Vivace - An Open Music Transcriptor

PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

(Computer Science & Engineering)

SUBMITTED BY

Aju Tom Kuriakose (Reg No. 10014052)

Jesmy Sunny (Reg No. 10014075)

Prince Mathew (Reg No. 10014096)

Rahul Vijayakumar (Reg No. 10014098)

Saintgits College of Engineering

Kottayam-686532, INDIA

(Batch 2010-2014)

Vivace - An Open Music Transcriptor

PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

(Computer Science & Engineering)

SUBMITTED BY

Aju Tom Kuriakose (Reg No. 10014052)
Jesmy Sunny (Reg No. 10014075)
Prince Mathew (Reg No. 10014096)
Rahul Vijayakumar (Reg No. 10014098)

SUPERVISED BY

Dr. RAMANI BAI V
Professor, CSE

Saintgits College of Engineering
Kottayam-686532, INDIA

(Batch 2010-2014)

CERTIFICATE

This is to certify that the work which is being presented in this report entitled "Vivace - An Open Music Transcriptor" in partial fulfillment of requirements for the award of degree of Bachelor of Technology in Computer Science & Engineering and submitted at Saintgits College of Engineering, Kottayam-686532, Kerala is a bonafide record of the work carried out under the supervision of Dr. Ramani Bai V.

Aju Tom Kuriakose

Jesmy Sunny

Prince Mathew

Rahul Vijayakumar

(SUPERVISOR)

(HOD)

(EXAMINER)

ABSTRACT

The ability to understand music score is a basic requirement for learning any musical instrument. The project Vivace is aimed at making this process of understanding music score more easier. For this, computer aided music recognition is used. This is a process of recognizing printed music score and converting it to a format that is understood by computers. This helps to convert the music score to a format which is easily editable by representing digitally. In other words the score could be recognized and played back through computer speakers.
The primary objective of our project is to help people in learning music. This application makes learning process easier and will also help students to improve their ability to play music. Most of the software which reads image as input and creates midi file as output is very expensive software. So one of our aims is to create an open source project for this purpose.
We created a software that takes in almost all image formats and recognizes the image and generates a midi file as output.

Place: Saintgits College of Engineering, Kottayam

Date: May 2, 2014

Aju Tom Kuriakose
Jesmy Sunny
Prince Mathew
Rahul Vijayakumar

ACKNOWLEDGEMENTS

Vivace would have been just a dream if we had not received the help of many dedicated and highly qualified persons during its development. We humbly take this opportunity to express our gratitude and indebtedness to all whom, directly or indirectly had an impact on us for the success of this project.

First of all, we thank God almighty for His grace and blessings for without his unforeseen guidance this project would have remained in dreams.

We would like to thank **Prof. Santhosh P Mathew** (Head of Department, Computer Science and Engineering, Saintgits College of Engineering) for his constant support in every step of our project and for the source of strength in completing this project.

We are grateful to **Dr. Ramani Bai V**, **Dr.Sajeev G.P** and **Er. Milash M Fuvad** of the Computer Science and Engineering Department for providing us all the facilities, valuable suggestions, helpful supervision and encouragement which led to the successful completion of our project.

We are highly indebted to our staff advisors and all the faculties in the department for being a source of encouragement and providing us with valuable guidance and instant help and also for being with us. We would like to express our special words of gratitute to **Dr.Aswan Thomas Elias**(Director, Harp'N Lyre Academia Musica),**Asst.Prof Tina Susan Thomas,Asst.Prof Veena A Kumar, Asst.prof Abraham George**(Dept. of Electrical and Electronics Eng.) and **Asst.Prof Siju K Swamy**(Dept. of Mathematics).

Finally, we acknowledge the invaluable support of our parents, siblings and friends for being a consistent source of help and support, without which Vivace wouldnt have become a reality.

Place: Saintgits College of Engineering, Kottayam

Date: May 2, 2014

Aju Tom Kuriakose

Jesmy Sunny

Prince Mathew

Rahul Vijayakumar

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

The project Vivace, is a software that converts sheet music in the form of an image file to digital representation of music, MIDI. This process of conversation is called Optical Music Transcription. It has been an active research area. In contrast to Optical Character Recognition (OCR) systems, OMR systems are subject to greater complexities as music notation is represented in a two-dimensional space. The horizontal direction can be associated with the note duration (time) whilst the vertical direction can be associated with pitch. There are several applications to OMR systems. Scores sometimes need to be adapted to different instruments (transposed) and having the score in a digital format greatly reduces the time and effort required to do that. Converting music scores in Braille code for the blind is yet another application of an OMR system. Our software converts this image of music notation into a midi file. This enables the conversion of sheet music to an editable form. There are plenty of software available that can edit the midi file which is easier to produce results. But software that take image as input are very few.

1.1 Background

The goal of this chapter is to build the basic foundations needed in order to understand how an OMR system works along with the theory used for the implementation of the optical music recognition system. We first begin by briefly mentioning the underlying concepts of music theory after which two musical typesetting programs will be introduced. An evaluation of current commercial OMR systems is then presented, followed by a description of how OMR systems can be evaluated. We will then look at some issues and problems which make the OMR tricky. Previous research and projects are then summarized in order to gain a certain knowledge in the field of OMR. Finally, some underlying theory which is used in the design and implementation of the OMR system is presented.

1.1.1 Music Theory and Terminology

In order to understand how Optical Music Recognition (OMR) systems work, it is important to have a notion of the underlying concepts in music theory and its associated terminology.

The fundamental music element in a music score is the stave (also referred to as the staff). The stave is composed of five horizontal lines and four spaces in between each line. High notes are written near the top of the stave and low notes near the bottom. Musical notes are placed on both lines and spaces of the stave to show the pitch of the notes. Vertical bars (known as a bar line) are placed in order to separate different measures.

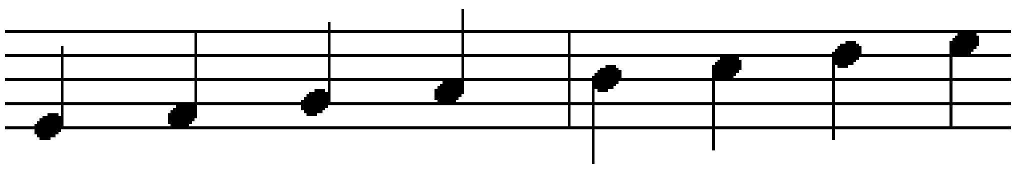


FIGURE 1.1: Notes placed on a stave with a Bar line

There are three types of clefs and they are known as the treble, bass and alto clefs. The clef is commonly placed at the beginning of the stave although it can also appear anywhere in a measure. Polyphonic scores are ones in which multiple independent melodies are present as opposed to monophonic scores in which only one is present.

Piano scores typically use a grand stave which is composed of a bass and treble stave. Notes are placed either on a stave line or in between a stave line. The higher the notes placement on the stave, the higher its pitch. Depending on the type of note, it will be played for a certain duration.Notes are given different name using the first seven letters of the English alphabet: A B C D E F G

A Clef is a sign placed at the beginning of a stave. A Treble clef of G clef is used for high notes. A Bass clef of F clef is used for low notes.

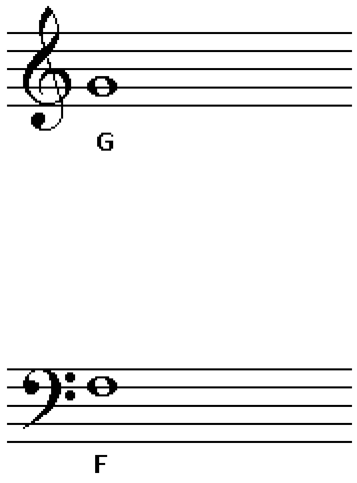
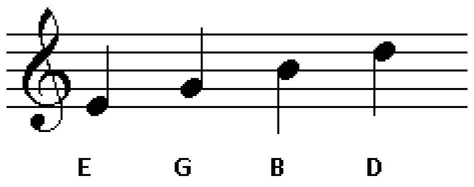


FIGURE 1.2: Treble and Bass clefs

1.1.2 Notes on the Stave



A ledger line is a short line placed above or below a stave to write notes which are higher or lower than the range of the stave. They are spaced at the same distance as the lines within the stave. If two notes have the same letter name but they are in different

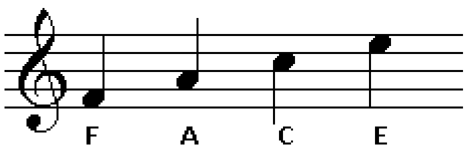
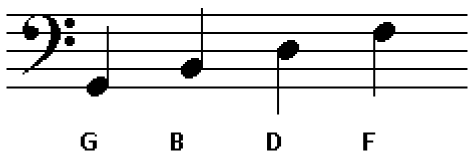


FIGURE 1.3: Notes on the treble stave

positions on the stave, they are said to be at different registers. The distance between one note and the next note with the same letter name is called an octave.



