts which are:

1. Record Humming
2. Result

The system interface is developed using Java programming language.

Functionalities of the system is activated when the user click the buttons available on the interface. Further discussion on how the users use the system is in Chapter 4.

## 1.4. Implementation Phase

During this phase, activities done were the construction of the system that involve code writing using Java language, the implementation of comparing algorithm into coding and activities to realize the system. These activities are discussed further in this section.

### 1.4.1. Humming Voice Capture Using Java Sound API

Capturing refers to the process of obtaining a signal from outside the computer. This system captures audio from a microphone and when the user save their humming voice, the system will save it as a wave file.

With the facilities that Java Sound API package provides, the system can record user's humming voice and save it into 'wav' file. The following codes are extracted from the system on the function of capturing humming voice melody.

The codes in the box below shows how the system reads data from the input channel which is the microphone and writes to the output stream

```
// define the required attributes for our line and make sure a compatible line is supported.

AudioFormat format = formatControls.getFormat();
DataLine.Info info = new DataLine.Info(TargetDataLine.class,
    format);

if (!AudioSystem.isLineSupported(info)) {
    shutDown("Line matching " + info + " not supported.");
    return; }
try {
    line = (TargetDataLine) AudioSystem.getLine(info);
    line.open(format, line.getBufferSize());
} catch (LineUnavailableException ex) {
    shutDown("Unable to open the line: " + ex);
    return;
} catch (SecurityException ex) {
    shutDown(ex.toString());
    //JavaSound.showInfoDialog();
    return;
} catch (Exception ex) {
    shutDown(ex.toString());
    return;
}
```

When the recording is done, the output stream is closed and the humming voice melody is ready for playback or saves into 'wav' file. The coding below shows how the recorded humming voice melody is saved into 'wav' file.

```
public void saveToFile(String name, AudioFileFormat.Type fileType) {  
    if (audioInputStream == null) {  
        reportStatus("No loaded audio to save");  
        return;  
    } else if (file != null) {  
        createAudioInputStream(file, false);  
    }  
  
    // reset to the beginning of the captured data  
    try {  
        audioInputStream.reset();  
    } catch (Exception e) {  
        reportStatus("Unable to reset stream " + e);  
        return;  
    }  
  
    File file = new File(fileName = name);  
    try {  
        if (AudioSystem.write(audioInputStream, fileType, file) == -1) {  
            throw new IOException("Problems writing to file");  
        }  
    } catch (Exception ex) { reportStatus(ex.toString()); }  
    samplingGraph.repaint();  
}
```

When the 'wav' file of humming voice melody is available, an outside program is use to convert the 'wave' into 'midi' file.

#### 1.4.2. MIDI Files and JFugue

The next step is the processing of 'midi' file of humming melody and songs 'midi' files stored in the system. Once the 'midi' files are ready, the system will implement the JFugue processing for the purpose of identifying the musical notes that creates the 'midi' files and convert the musical notes into string.

The process done is by using 'Pattern' class provided in JFugue where the 'midi' files are loaded into the system and converted into strings which are the musical notes of the song. The box below shows the example on how 'Pattern' class is implemented.

```
Player player = new Player();

    Pattern humming = new Pattern();
    String shumming;

    try
    {
        humming = player.loadMidi(new File("humming.midi"));

        shumming = humming.toString();
    }

    catch (Exception e)
    {
        System.err.println("File input error");
    }
```

The some process is done to every 'midi' song stored in the system to get the musical notes as these string are to be process in the comparing algorithm.

### 1.4.3. Comparing Algorithm Using Levenshtein Distance

In this process, both the musical notes string of the humming voice melody and all the musical note strings of the songs are compared. The music string of humming voice will be compared one by one to all the music strings of the song to find similarities of the song. The comparison algorithm used for this process is based on Levenshtein Distance.

The box below shows the coding of how the Levenshtein Distance is implemented as a function in the system. This function will be called when the comparing process is needed.

```

public int ld( String s, String t) {
    int n = s.length();
    int m = t.length();

    if (n == 0) return m;
    if (m == 0) return n;

    int[][] d = new int[n + 1][m + 1];

    for ( int i = 0; i <= n; d[i][0] = i++ );
    for ( int j = 1; j <= m; d[0][j] = j++ );

    for ( int i = 1; i <= n; i++ ) {
        char sc = s.charAt( i-1 );
        for (int j = 1; j <= m; j++) {
            int v = d[i-1][j-1];
            if ( t.charAt( j-1 ) != sc ) v++;
            d[i][j] = Math.min( Math.min( d[i-1][j] + 1, d[i][j-1] + 1 ), v );
        }
    }
    return d[n][m];
}

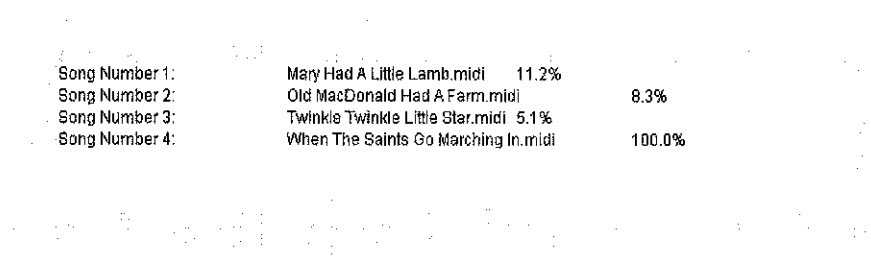
```

This is how Levenshtein Distance works and implemented into the system. The table below shows humming voice melody notes of (C C G G A A G) and being compared to song melody notes of (C C G A B E F). The result of the comparison based on Levenshtein algorithm produces 4 operations which means, 4 differences occurred with 4 different music notes for the comparison.

**Table 3.1: Levenshtein Distance Operation Result**

	C	C	G	G	A	A	G
C	<b>0</b>	1	2	3	4	5	6
C	1	<b>0</b>	1	2	3	4	5
G	2	1	<b>0</b>	1	2	3	4
A	3	2	1	<b>1</b>	2	3	4
B	4	3	2	2	<b>2</b>	3	4
E	5	4	3	3	3	<b>3</b>	4
F	6	5	4	4	4	4	<b>4</b>

The result produces which are 4 as an example showed is used to calculate the percentage different of musical note string of the humming voice melody to each of the 'midi' song stored in the database. When the calculation is done, the system will display result to the user by percentage of similarity to every song in the system.