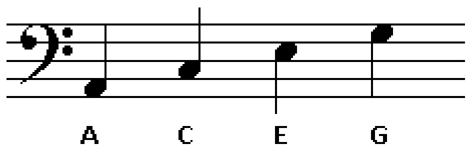


FIGURE 1.4: Note on the Bass Stave

1.1.3 Semitones and Tones

A semitone is the shortest distance between two adjacent notes on the keyboard. It is the distance from one key to the next key , black or white. A tone is made up of two semitones. On the keyboard, a tone is any two keys with one key, white or black, between them.

Figure 1.1 shows a series of notes (left) and rests (right). The number below each note and rest indicates their associated temporal value as highlighted in figure. The notion of time in musical scores is represented by a meter. The meter indicates how many notes of the same type are to be played in one measure and is expressed as a fraction. The

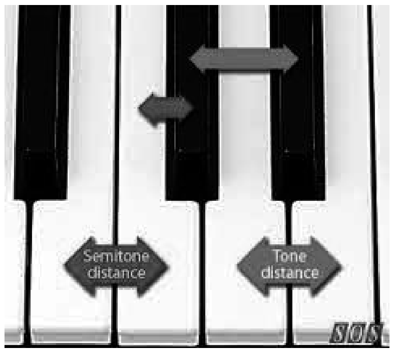
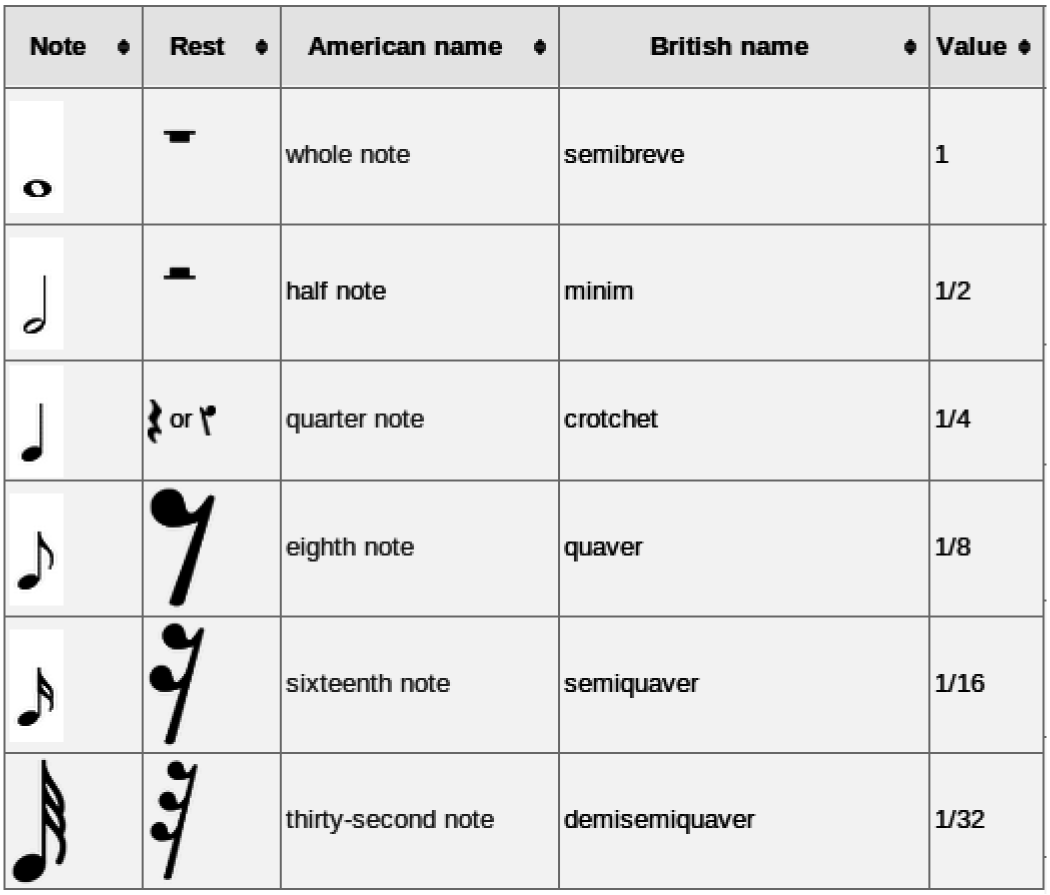


FIGURE 1.5: Semitone and Tone distance



Note	Rest	American name	British name	Value
𝅝	𝄻	whole note	semibreve	1
𝅗𝅥	𝄼	half note	minim	1/2
♩	𝄽 or 𝄾	quarter note	crotchet	1/4
♪	𝄾	eighth note	quaver	1/8
𝅘𝅥𝅯	𝄿	sixteenth note	semiquaver	1/16
𝅘𝅥𝅰	𝅀	thirty-second note	demisemiquaver	1/32

TABLE 1.1: Notes and Rest notations in Sheet Music

numerator indicates how many beats occur in a measure and the denominator indicates how many beats a crotchet receives. A 4/4 beat is also referred to as a common beat and is usually represented by a C instead of a fraction.

1.1.4 Sharps and Flats

A sharp raises a note by a semitone. A flat lowers a note by a semitone. When a sharp or flat appears in a key signature, all the notes on that line or space and all other notes with the same letter name are to be played sharp or flat. The position of sharps in the key signatures of G-Major.

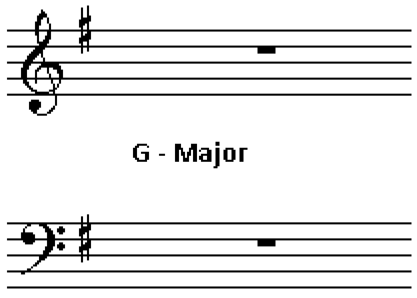


FIGURE 1.6: Position of Sharp in G-Major Scale

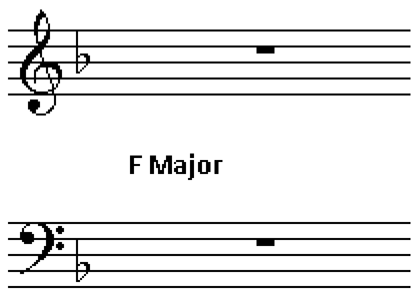


FIGURE 1.7: The position of flats in the key signature of F-Major

1.1.5 Transpose

The shifting of a melody, a harmonic progression or an entire musical piece to another key, while maintaining the same tone structure, i.e. the same succession of whole tones and semitones and remaining melodic intervals. In music transposition refers to the process, or operation, of moving a collection of notes (pitches or pitch classes) up or down in pitch by a constant interval. Transpose= +1, Shifts the value of all the notes in the music one semitone higher.

1.1.6 MIDI-Musical Instrument Digital Interface

It is a technical standard that describes a protocol, digital interface and connectors and allows a wide variety of electronic musical instruments, computers and other related devices to connect and communicate with one another. Advantages of MIDI include compactness (an entire song can be coded in a few hundred lines, i.e. in a few kilobytes), ease of modification and manipulation and choice of instruments.

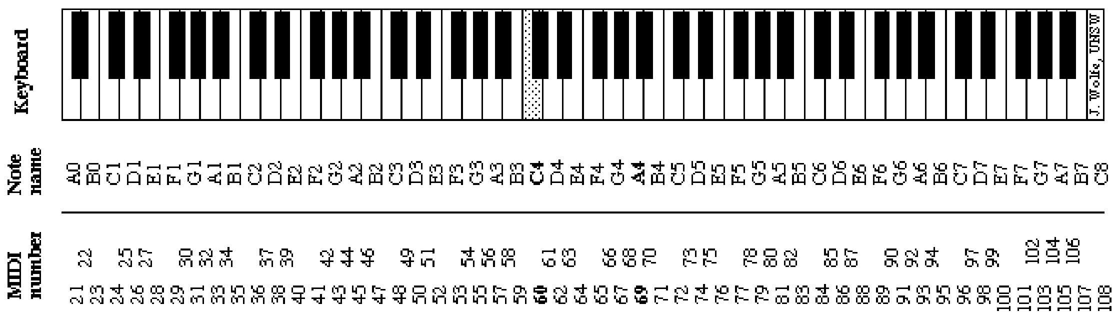


FIGURE 1.8: Midi values of various notes on a keyboard

1.1.7 Music Typesetting

The softwares that are use to write music notes are called scorewriters. There exist several music typesetting packages in the open source community of which two will be briefly mentioned.