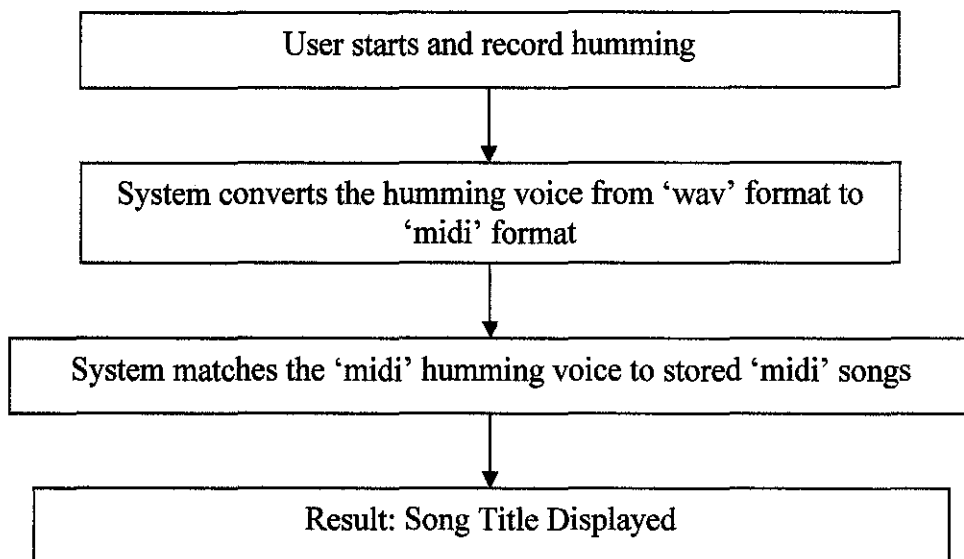
Song Number 1:	Mary Had A Little Lamb.midi	11.2%
Song Number 2:	Old MacDonald Had A Farm.midi	8.3%
Song Number 3:	Twinkle Twinkle Little Star.midi	5.1%
Song Number 4:	When The Saints Go Marching In.midi	100.0%

**Figure 3.3: System Result Display**

## CHAPTER 4

### RESULTS AND FINDING

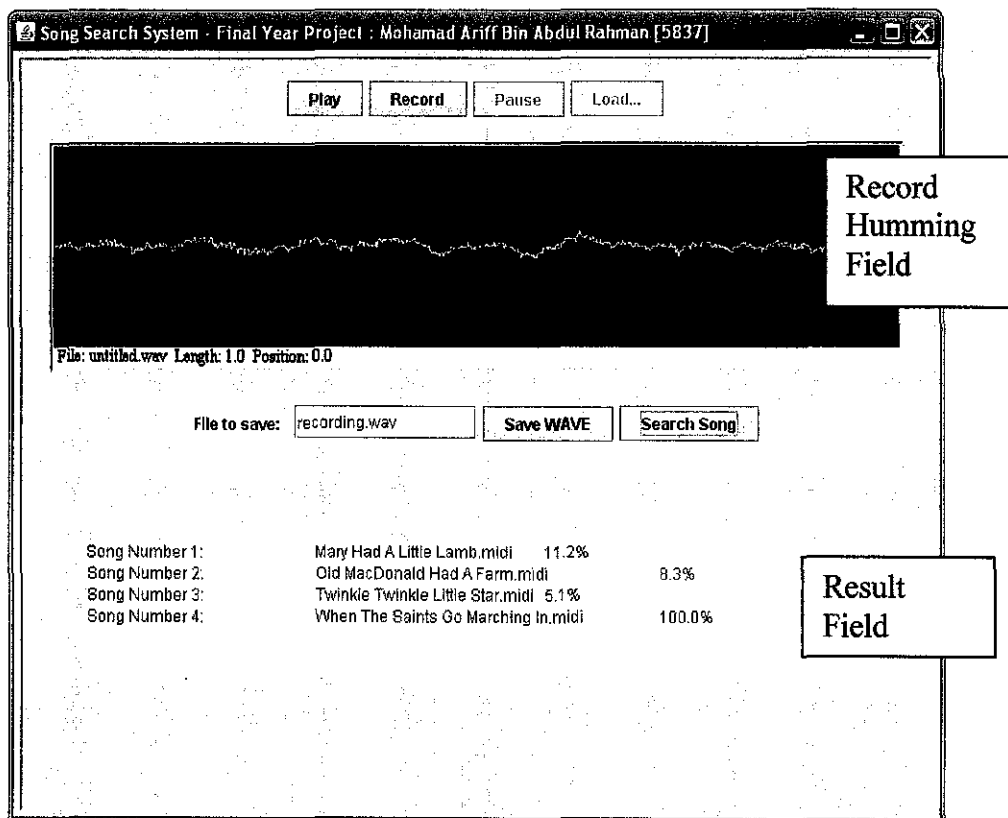
#### 1. SYSTEM PROCESS FLOW



#### 2. USER INTERFACE

The system provides a very simple and easy to use interface in order to minimize the time consumption. User will have to click only three to four buttons and the result will be displayed.

The user interface of the system is easy to be learn and attractive. The user interface will be shown with screenshots and followed by how the system can be used by the user.



**Figure 4.1: System Interface**

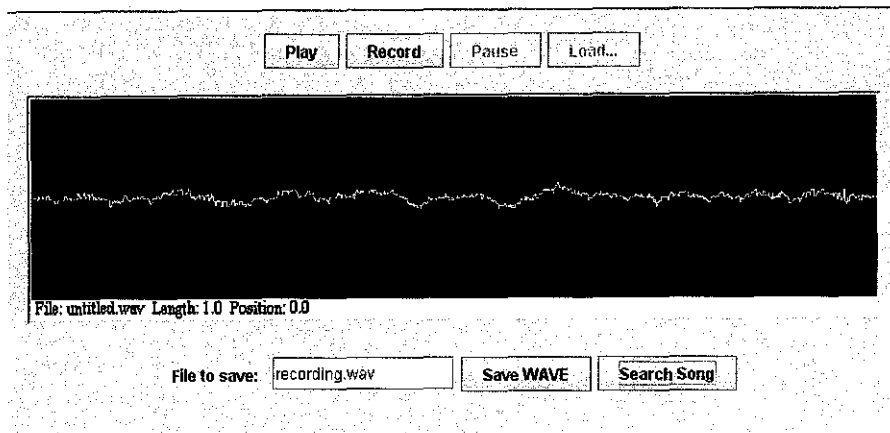
The user interface of the system is divided into two parts which are:

3. Record Humming
4. Result

### 3. USING THE SYSTEM

The first step that the users do to use the system is by recording the humming voice to the melody of the song intended to be searched inside the system.





**Figure 4.2: Record Humming Field**

In the record humming part, it is the step where user will record his/her humming voice of a song. User will push the 'Record' button to start recording and push 'Stop' button once user finishes recording. The user can playback the recorded humming voice and try recording again if not satisfy before proceeding to save the file into 'wav' format inside the system.

Next, the user proceeds to having the result. When the 'Search Song' button is clicked, the system will start processing the recorded humming voice. The system will perform the comparing algorithm to find the exact song from database and match it with the humming melody. Once the match occurred, the result will be displayed in the result area with percentage of similarity of the humming voice melody compared to every song in the system.

Song Number 1:	Mary Had A Little Lamb.midi	11.2%
Song Number 2:	Old MacDonald Had A Farm.midi	8.3%
Song Number 3:	Twinkle Twinkle Little Star.midi	5.1%
Song Number 4:	When The Saints Go Marching In.midi	100.0%

**Figure 4.3: Result Display**

## **CHAPTER 5**

### **CONCLUSION AND RECOMMENDATION**

From the result of the development of the project, it is concluded that the development of this system has met the objectives of its result as planned. Based on the objectives that had been discussed earlier, the functions planned to be implemented to the system has been tested and work successfully.

The system developed is found to be useful to the user as it helps user to find song just by humming to the melody of the song intended to be search instead of going to each an every song in the system and play all the songs to find the right one.

The use of the system is easy and less time consumption and it is also capable to be installed in any computer that has Java environment. The use of java language for the programming development has also help to diversify the technology that can be used as Java is expandable.

For the current system, it has rooms for improvement for example future work can be done in several areas such like:

a) **Humming Voice Melody Capturing**

In the future, more complex multimedia and sound programming can be implemented into the system in order to have humming that can be altered to produce exact pitch similar to pitch of musical notes so that error on having the wrong note will not occur.

b) 'wav' to 'midi' Conversion

The current system implemented the process by using an outside program to convert 'wav' to 'midi' for humming voice melody. Therefore in the future expansion can be done to this system so that the conversion takes place inside the system.

c) Enhancement of Comparing Algorithm

This system uses Levenshtein Distance for comparing 'midi' file of humming voice melody and the stored songs melody to find similarity of both melodies. Future work on this aspect can focus on enhancing the system capability to find songs with different key signature. In example is that even though humming voice melody is hummed in C Major, it can be compared to find similar song that is stored in D Minor or other key signature.

It is hoped that the development of the system can help its users in any way possible. A lot of efforts had been put on in realizing the project and it is also hoped that anyone who encountered with this project whether a user or a developer, would benefit and gain a lot of knowledge from the system.

## REFERENCES

1. Dennis A., Wixom B. H., Tegarden D., *System Analysis and Design with UML Version 2.0, Second Edition*, John Wiley & Sons, Inc., 2005
2. Paul J. 2007, *Java How To Program*, Upper Saddle River, New Jersey, Pearson Prentice Hall
3. Bell, Doug. 2006, *Java for Students*, Harlow, England, Pearson Prentice Hall
4. Horstmann, Cay. 2005, *Java Concepts*, New Jersey, John Wiley & Sons
5. Miranda, Eduardo Reck. 2002, *Computer Sound Design: Synthesis Techniques and Programming*, Oxford, Focal Press
6. McCarthy, Bob. 2007, *Sound System: Design and Optimization*, Boston, Focal Press
7. MOT Press. 1999, *The Csound Book : Perspectives In Software Synthesis, Sound Design, Signal Processing And Programming*, Cambridge, The MOT Press.
8. Java, Sun Microsystems. 24 October 2001, "JavaSound API Programmer's Guide"  
<[http://java.sun.com/j2se/1.4.2/docs/guide/sound/programmer\\_guide/index.html](http://java.sun.com/j2se/1.4.2/docs/guide/sound/programmer_guide/index.html)>
9. David Koelle, 2007, "Features: JFugue, Java API for Music Programming"  
<<http://www.jfugue.org/features.html>>

10. Kim, Youngmoo E. and Whitman, Brian. 2004. Singer Identification in Popular Music Recordings Using Voice Coding Features
11. AKoff Sound Labs, 1998-2001, Wav to Midi Conversion  
<<http://www.akoff.com/about.html>>

# A Voice-to-MIDI System for Singing Melodies with Lyrics

Naoki Itou and Kazushi Nishimoto

Japan Advanced Institute of Science  
and Technology

1-1, Tatsunokuchi, Nomi, Ishikawa,  
923-1292, Japan

Phone: +81 761 51 1812

{n-itou, knishi}@Jaist.ac.jp

## ABSTRACT

In this paper, we propose a robust Voice-to-MIDI (V to M) system with which a user can input MIDI sequence data by naturally singing melodies with lyrics. A Voice-to-MIDI system translates singing voices into digital musical data, i.e., MIDI sequence data. Therefore, with such a system, users can input melodies intuitively, which releases them from manual translating memorized melodies into chromatic pitches. However, the quality of translation of ordinary Voice-to-MIDI systems is insufficient. One of the most significant problems is the poor accuracy of the segmentation of notes. We solve this problem by employing "rhythmic tapping" concurrently with singing. We examined the proposed method by the accuracy of the numbers of segmented notes and their pitches. As a result, we confirmed that our system outperformed ordinary Voice-to-MIDI systems. Thus, this system satisfies both of easy and intuitive composition of MIDI sequence data and high accuracy of translation of sung data into MIDI sequence data.

## Categories and Subject Descriptors

H.5.5 [Information Interface and Presentation]: Sound and Music Computing – *Signal analysis, synthesis, and processing.*

**General Terms:** Algorithms, Performance, Experimentation, Human Factors.

## Keywords

Voice-to-MIDI, melody input, note segmentation, pitch recognition, lyrics, tapping, FFT.