- LilyPond is a program that enables musicians to mark up music scores by using a simple ASCII notation as its input. That notation is then processed by the LilyPond program and is able to output a music score in several different formats including the PostScript and EPS formats. Furthermore, LilyPond is capable of reading different file formats including the MusicXML and the MIDI file format. The LilyPond package is extremely well documented and many examples are provided, making it an attractive package to use. LilyPond also runs on many different platforms (Linux, Windows and Mac OS X) and does not require the installation of dependant packages. The installation of the LilyPond package is simple, making it a popular typesetting application used amongst professional musicians.

- MusiXTeX is a suite of open source music engraving macros and fonts that allow music typesetting in TeX. Macros for typesetting music in TeX first appeared in 1987 (MuTeX) and were limited to one-staff systems. In 1991, Daniel Taupin created MusicTeX, whose macros allowed the production of systems with multiple staves, but which presented a few problems in controlling the horizontal positioning of notes. MusicTeX used a one-pass compilation. In 1997 the positioning problems were corrected in MusiXTeX, which includes the external application musixflx to control the horizontal distances. This new module requires a three-pass compilation: TeX, musixflx and TeX again.

1.2 Motivation

The existing softwares that performs music notation transcription are proprietary and these softwares typically costs hundreds of dollars and requires specific file types. An accessible and easy-to-use OMR application could provide an amazing tool for improving the musical education experience.

There is no effective and low cost product for converting image of sheet music to MIDI and the ones which transcripts it into mere sound and shows an average performance is also not affordable. This project Vivace is hence aimed at providing a low cost solution to this problem of music notation transcription. There are numerous companies who have spent a lot of money on developing softwares for music notation transcription and still haven't made a perfect solution for this problem. Hence,we are making this an Open Source Project. So that in future, other people will be able to improve the performance of this software by using better algorithms and make it more effective.

CHAPTER 2

Design

In this chapter, we discuss about the programming language, the packages and brief description of methods we used is described here.

2.1 Overview

This software uses Pipe and Filter architecture to process input and obtain the output. The major components of the system are

- **Read scanned music score.**
  As mentioned earlier, the input is an image file. The sheet music can be scanned from a scanner. For better results, the image must be scanned at atleast 300dpi. This image is then given to the engine to start recognition process.
- **Detect staves**.
  The stave lines are detected by taking the Y-Projection of the image. Certain threshold value is checked. The pixels that exceed that value is considered as a potential stave line. The stave detection is confirmed only after finding 5 similar equispaces pixels on Y axis. This is the most important step in detection process. The performance of the entire system depends on how well the staves are detected.

- **Segmentation and notehead detection.** The input image is converted into black and white as per the user's requirement. The note head is found using RLE algorithm. This is done my checking the black pixel density in an area. The empty note is found by checking alternative consecutive black and white pixels. Due to the requirement in pitch detection module, the empty note head is filled with black pixels. The notes is divided into blocks first and later divided into individual notes.
- **Pitch detection.** The pitch of a note is found from the position of note head with respect to stave lines. The position of the note head is compared with the stave lines' position to find the pitch. The mid of the note is found by checking the black pixel density.
- **Duration identification.** The duration is actually identified from the lines above the notes or flag of note. In this software, each image is compared to some image signatures of all notes. This helps to significantly improve the execution speed.
- **Midi file generation.** The midi generation module is in python. This module creates midi file from values provided as input. The midi values, which is a standard is stored in a file from java.

2.1.1 System Architecture

The simplified view system architecture of the Vivace software system is given in Figure 2.1.1 which can help in understanding the various modules of the system and the interaction between them.

The input of this system is an image file of a sheet music (music notation). The image is converted into black and white if specified using the module 'DoBlackandWhite'. The control flow starts from the module 'Open Music Transcriptor'[1]. This module contains the main function.
From here we pass this image to the module 'Stave Parameters'[2], where the stave parameters of the music notation such as the thickness of stave lines and the distance between the two stave lines are calculated. The program control then returns to the module 'Open Music Transcriptor'[1].
Then the image is passed to the module 'Divide Image'[3], for the initial segmentation

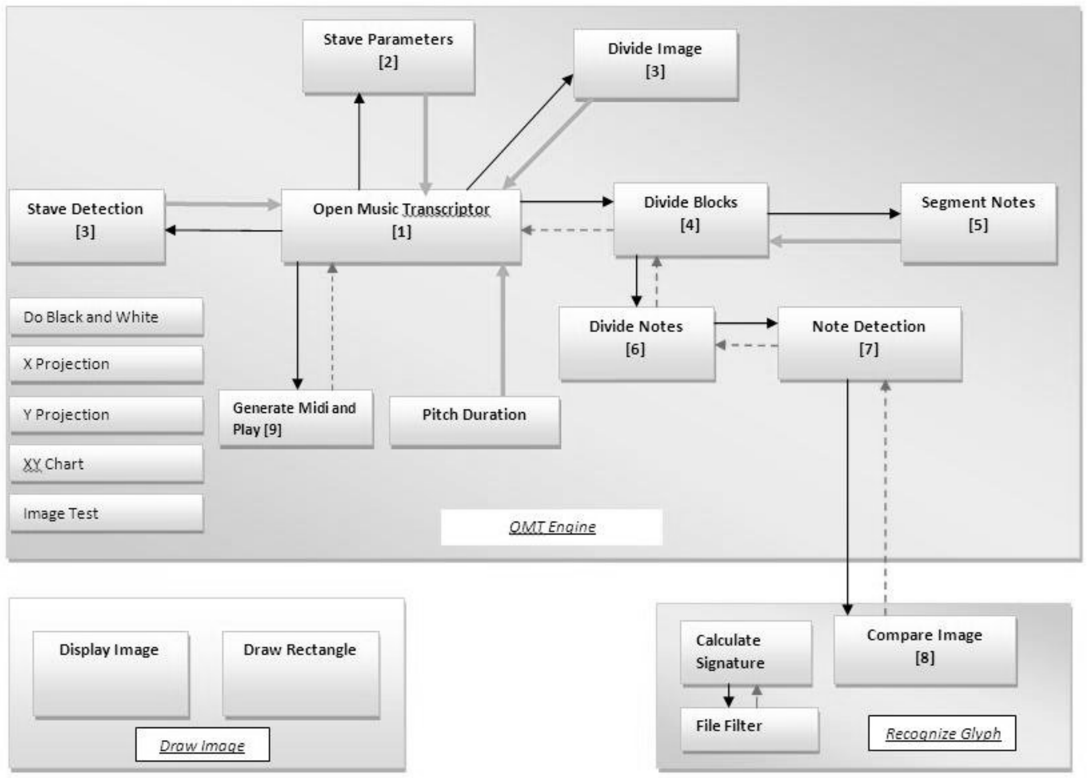


FIGURE 2.1: Simplified view of the Vivace software system

process. Here the image will be divided into independent staves. Now the control is again returned to the module 'Open Music Transcriptor'[1]. Now each stave is passed one after the other to the module Divide Blocks [4]. Here the next phase of segmentation is done, where the given stave is divided further into different segments. A copy of each segment obtained in this module is passed to the module 'Segment Notes'[5]. In this module we do the process of note head detection and filling empty note heads.

The processed image that we will get after passing through this module will be one containing only the note heads in the given segment. The control is again transferred back to the module 'Divide Blocks'[4]. From here we pass the original image obtained after 'Divide Blocks'[4] and the processed image obtained after processing in 'Segment Notes [5] to the module 'Divide Notes'[6].

Here we find the position of each note heads in the processed image and then divide the original block into segments containing individual notes. These segmented notes are then passed to the module 'Note Detection'[7]. Here the pitch of the note is identified based on its relative position to the start of the corresponding staff. Now this segmented notes are again passed to the module Compare Image [8]. Here the image will be normalized

and will be compared with a stored set of values to determine the duration of that note. The control then returns all the way back to the initial module Open Music Transcriptor [1]. From here we invoke the javapy module to generate the MIDI file by passing the start time, duration and pitch of each note. The midi file hence generated will be stored in the computer with the predefined name given by the user through the GUI.
The package DrawImage helps to draw rectangles and to show the various levels of segmentation clearly. The package Recognize Glyph, as we saw earlier, deals with finding the duration of the given note. The main package here is the OMTengine, which performs most of the operations. It contains more modules than we have already discussed. Such as the XProjection, YProjection, XYChart, Image Test etc. which are used in various modules of the OMT Engine for obtaining the expected results.
Each of these modules are later discussed in detail.

2.2 OMT Engine

This package does the note recognition and identification. The modules are described below.

2.2.1 Black and White Conversion

In this module, we convert the input image into black and white. For this, Red, Green and Blue values of each pixel are added. If it is 0, it means that the pixel is black. So a threshold value of 450 is set. Now if the sum of RGB of pixel is greater than 450, then that pixel is converted into white else, it is converted into black.

2.2.2 X Projection

The X-Projection is the projection of an image onto the the X-axis. The result is a vector whose i^{th} component is the sum of all black pixels in the i^{th} column of the image. This is done by adding all black pixels of the column to an array. X projection was used in segmentation and note head detection.

2.2.3 Y Projection

The Y-Projection is the projection of an image onto the Y-axis. The result is a vector whose i^{th} component is the sum of all black pixels in the i^{th} row of the image. This is done by adding all black pixels of the row to an array. Y projection is used in stave line detection.

2.2.4 XYChart

This class will render a chart in the form of a BufferedImage. This displays chart directly in a GUI. This was used for displaying X and Y projections.

2.2.5 Stave Parameters

Here initially we perform a run-length encoding algorithm to find the number of consecutive black and white pixels vertically and store these values in two linear arrays. The arrays are checked for the number which occurs the most number of times and are considered to be the stave line thickness and the distance between the stave lines. We ignore values greater than **100** because there is no chance for distance between stave lines or stave line thickness to be greater than 100 .

2.2.6 Stave Detection

Here we find the positions of every stave in the given notation. We find the maximum value in the Y-Projection and set the stave threshold correspondingly. From **55%** to **88%** of the maximum value depending on the thickness of black pixels.Then we group each set of 5 stavelines together and save their coordinates in an array named stavearray.

2.2.7 Divide Image

In this module we divide the image into buffered images each consisting of a single stave as shown in Figure 2.2. Each buffered image will have **75%** of the stave thickness above and below the stave or till the top or bottom of the image, whichever is the smallest.This is the level 0 segmentation.

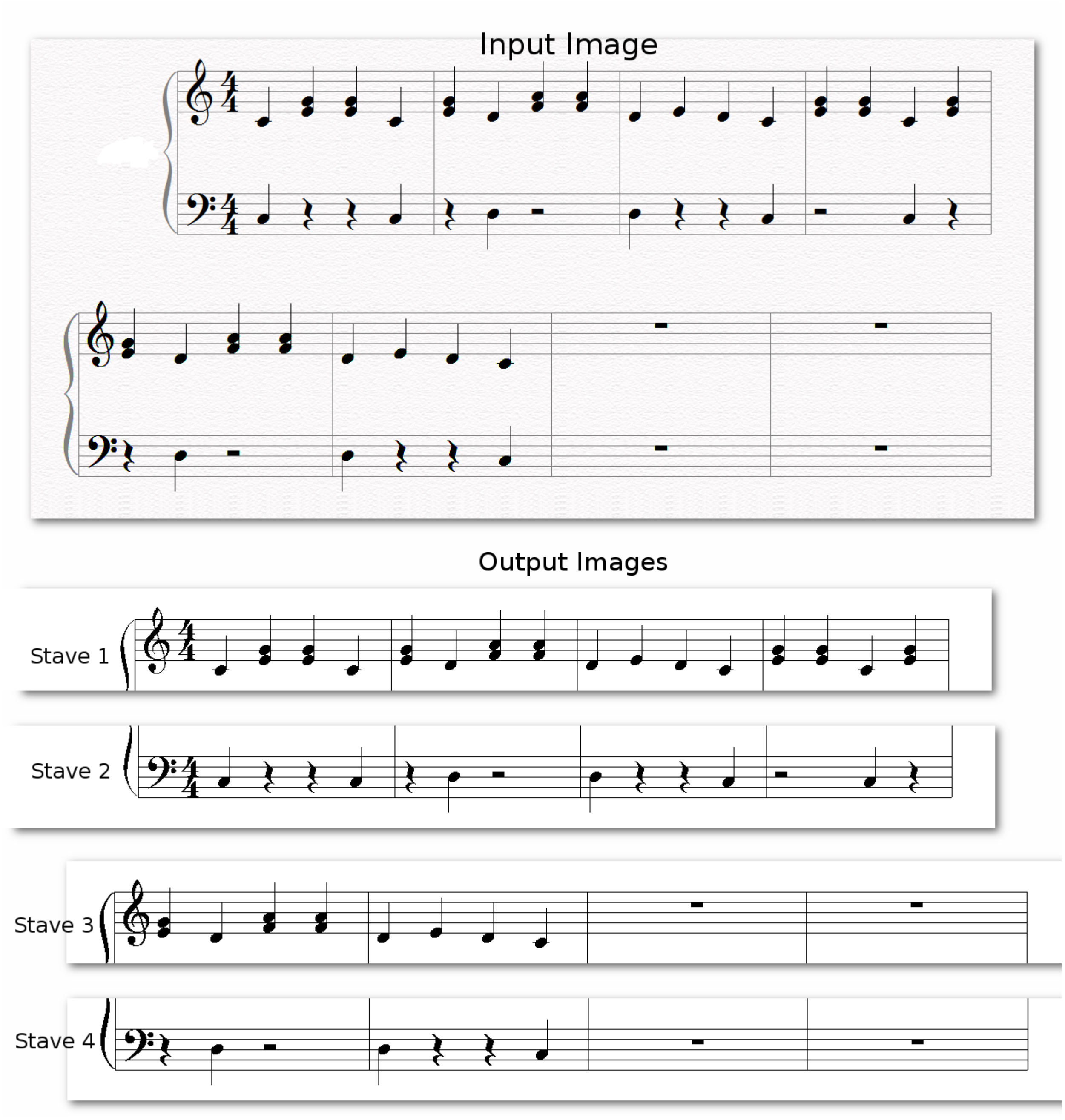


FIGURE 2.2: An Example of image cut into several with respect to staves

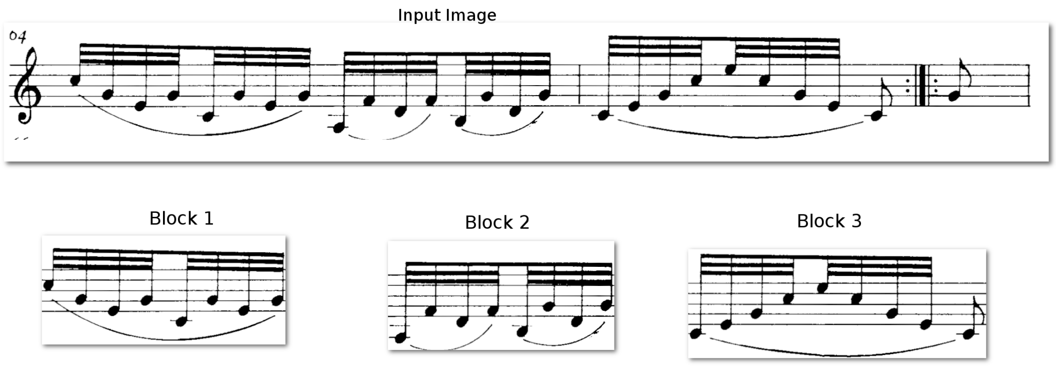


FIGURE 2.3: Dividing Staves into Blocks

2.2.8 Divide Block

Here we take the X-Projection of each stave separately and check if there is something present other than just stave lines.This is the first level of segmentation.Here we set the threshold as

$$5 * stavelinethickness + 2 \tag{2.1}$$

We divide the stave into blocks ,where each blocks X-Projection exceeds the threshold value. The block division is shown in figure 2.3

2.2.9 Segment Notes

Here we detect the noteheads,both empty note heads and filled notes heads in each block and finds the coordinates to segment each blocks into individual modules.For this we take a copy of the buffered image of each block. Here for detecting empty noteheads,we check the buffered image of each blocks length wise using the runlength encoding technique to find continues white pixels,whose number is less than distance between stavelines-2.And replaces those with black pixels.If there is any empty noteheads,then all of them will be filled when this step is completed. Now we just need to check for filled noteheads to find the notehead position.
For this we check the modified buffered image for continues black pixels vertically whose number lies between

$$(distancebetweenstavelines - 2) - 3 * thicknessofstavelines \tag{2.2}$$

and

$$(distancebetweenstavelines - 2) * 4 + 3 * thicknessofstavelines \tag{2.3}$$

Everything which doesnt lie between these values are set to white.
This process is then repeated horizontally where we check for continues black pixels whose number lies between between

$$(distancebetweenstavelines - 2) - 3 * thicknessofstavelines \tag{2.4}$$

and

$$(distancebetweenstavelines - 2) * 2 \tag{2.5}$$

Everything which doesnt lie between these values are also set to white.
Now the image contains only the noteheads. We then find the positions of each of these noteheads and store their start and end position values in an array for further segmentation.