## 1. INTRODUCTION

Owing to improvement of computer power and technologies of musical information processing, computer-based music production has become a very popular entertainment activity as a hobby. However, existing systems still require users a lot of musical knowledge and skills. As a result, not few people eventually give up using the systems before they enjoy music creation. To solve his situation, in this paper, we propose a novel and robust Voice-to-MIDI system with which even a user who has little musical

knowledge and skill can input MIDI (Musical Instruments Digital Interface) sequence data by naturally singing melodies with lyrics.

A Voice-to-MIDI system translates singing voices into digital musical data, i.e., MIDI sequence data, which is used in computer-based music production, for example, composition, transcription, and ringing melodies for cell phones. With the Voice-to-MIDI system, users can readily input the MIDI sequence data of any melody into computers by simply singing the melody with a microphone: the Voice-to-MIDI system automatically extracts such acoustical characteristics as pitch, volume, and duration from the sung voice data and translates them into a time series of musical notes for building music scores. Thus, users are freed from manually translating their memorized melodies into chromatic pitches.

However, there is a significant constraint in ordinary Voice-to-MIDI systems: users must adhere to special ways of singing to obtain reasonable results. For example, some Voice-to-MIDI systems, e.g., [1] and [2], ask users not to expressively sing with lyrics but to sing plainly with "ta ta ta ..." Expressively singing causes many inaccurate translations. Thus, users are forced to sing unnaturally. To allow users to exploit the Voice-to-MIDI system more naturally and intuitively, they must be freed from unnatural constraints. Such constraint has been incorporated into ordinary Voice-to-MIDI systems that are mainly for achieving accurate note segmentation. It is very difficult to accurately segment notes from natural singing data. Furthermore, even if users sing with "ta ta ta ..." errors remain.

We solve this segmentation problem by incorporating a "rhythmic tapping" function into our Voice-to-MIDI system. In other words, we divorce handling of inputting separate notes from "singing." Users of our system can naturally sing while concurrently inputting the melody's rhythm by finger tapping. We examined the proposed method for accuracy of the numbers of segmented notes and pitches by comparing it with two Voice-to-MIDI products. As a result, we confirmed that our system outperformed ordinary Voice-to-MIDI systems. Additionally, we found that rhythmic tapping did not impose extra loads on users, in particular those without experience performing musical instruments.

The rest part of this paper is organized as follows. Section 2 reviews related works and compares them with our proposed system. Section 3 describes our proposed method and the prototype system. Section 4 describes experiments to estimate our proposed method and discusses its effectiveness by comparing it with two Voice-to-MIDI products. Section 5 concludes this paper.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACE'07, June 13–15, 2007, Salzburg, Austria.

Copyright 2007 ACM 978-1-59593-640-0/07/0006...\$5.00.

## Aka Tombo (Red Dragonfly)

Lyrics: Rofu Miki  
Music: Kosaku Yamada

The image shows a musical score for the song 'Aka Tombo'. It consists of two staves of music in G major and 2/4 time. The first staff has the lyrics 'Yu - ya ke Ko ya ke - no A ka to n bo' and the second staff has 'O wa re te Ni ta no - wa - I tsu no - hi - ka'. The notes are simple eighth and quarter notes.

Figure 1. Score of "Aka Tombo"

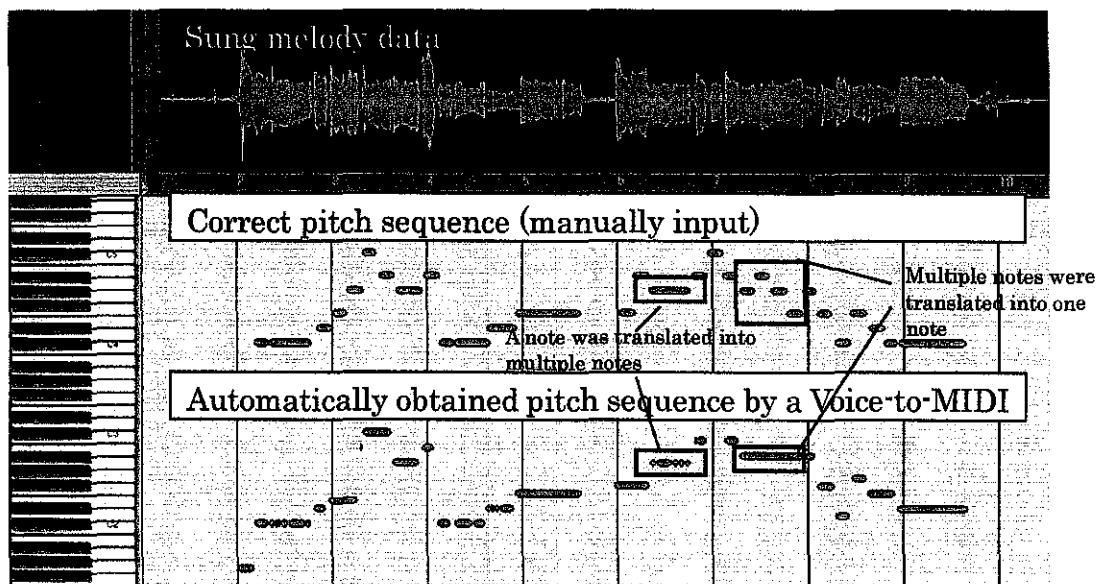


Figure 2. Sample translation result of "Aka tombo" by a product Voice-to-MIDI system.

Each note in "Automatically obtained pitch sequence" is not quantized by beats or metrics.

All notes in "Correct pitch sequence" were manually inputted by mouse.

## 2. RELATED WORKS

[3] describes estimate pitch for an automatic scoring system by Voice-to-MIDI. The system does not use a musical scale based on absolute frequency position to pitch estimation, but instead uses a scale based on a singer's own musical scale. The target of the system is humming, not singing with lyrics. [4] is a commercial product to input MIDI sequence data with high precision for pitch contours and expressions. Our target, however, is to input notes in chromatic precision for musical score making.

Concerning systems that treat the processing of voice/vocal signals or inputting MIDI sequence data, [5] describes a system that inputs MIDI sequence data by voice commands. But it also requires musical ability and knowledge to input melodies.

Researches that generally treat the processing of vocal signals include vocal detection [6], chorus detection [7][8], lyric recognition [9], and so on. One research that integrated these technologies, [10], automatically aligns acoustic musical signals with textual lyrics. But it only extracts the vocal segments, not the precise note information.