2.2.10 Divide Notes

In this module we divide each blocks into individual modules containing a single note or a chord based on the start and end positions obtained from the SegmentNotes module.And then we pass these bufferedImages containing individual notes to the NoteDetection module.

2.2.11 Note Detection

In this module, we initially find the number of noteheads in each segment and check whether it is a single note or more than one note and then save the midpoint of each notehead into an array. Let 'beg' indicate the start position(first stave lines position) of the stave, 'thic' indicate the stave line thickness and 'dist' indicate the sum of staveline thickness and distance between stavelines.

Here S1,S2,S3 ... are midpoint of stavelines and D1,D2,D3 ... are mid of two stavelines. These are used to calculate the position of noteheads with respect to the stave and hence

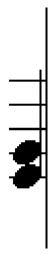
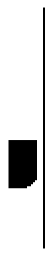
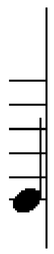
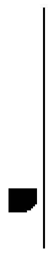
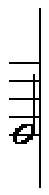
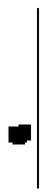
Note	Filled Note
	
	
	

TABLE 2.1: Filling notehead

S17	S18-dist		D17	(S17+S18)/2
S18	S19-dist		D18	(S18+S19)/2
S19	S20-dist		D18	(S19+S20)/2
S20	S1-dist		D0	(S20+S1)/2
S1	beg+thic/2		D1	(S1+S2)/2
S2	S1+dist		D2	(S2+S3)/2
S3	S2+dist		D3	(S3+S4)/2
S4	S3+dist		D4	(S4+S5)/2
S5	S4+dist		D5	(S5+S6)/2
S6	S5+dist		D6	(S6+S7)/2
S7	S6+dist		D7	(S7+S8)/2
S8	S7+dist		D8	S8+dist/2

TABLE 2.2: Detection of noteheads on and off stave lines

is used to determine the pitch of a note. The figure2.4 gives a clear idea about these values.

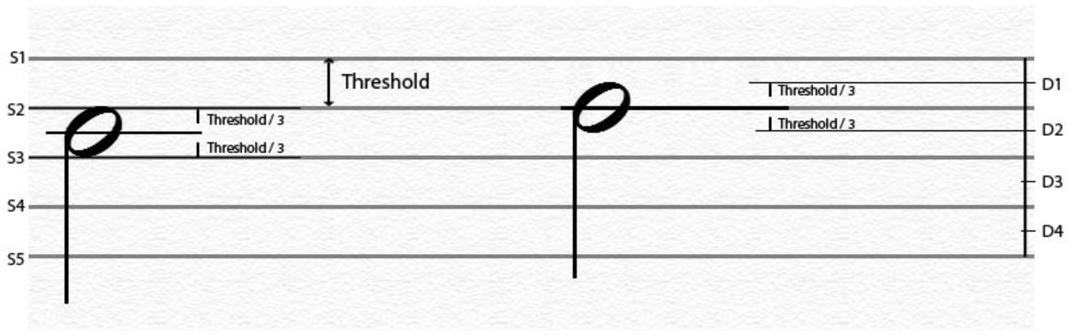


Figure 2.4: Note head identification

While selecting a particular scale ,by specifying the number of flats or sharps,the midi values of the notes given correspondingly in the table will be increased by one.

No. of Sharp	Notes that become Sharp	No. of Flats	Notes that become Flat
1	F	1	B
2	F C	2	B E
3	F C G	3	B E A
4	F C G D	4	B E A D
5	F C G D A	5	B E A D G
6	F C G D A E	6	B E A D G C
7	F C G D A E B	7	B E A D G C F

Table 2.3: Sharp/Flat number and corresponding notes

While doing transpose(+1) the pitch of each note is increased by a seminote.That is, midi value of every note is increased by 1.

2.2.12 Pitch Duration

It defines linked list for storing pitch, duration and position of a note. Add function is used to add values to the linked list. Sizeoflist() returns the size of the linked list. Valu() function returns pitch, duration and position in an array.

2.3 Draw Image

This module is used to display the image in a GUI interface without writing to disk. It was used to see the output of various stages.

2.4 GUI

GUI is an important element when it comes to a software whose users need not have good computer knowledge. In the case of this software, the primary users are people who related to music. So interface must be easy to understand. The GUI of Vivace was made accordingly. The vivace gives a one click interface to obtain the results. Basic music knowledge is required to operate the software.
The GUI gives the options to vary the following:

- tempo
- sharp
- flat
- transpose

The user is able to choose between single and both hand notes. Since the conversion to black and white sometimes varies the quality of recognition, that option is also given to user. So user can decide whether the given input file has to be converted into black and white or not.

2.5 Java Python

2.5.1 Create Midi

This module is used to create a file which acts as the input file to the python module. Then the python file is executed. After the execution of the python module, the generated midi file is played using VLC media player. For the VLC to support the midi, sound font has to be installed. For the proper execution of the commands, in windows, python and VLC system variables has to be created.

2.6 Projection

Used to display the graph of projections in a GUI. This module is also used while the development of the software to analyze the projections and select suitable range of threshold.

2.7 Recognize Glyph

This module is used to find the duration of a note. This is done by comparing the segmented note with image signatures in the software.

2.7.1 Compare Image

This module identifies whether the segment is actually required and if required, its duration.
Each symbols used in a sheet is stored in an array. When a segment is obtained, it is converted into similar signatures. The signatures is taken such that it gives weight is given to the mid of the image.The features for our each image will be 25 RGB triples, corresponding to the average of the RGB values on the 25 regions marked in the figure on the left. No texture or variance feature will be stored, only the color averages. Each region has 30x30 pixels.Each image will be represented, then, a 25x3 feature vector.

To calculate the similarity measure between two images A and B we will take each of the 25 regions, calculate the Euclidean distance between the regions and accumulate.
The distance from A to A will be, by definition, zero. The upper bound (maximum possible distance between two images, using this similarity measure method) is calculated as

$$\sqrt{25 * (255 - 0) * (255 - 0) + (255 - 0) * (255 - 0) + (255 - 0) * (255 - 0)} \qquad (2.6)$$

or a little bit over 11041.
The distance from each symbol is calculated. If the distance is less than 2000 then, it is given a weight according to the distance and the number of images in the array with

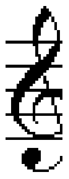
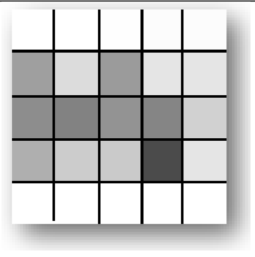
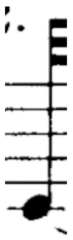
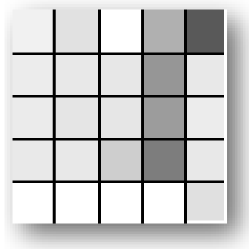
Image before normalization	Normalized Image
	
	

Table 2.4: Images before and after normalization

which it is compared.The weight is as given in table 2.5

Distance	Weight
0	1000
<100	100
<500	25
<1000	10
<2000	3

Table 2.5: Distance and corresponding weights

For example, if the distance is 0, ie the exact image is found, then its weight will be added with 1000.

2.7.2 Calculate Signature

This module is an independent one and is not executes a part of the system. This module is used to calculate the signatures of images in a directory. The signature is nothing but rgb values of a normalized image. Even though this method is not much effective, it gave results. This module gives the signature and number of images in a directory. This can be directly used to initialize a new array (new symbol) or add values to an existing array (existing symbol).