## 3. METHOD AND PROTOTYPE

### 3.1 Segmentation Failures of Ordinary Voice-to-MIDI Systems

First, we examined how and why ordinary Voice-to-MIDI systems cannot properly translate singing with lyrics into notes. Using a Voice-to-MIDI product, the first author input the melody data of "Aka Tombo" (a popular traditional Japanese children's song with

yrics by Rofu Miki and music by Kosaku Yamada) by singing its lyrics. Aka Tombo's score is shown in Figure 1. A sample of translation results of an ordinal commercial Voice-to-MIDI system is shown in Figure 2. Here, the top row shows the acoustic waves of the input sound melody data, the middle row shows the correct pitch sequence manually input by mouse in a piano-roll format, and the bottom row shows automatic translation results of the sung melody data into pitch sequence using the Voice-to-MIDI product. Note that the correct pitch sequence (the middle row) is shifted two octaves higher than their real pitches for viewing convenience.

According to Fig. 2, the sample Voice-to-MIDI system failed to segment many notes. For example, multiple notes were translated into only one note, while one note was translated into multiple notes. These phenomena should relate to translation errors in ordinary systems. The translation process from singing data to MIDI note data is hierarchical, and each step is chained: the quality of the extracted or calculated data in the former steps directly affects the quality of the latter steps. Therefore, if note segmentation failed in the first step, it causes the failure of pitch estimation in the next step, and finally, wrong notes are output.

To alleviate this problem, most ordinary Voice-to-MIDI systems ask users to sing in a discrete "ta ta ta" manner to detect the boundaries of notes more accurately. Indeed, accuracy improves if user sang in a "ta ta ta" manner. However, accuracy is still insufficient, and so users are forced to sing unnaturally. A method is required that accurately segments notes from naturally sung melody data.

## 3.2 Prototype System Setup

We attempt to solve this problem with a very simple method: concurrently inputting note segmentation information of melody with singing. Note segmentation information is generated by the rhythmic tapping of button(s), e.g., the key(s) of a PC keyboard or of a MIDI musical instrument. Figure 3 shows the differences between the ordinary and proposed methods. Our proposed method not only accurately segments the notes but also avoids wrong influences from noise, e.g., coughing and talking voices.

The input data are acoustic singing voice data and rhythm tapping data. The sampling quality of singing voices is 44.1 KHz with 16 bit monaural quality. We use the push-down and release timings of one specified key of the MIDI keyboard as rhythm tapping data; we use the input timings of MIDI note on message and note off message. Output is a sequence of chromatic note names between C2 and G4 ( $A4 = 440$  Hz). The output note names are figured in real-time (online processing). At present, the prototype does not output complete MIDI sequence data; only pitch sequence is output. However, it is easy to obtain note length data by using the rhythm tapping data.

In experiments, this prototype can record the timings of the push-down and release of the key, the time series of instantaneous pitches (pitch contour), and the input of singing waves. The base development environment is Microsoft Visual C#2005. A DLL module of instantaneous pitch calculation was built by using Visual C++2005, and the sound recording part is built on DirectSound.

Figure 4 shows the computation process flow in the prototype system. When the user pushes down a key (inputting "MIDI note-

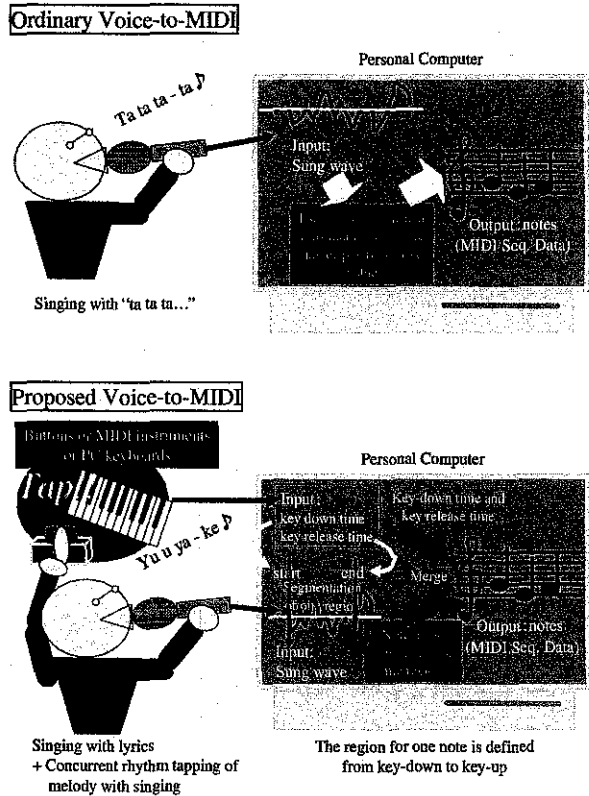


Figure 3. Comparison of ordinary Voice-to-MIDI and our proposed methods

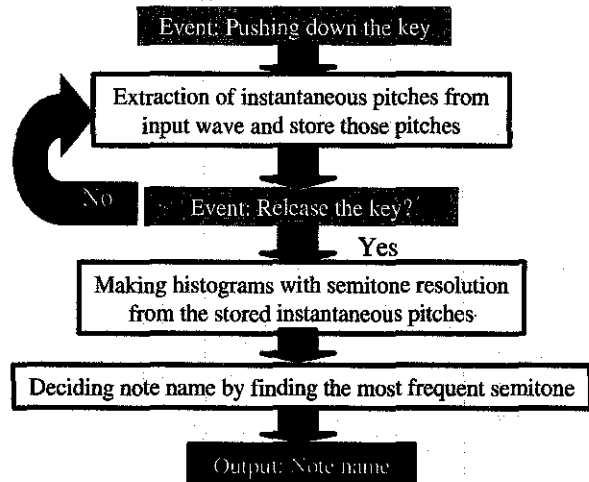


Figure 4. Flow of note name estimation

on" data) as rhythmic tapping, the system starts to extract instantaneous pitches from the stream of the acoustical recordings for making pitch contour until the user releases the key (inputting "MIDI note-off" data). Instantaneous pitch is calculated by the Short Time Fourier Transform (STFT) method for each frame whose size is 4096 samples and the STFT interval is 512 samples. Each instantaneous pitch is obtained at cent precision by



**Table 1. Numbers of correct answers for note name tasks, absolute pitch tasks, and interval tasks**

	Note name	Abs. pitch	Interval
subject A	5	3	3
B	0	0	0
C	5	2	2
D	5	0	0
E	5	1	1
F	5	1	3

interpolation of spectrum [11] and mapped to a corresponding bin of a semitone. For each bin of a semitone, the numbers of mapped instantaneous pitches are counted. Finally, when releasing the key, the system looks for a bin where the number of mapped instantaneous pitches is maximum, decides the corresponding semitone is the pitch of the period, and output its chromatic note name.