CHAPTER 3

Impementation

3.1 Package Structure

In this section, the implementation of all classes of every packages are explained.

3.1.1 drawimage Package

3.1.1.1 DisplayImage.java

public DisplayImage(BufferedImage image)

This constructor reads image as parameter

public MyCanvas(final BufferedImage image)

In this function, the image is displayed in a jpanel

3.1.2 GUI Package

GUI package was created using netbeans. Java Swings was used to create the GUI. The combination these java functions were used to create GUI.

- JFileChooser() : To choose the input image.
- JFrame() : To create the frame for components.
- JToolBar() : The toolbar which contains different options.
- JPanel() : Panel to hold components.
- JScrollPane() :To create a area with a scroll bar.
- JScrollBar(): To create a scroll bar
- JLabel(): To create a label
- JSpinner(): To create a value showing box with up and down spinner.
- JSlider(): To create a value slider.
- JButton(): To create a button.
- JMenuBar(): To create a menu bar
- JMenu(): To create a menu list
- JMenuItem(): To insert an item in menu.

3.1.3 javapy Package

This package is used to create an intermediate file which stores the values to give as an input for python program. The file is provided as an input to python program which is executed using command. The vlc is then called to play the created midi file.

3.1.3.1 CreateMid.java

public CreateMid(int noofnotes,double[][] note,String opfile,int volume,int tempo)

Writes note, duration and position to a file in disk. This file is given as the input to python module. The python file is run by command and later the midi file is played from vlc.

public static void savef(String filename,String opfile,int volume,int tempo, double[][] note)
Writes initially, the file name, volume and tempo of the midi file to be generated. Then the note, duration and the position is written which is in note[][].

3.1.4 omtengine Package

This is the core package of the project. This package contains the classes that segments the image and identifies the pitch.

3.1.4.1 omtengine.DoBlackandWhite.java

public DoBlackandWhite(BufferedImage buffImage)

Constructor with buffered image as parameter.

public BufferedImage doBW()

This method converts the image passed to the constructor to black and white.

3.1.4.2 omtengine.XProjection.java

public void calcXProjection(int startH, int endH, int startW, int endW)

Calculates the Xprojection of the image with given start and end horizontal as well as vertical positions.

public int[] getXProjection()

Returns an array with the value of XProjection.

public void printXProjection()

Prints the XProjection of the Buffered Image.

3.1.4.3 omtengine.YProjection.java

public YProjection(BufferedImage buffImage)

public void calcYProjection(int startH, int endH, int startW, int endW)

Calculates the YProjection of a given image with the given start and end horizontal as well as vertical positions.

public int[] getYProjection()

Returns an array with the value of YProjection.

public void printYProjection()

Prints the YProjection of the Buffered Image in the console

public int getHeight()

Retuns Height of the Buffered Image

3.1.4.4 omtengine.XYChart.java

public XYChart(int size, int data[], String name) :

Creates chart from the provided array and the provided name is given to the chart.

public BufferedImage getChart(int x, int y) :

Returns the chart as a buffered image.

3.1.4.5 omtengine.StaveParameters.java

public StaveParameters(BufferedImage buffImage)

public void calcStaveParameters(int startH, int endH, int startW, int endW)

Performs run length encoding on the input Buffered Image.

public int[] getW()

Returns an array which contains the output of the run line encoding of white pixels performed on the buffered image.

public int[] getB()

Returns an array which contains the output of the run line encoding of black pixels performed on the buffered image.

public int[] findStaveParameters()

Calculates the stave parameters such as stave line thickness and distance between stavelines.

public void printB()

Print the output of the run length encoding of black pixels performed on the Buffered Image.

public void printW()

Print the output of the run length encoding of white pixels performed on the Buffered Image.

3.1.4.6 omtengine.StaveDetection.java

public StaveDetection(BufferedImage buffImage)

public int[][] DetectStave(int startH, int endH, int startW, int endW,int blackp)

Detects the number of staves present in the image and stores the start and end coordinates of the stave lines.

public int getHeight()

Returns the height of the Buffered Image.

3.1.4.7 omtengine.DivideImage.java

public DivideImage(BufferedImage buffimg, int stavearray[][])

public BufferedImage[] dodivide(DrawRectangle drawrec)

Divided the given image based on the start and end coordinates of the image and also by considering the start coordinate of the first stave and the last coordinate of the last stave with respect to the height of the image.

int[][] returnstartend()

Returns the calculated start and end postion of each stave block

3.1.4.8 omtengine.DivideBlocks.java

public DivideBlocks(double position,BufferedImage buffimg,int thres,int coun int sp[],int stavearray[][],int startend[],int stavno,int sharp,int flat,int transpose

public void divideb(DrawRectangle drawblk,PitchDuration pd)

Divides the buffered image of the stave into blocks

public int [] returnbars()

Returns the X-Projection of the Buffered Image

public double returnposition()

Returns the position of a note with respect to start of the stave]

3.1.4.9 omtengine.SegmentNotes.java

public SegmentNotes(BufferedImage buffimg,int thresw,int thresb)

public int[][] divideseg()

Detects the note heads, both empty notes and filled note heads and stores their coordinates.]

3.1.4.10 omtengine.DivideNotes.java

public DivideNotes(double position,BufferedImage buffimg,BufferedImage org, int pos[][],int thres,int coun,int c,int stavestart,int staveend,int sp[],int tx1tx2[],int startend[],int stavno,int sharp,int flat,int transpose)

public void dividen(DrawRectangle drawblk,PitchDuration pd)

Divides the image into segment containing a single or double note or a chord and calculates the position of a note note with respect to the start of the stave

public double returnposition()

Returns an integer value showing the position of a note head with respect to the start of a stave

3.1.4.11 omtengine.NoteDetection.java

public Notedetection(double position,int coun,BufferedImage buffImages, BufferedImage org,int yp[],int thres[],int startstave,int xstart, int ystart,int stavno,int sharp,int flat,int transpose)

public void Recognizenotes(DrawRectangle drawblk,PitchDuration pd)

Assigns a pitch value to a note and also changes this pitch value based on the parameters specified in the GUI ,like transpose and scale (flats and sharps).

public double returnposition()

Returns an integer value showing the position of a note head with respect to the start coordinate of a stave]

3.1.4.12 omtengine.PitchDuration.java

This class handles the list which stores the pitch, duration and position of a note.

public PitchDuration() Constructor for the class.

public void add(int pit,double dur,double pos) This function inserts a value into the list. First pitch, then duration and finally the position.

public int sizeoflist() This returns the number of notes detected.

public double[] valu(int i) This function returns an array in which pitch,duration and position is inserted.

3.1.5 projection package

3.1.5.1 Projection.java

public Projection(int[] Projection, int length, String name)

Creates chart and displays it on GUI.

3.1.6 recogniseglyph Package

3.1.6.1 CalcSignature

public CalcSignature(String reference)

Constructor which reads the path to the directory whose signature has to be found.

public Color[][] weight()

Returns the calculated weight of each image.

private RenderedImage rescale(RenderedImage i)

Rescales or normalized the image. It uses the JAI liberary functions.

private Color[][] calcSignature(RenderedImage i)

The signature is calculated here for each image.

private Color averageAround(RenderedImage i, double px, double py)

Returns the normalised rgb value of a pixel.

3.1.7 midi Package

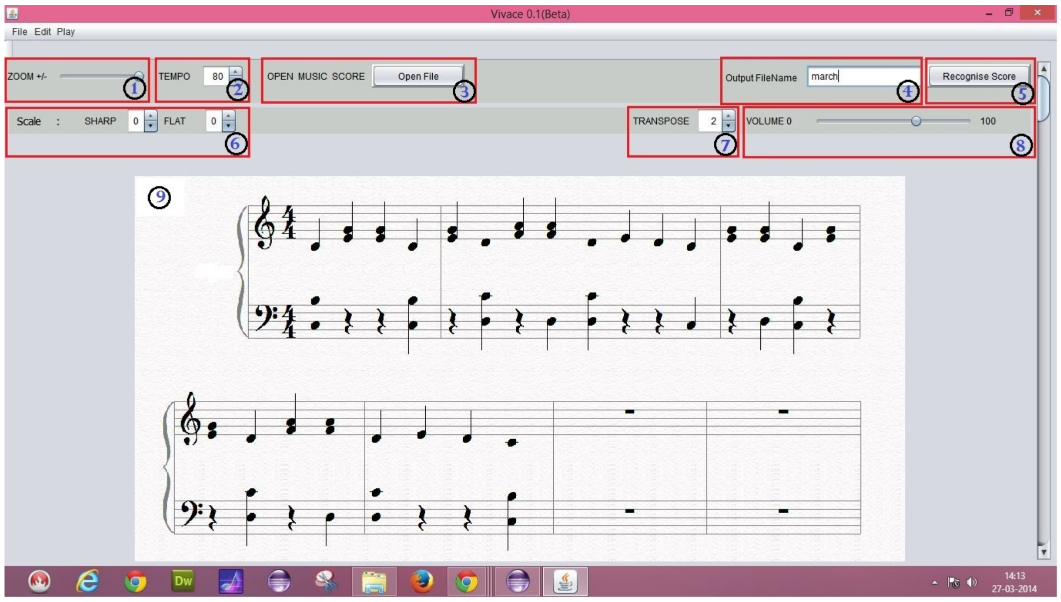
addNote(track, channel, pitch,time,duration,volume)

Add notes to the MIDIFile object Arguments:
track: The track to which the note is added.
channel: the MIDI channel to assign to the note. [Integer, 0-15]
pitch: the MIDI pitch number [Integer, 0-127].
time: the time (in beats) at which the note sounds [Float].
duration: the duration of the note (in beats) [Float].
volume: the volume (velocity) of the note. [Integer, 0-127].

3.2 GUI

The below shown is a figure of the GUI designed using the java Swing components for the Vivace Music Transcriptor.

1. Zoom :- zoom component is designed using a java swing slider component. It is used for rescaling the imaged loaded on to the frame.
2. Tempo :- The tempo, in Beats per Minute. Implemented using the swing jspinner component.The tempo can be changed as like the user wishes. Normally 80 is considered as a default tempo.
3. open music score :- Its a file open button. On clicking the button the file chooser will be opened and the needed file can be opened and the file will be displayed on to the frame



TEMPO 80

Output FileName march

Scale : SHARP 0 FLAT 0

TRANSPOSE 2

VOLUME 0 100

FIGURE 3.1: Vivace - An Open Music Transcriptor GUI

4. Output file name :- The final midi file generated can be saved with a new file name entered by the user in the textbox given.

5. Recognise score:- The recognise score runs the code for image processing and it ask for whether the processing is to be done for both hand notation or single hand notation. It also asks for conversion of image to black and white , in case of any noise or texture in the background of the input image.

6. Scale : The scale can be either sharp or flat . the sharp values and flat values can be specified by the user by changing the jspinner component.

7. Transpose : The music score recognised can be assigned with a transpose value with which it would be played. The transpose denotes the pitch of a music. It can be adjusted by changing the jspinner component.

8. Volume :- the volume can be increased or decreased by changing the jslider component.

9. Frame :- The Frame is the java component on to which the sheet music image is loaded.

The values of the java swing components are passed to the java program that would be handling the midi file generation from the sheet music given

CHAPTER 4

Testing

This chapter will discuss how the overall system was tested and the results will be presented in the next chapter. It is critical to thoroughly test an application in order to identify which components should be improved in future releases.

4.1 Testing Environment

The application was tested under the Windows 7, 8, 8.1 and Linux operating systems. It was therefore important to test any new GUI features on the three operating systems in order to preserve the cross-compatibility feature of the application.

4.2 False Positives and False Negatives

We want to decide if a person will fail as a police officer. So a false positive is if we incorrectly say that a person will fail. A false negative on the other hand is if we incorrectly predict that person wont fail. In terms of evaluating our system:

1. A false positive is for example if the application falsely identifies a note head which isnt one. 2. A false negative is for example when the application fails to detect a note present in the image.

4.3 Stave Detection

The simplest way to check that the staves were all correctly identified is to paint them over the recognised score. This module will therefore be visually assessed and if the staves as painted by the Vivace application do not appear to completely cover the staves in the original image, then we will reject it as being correctly identified. Stave lines detected can be identified from the y-projection graph and the pink rectangle drawn around the stave lines.

4.4 Note Heads Detected

The note heads are painted on top of the original notes as blue square boxes and this will enable us to visually assess how many note heads were correctly and incorrectly identified. We will count the total number of note heads in the original score, the number of false positives and the number of false negatives. This will then allow us to calculate the accuracy of the note head detection algorithm in terms of notes correctly recognized.

4.5 Pitch Calculation

For each note head found, the program outputs a letter A through G based on the calculated pitch of the note head. The pitch for each note head on the original score needs to be manually determined and this can then be compared with the pitch found by the program.

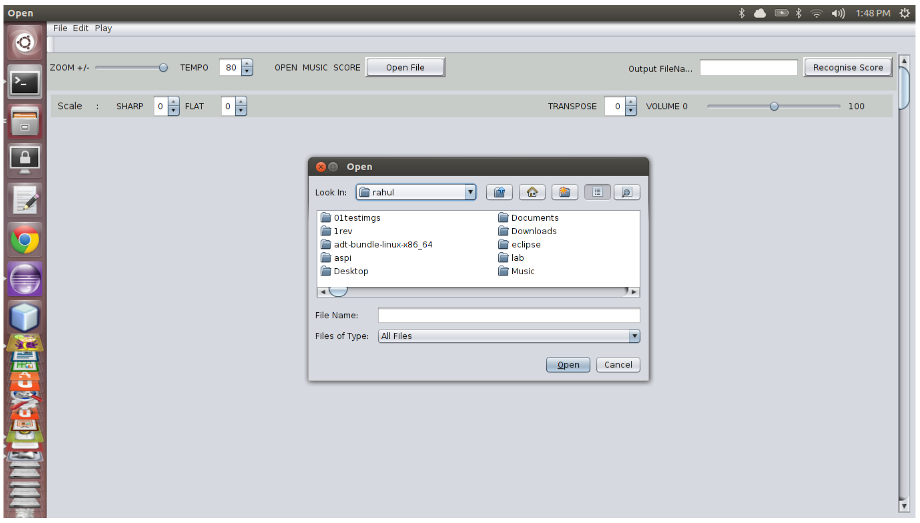
4.6 Note Duration

For each note head found, the program outputs the duration of the notes if it is a note else it says whether it is treble or bass clef. The following values are the output of the note identification.
treble, bass, 1/8, 1/16, 1/32, 1/4, 1/2, r/4, s1. In order to test and train program, several music glyphs were extracted from a selection of music scores.

4.7 Sample Test

4.7.1 Opening a File

A file is opened by clicking the open file button. Figure 4.1 shows the screenshot of this process.



TEMPO 80
SHARP 0
FLAT 0
TRANSPOSE 0
VOLUME 0 100

Look In: rahul

01testimgs
1rev
adt-bundle-linux-x86_64
aspi
Desktop
Documents
Downloads
eclipse
lab
Music

File Name:
Files of Type: All Files

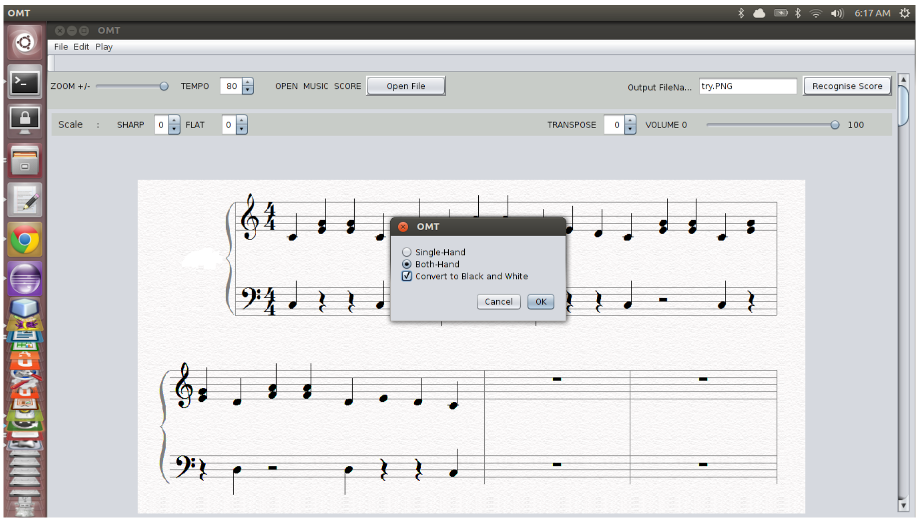
FIGURE 4.1: Opening an image

4.7.2 Recognition Options

The recognition is initiated by clicking the Ŕecognise Scoreb́utton. On clicking, a dialogue box appears that allows to select options. The options that can be selected are both hand or single hand and black and white.In this test case, we choose both hand and opted to convert image to black and white. Figure 4.2 shows the screenshot of this process.