#### 4. EXPERIMENTS

To estimate the effectiveness of the proposed Voice-to-MIDI method, we conducted user studies. We compared our prototype system with two Voice-to-MIDI products: XGworks ST [1] (YAMAHA Co., Ltd.) and SingerSongWriter Lite5 [12] (INTERNET Co., Ltd.). This section describes the experiment procedure and results and discusses the advantages and problems of the proposed method.

##### 4.1 Procedure

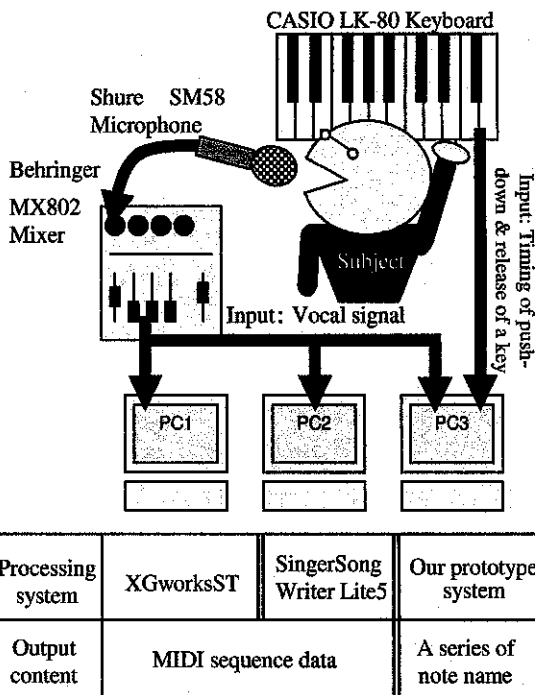
As subjects, we employed six male students at the authors' institute. The following are the musical experiences of each subject:

- Electric organ for nine years
- None
- Piano for five years and guitar for ten
- None
- Drums and guitar for five years
- Voice, guitar, bass, and drums for eight years

Before the experiment, we first examined their musical knowledge and ability using the list of questions below.

- Note name task: while pointing to a keyboard key we asked the subjects: "what note is output when this key is hit?"
- Absolute pitch task: we asked for the note of a tone to which subjects listened without being able to see which key we hit.
- Interval task: we asked the subjects to sing relative pitch against a given tone. For example, we presented a C tone and asked them to sing a note that is a perfect 5<sup>th</sup> higher.
- Tapping task: we asked the subject to sing "Ware wa Uminoko" (I am a son of the sea), a very popular traditional Japanese song while tapping the melody's rhythm.

The tapping task checked the fundamental ability of concurrent singing and tapping. All subjects could concurrently perform them.



**Figure 5. Signal routing diagram from the inputs to the outputs for experiment**

We provided five questions for each task except for the tapping task. Table 1 shows the results for all subjects.

Table 1 indicates that subjects A and C might have absolute pitch recognition, while subjects B and D did not have sufficient musical ability due to their lack of musical experience. Although the target users of our proposed method are mainly non-skillful users in pitch recognition, we employed not only non-skillful subjects but also relatively skillful individuals in pitch recognition for a complete evaluation of our method.

All experiments were conducted in a soundproof room. Figure 5 shows a snapshot of the experiment. We prepared three PCs so that each Voice-to-MIDI system ran on an independent PC. The input singing data from the microphone were distributed to each PC. In addition, the tapping data (push-down and release key information of a MIDI keyboard for note segmentation information) were fed to the PC on which our system ran. Namely, the two baseline systems (XGworks ST and SingerSongWriter Lite 5) only dealt with singing data, while our proposed system dealt with both singing and tapping data. XGworks ST is equipped with configuration menus for translation algorithms. We set the target range of notes between E2 and G4 and turned off the "quantizing" and set precedence scale as a diatonic mode. SingerSongWriter Lite5 is not equipped with any configuration menus.

The set piece for this experiment was "Aka Tombo" (See Section 3.1 and Figure 1), which consisted of 31 notes. This piece is very popular in Japan, and since all subjects knew it very well, they had no difficulty singing it. However, we also let the subjects listen to the song on CD three times. Then each subject sang the melody three times and concurrently tapped its rhythm while singing at his

**Table 2 Results of experiment**

subject		Interval (Number of semitones)														Multi- ple	Miss- ing	Extra	precision (%)	
		0	1	2	3	4	5	6	7	8	9	10	11	12	over 12					
A	Prop.	52	19	4	5	3	0	0	0	0	0	2	1	7	0	0	0	0	55.91	
	XGW	33	16	5	2	1	0	0	0	0	0	0	0	0	0	0	1	35	10	35.48
	SSW	8	26	6	3	2	2	1	0	0	0	0	0	0	0	0	1	44	0	8.60
B	Prop.	67	1	0	2	3	0	0	2	0	0	2	0	14	0	0	2	0	72.04	
	XGW	27	13	2	1	0	0	0	0	0	0	0	0	0	0	0	0	50	1	29.03
	SSW	18	11	8	4	2	0	0	0	0	0	0	0	0	0	0	0	50	0	19.35
C	Prop.	66	7	5	4	0	0	1	0	0	0	0	0	2	0	0	8	0	70.97	
	XGW	27	28	3	0	0	0	0	0	0	0	0	0	0	0	0	35	30	29.03	
	SSW	2	6	9	8	8	3	4	0	3	1	0	2	0	4	12	31	18	2.15	
D	Prop.	52	13	2	4	1	0	0	0	1	0	1	5	14	0	0	0	0	55.91	
	XGW	21	17	16	3	0	0	1	0	0	0	1	0	0	0	3	31	17	22.58	
	SSW	8	12	7	4	4	4	0	0	0	0	0	0	0	0	0	54	0	8.60	
E	Prop.	50	11	4	2	2	0	0	0	0	0	0	0	2	0	0	22	0	53.76	
	XGW	9	14	11	0	2	0	0	0	0	0	0	0	0	0	1	56	2	9.68	
	SSW	14	9	7	9	1	1	0	0	0	0	0	0	0	0	0	52	0	15.05	
F	Prop.	81	4	3	1	0	1	0	0	0	0	0	0	1	0	0	2	0	87.10	
	XGW	16	22	2	2	0	0	0	0	0	0	1	0	0	0	0	50	2	17.20	
	SSW	7	9	11	7	3	2	0	0	1	0	0	0	0	0	1	52	0	7.53	

- The sum of the numbers from “Interval 0” to “Missing” is always 93 notes.
- “Precision (%)” = (the number of “Interval 0”) / 93 \* 100

own tempo. We did not specify a key. Subjects were allowed to see the printed lyrics while singing, although we did not show them a musical score throughout the experiment. Finally, we asked them to answer questionnaires.

## 4.2 Results

We evaluated two aspects for the measurement of the quality of translations: accuracy of pitch discrimination and number of recognized notes. To purely evaluate the performance of the Voice-to-MIDI systems, we must clearly distinguish whether pitch discrimination failure was caused by subject or system mistakes.

Hence, the first author, who has musical experience including three years of chorus and 18 years of musical composition, manually decided chromatic pitches for every note by listening to the acoustical singing waves of all experiments to obtain actual sung data sets. If impossible to decide only one note name for a certain tone, all candidate note names were selected. We evaluated the accuracy of pitch discrimination by comparing the actual sung data sets with the automatic translation results of each system on a line. We classified each recognized note by pitch interval between the recognized note and the corresponding actual note and counted the notes for all classes. Each subject sang 31 notes with no missing notes as well as extra notes in each attempt: a total of 93 notes were input three times by the singing of each subject.

Table 2 and Figure 6 show the results of the number of classified notes. There are 17 categories: chromatic intervals from 0 to 12 and over 12, and three more categories, i.e., “Multiple,” “Missing,” and “Extra.” “Multiple” is where one correct note was recognized as multiple notes; “Missing” is where a correct note

was not recognized; “Extra” is where an unexpected note not included in the actual sung data set was recognized, but it does not include cases of “Multiple.” “Precision,” shown in the upper left of Table 2 and Fig. 6, is calculated by dividing the numbers from the “pitch interval is 0” category by the actual note total, i.e., 93.

## 4.3 Discussion

According to Fig. 6, the precision of pitch discrimination of our prototype is much higher than others for all subjects. The average precision scores for our prototype are 65.9% and 24.0% for XGworks ST and 10.2% for SingerSongWriter Lite5. There are significant differences at the 0.1% rate between our prototype and XGworks ST and our prototype and SingerSongWriter Lite5 by two-tailed t-test. In the “Missing” category, the numbers of missing notes of our prototype are less than others for all subjects. There are significant differences at the 0.1% rate between our prototype and XGworks ST and our prototype and SingerSongWriter Lite5 by two-tailed t-test. There are no notes in the “Extra” for our prototype. Fundamentally, extra notes cannot be input by our system unless the subjects input extra taps. Thus, these results clearly demonstrate that our proposed method is promising as a practical Voice-to-MIDI system that allows users to naturally sing melodies with lyrics.

However, some problems remain in our method. The number of “Missing” notes of Subject E is relatively large because Subject E did not tap the boundaries of legato notes. There are many “1 octave” error estimations (included in the “12 semitone” category). These errors reflect the poor performance of the algorithm in extracting instantaneous pitches. This problem can be easily solved by replacing it with an already established more sophisticated pitch extraction algorithm, e.g., cepstrum analysis,

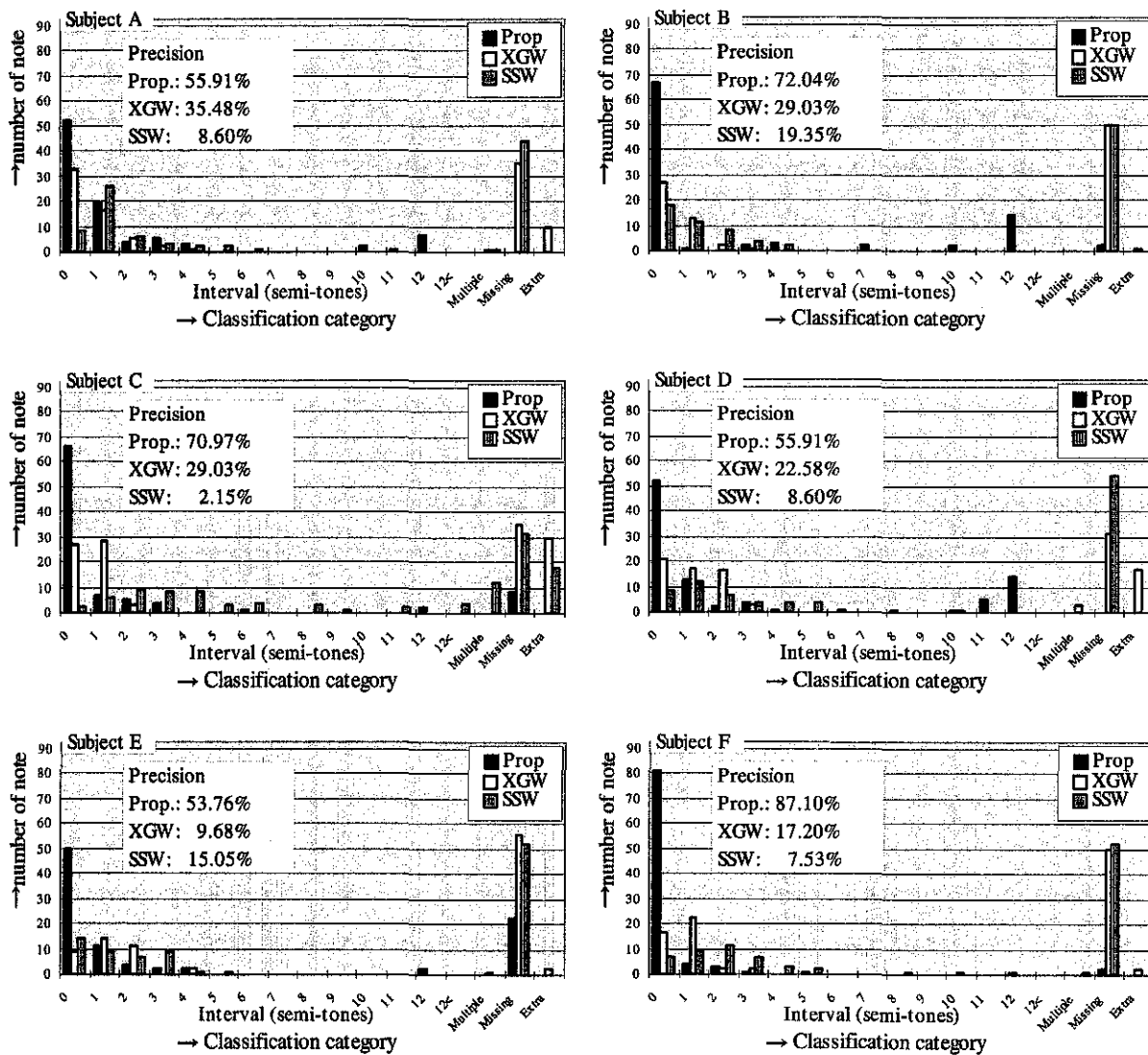


Figure 6. Results of the experiment for each subject

linear Predictive Coding and STRAIGHT[13]. Despite the inadequacy of the algorithm for instantaneous pitch extraction, our proposed method achieves high quality in both estimation of the accuracy of pitch discrimination and the number of recognized notes. This suggests the extremely high potential of our proposed method.

Finally, we enumerate some comments from subjects about our prototype system. Four subjects who have musical experience commented as follows:

- “I confused the timings of tapping.”
- “It was slightly difficult to tap along with the rhythms of the melody.”
- “I didn’t know whether I could sing correctly because the system provided no feedback.”

- “To keep the tempo, I hit the key where there are no note boundaries.”

These comments suggest that it might be difficult for those with musical experience to tap along with the melody’s rhythm because they are apt to tap the “tempo” out of habit.

In contrast, two subjects without musical experience commented as follows:

- “Because I could sing while tapping the same melody’s rhythm, I didn’t feel any discomfort.”
- “I usually don’t sing while tapping, but I felt that it made singing easier.”

Thus, those with little musical experience accepted our proposed method. Voice-to-MIDI systems were originally designed for

hose with little or no musical experience; normal musical instruments are more useful for people with musical experience. Therefore, we conclude that the design of our proposed method is reasonable because people with little or no musical experience accepted concurrent tapping without much discomfort.

## 5. CONCLUSION

We proposed a robust Voice-to-MIDI method by singing melodies with lyrics while concurrently tapping the rhythm of melodies. The experiment results showed that in our proposed method, the accuracy of pitch discrimination and the number of recognized notes is more accurate. For the accuracy of pitch discrimination, our prototype is 1.5–5.0 times higher than ordinary systems. According to comments from subjects, although those who can play musical instruments tend to have difficulty tapping along with the melody's rhythm, subjects who cannot play any instrument felt the melody rhythm tapping is not difficult and actually preferred it. The proposed method archived high accuracy with an additional "not so hard" task to input note segments, i.e., rhythm tapping, as well as freed the users from singing in an unnatural manner. Thus, this system satisfies both of easy and intuitive composition of MIDI sequence data and high accuracy of translation of sung data into MIDI sequence data. Therefore, we conclude that the proposed method is a promising method for Voice-to-MIDI systems. You can find some sample MIDI sequence data composed by our prototype system at: <http://www.jaist.ac.jp/~n-itou/>

In the future, we would like to adopt more sophisticated algorithms to extract fundamental pitches and estimate note names. We also want to adopt the prototype for songs with non-Japanese lyrics.

## 6. ACKNOWLEDGMENTS

This research was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research (C), 16500580, 2004

## 7. REFERENCES

- [1] YAMAHA Corp., XGworks ST, <http://www.yamaha.co.jp/product/syndtm/p/cmp/xgwstw/index.html>
- [2] Media Navigation, Inc., Hanauta Musician 2, <http://medianavi.co.jp/product/hana2/hana2.html>
- [3] Jun, S., Takeshi, M., Masanobu, M. and Masuzo, Y., Automatic Scoring of Melodies Sung by Humming. *Tech. Rep. Musical Acoust. Acoust. Soc. Jpn.*, Vol.23, No.5, pp.95-100, 2004.
- [4] Epinoisis Software, Digital Ear, <http://www.digital-ear.com/digital-ear/index.asp>
- [5] Lloyd A. S., Eline F. C., Brian L. S., A Speech Interface for Building Musical Score Collections. *Proceedings of the fifth ACM conference on Digital libraries*, 2000.
- [6] Goto, M., SmartMusicKIOSK: music listening station with chorus-search function. *Proc. of the 16th annual ACM symp. on User interface software and technology (UIST)*, 2003.
- [7] Goto, M., A Chorus-Section detection Method for Musical Audio Signals. *Proc. of IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, 2003.
- [8] Tzanetakis, G., Song-specific bootstrapping of singing voice structure. *Proc. of the Intl Conf. on Multimedia and Expo (ICME)*, 2004.
- [9] Wang, C. K., Lyu, R. Y. and Chiang, Y.C., An Automatic Singing Transcription System with Multilingual Singing Lyric Recognizer and Robust Melody Tracker. *Proc. of EUROSpeech*, 2003.
- [10] Ye, W., Min-Yen, K., Tin L. N., Arun S. and Jun Y., LyricAlly: automatic synchronization of acoustic musical signals and textual lyrics. *Proc. of the 12th annual ACM intl. conf. on Multimedia (MULTIMEDIA)*, 2004.
- [11] Yuichiro, H. and Seiji, I., Frequency Identification by Complex Spectrum. *Soc. Inst. And Cont. Engineering.*, pp.718-723, 1983.
- [12] INTERNET .Co.,Ltd., SingerSongWriter Lite5, <http://www.ssw.co.jp/products/ssw/win/sswlt50w/index.html>
- [13] Hideki, K., Haruhiro, K., Alain de C. and Roy D. P., Fixed Point Analysis of Frequency to Instantaneous Frequency Mapping for Accurate Estimation of F0 and Periodicity, *Proc. EUROSPEECH'99, Volume 6*, 2781-2784, 1999.