4.7.3 Recognition Process

The recognition process begins as the OK button for the option dialogue box is clicked. The Y-Projection graph is displayed. It shows 4 five equidistant peaks which represent the stave lines. So the stave line numbers matches the input. Figure 4.3 shows the screenshot of this process.



OMT

( ) Single-Hand
(●) Both-Hand
[✓] Convert to Black and White

FIGURE 4.2: Choosing recognition options

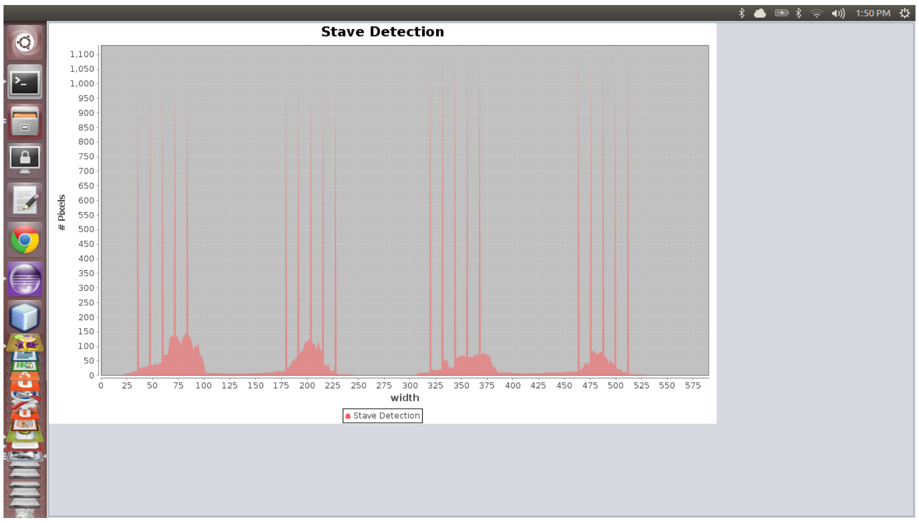


FIGURE 4.3: Y Projection used for stave line detection

As the black and white option was selected, the image converted to black and white is displayed as in figure 4.4.

Figure 4.5 shows the notes and stave lines that are identified. The pink rectangular boxes represent the detected stave lines. The blue boxes shows the recognized notes and symbols. The green box shows the objects that are identified but was not able to recognize. In this case, the whole rest note is not trained as now. So it is not recognized. All other notes are correctly identified.

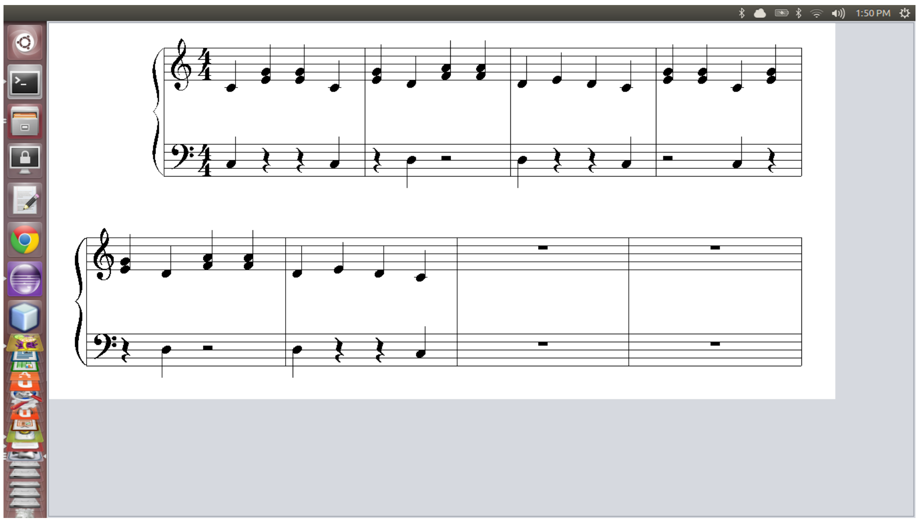


FIGURE 4.4: Image converted into black and white

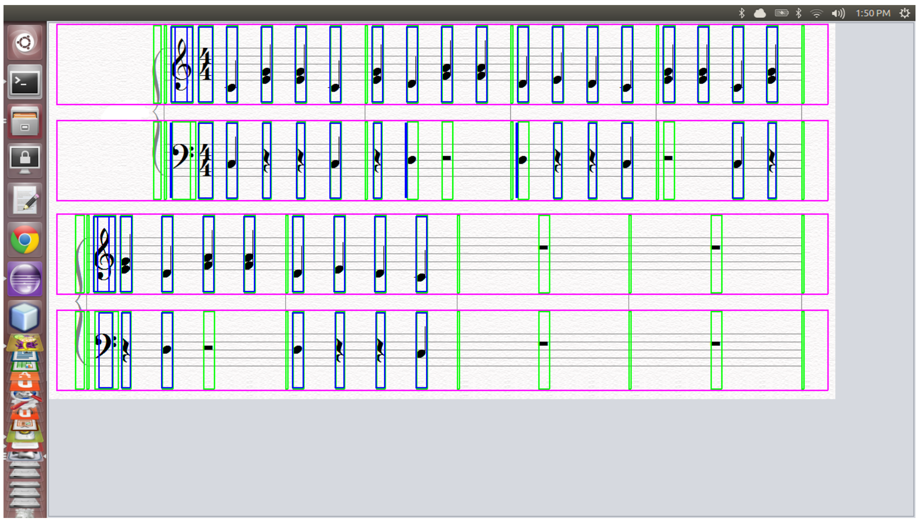


FIGURE 4.5: Recognized notes in the sheet

CHAPTER 5

Conclusion

Open Music Transcriptor (OMT) is a software for converting printed music scores into a format understandable by computers. Although several commercial applications exist, we have seen that they dont always perform accurately and those giving fairly good output is of very high cost.This was the motivation for this project. We investigated the research that has been conducted in the past in order to gain a better understanding in this field. A segmentation based approach was used for the design of this project and was based on the O^3MR and OpenOMR project.
The first step in a OMT application is to detect the thickness of stave lines and the spacing between the stave lines. We achieved this by using the RLE algorithm in order to produce a histogram for consecutive black and white pixel runs. The next phase involved detecting the stave lines and we saw that by taking the y-projection of the image, we were able locate the stave by looking for five equidistant peaks in the projection. The initial segmentation phase was performed to obtain horizontally segmented staves. We then proceeded to the next segmentation process of dividing the staves into blocks. In next step, we fill the empty note heads and then the note heads were detected by means of the RLE algorithm and the y-projection. Then a segmentation phase is involved to vertically segment the image to produce glyphs. The glyphs where the input to image comparison module. The pitch, duration and position is given to midi generation module.

CHAPTER 6

Future Scope

As one of the major goals of this project was to turn the OMT application developed into an open source project, it was important to identify aspects of the application that could be improved. Below is a set of suggested changes or enhancements that could be made to the current implementation .

1. **Improved Graphical User Interface** : Although much time was spent developing the current GUI, much improvement can be brought to it. The GUI was programmed in Swing and it may be worth investigating if it would be suitable to use SWT instead. Providing a zooming feature for the different images displayed is one example of a GUI enhancement.

2. **Interactive Features** : A nice feature would be to allow users to click on the recognised score in order to modify incorrectly identified notes. For example, if a note was omitted, the user could simply click on that note and the application would add it. Another interactive feature would be to illuminate the notes in a different colour as they are played.

3. **Accidental sharp and flats** : Include modules which helps to identify accidental sharps and flats in a music notation and also to identify the scale automatically.

4. **Time Signatures** : Make use of time signatures in a sheet music to get better outputs. Division into bars can help in better synchronization between right and left hand music, while generating music for both hand notations.

5. **Dotted Notes, dynamics and accents** : Identify dotted notes and alter the time duration of notes correspondingly also try to recognize the dynamics and accents given in the sheet music and alter the volume levels accordingly.

6. **Image Comparison** : The Image Comparison needs to be improved as this is currently affecting the overall performance of the application.

7. **Multi-Threading** : Currently the program is working on a single thread.If it is multi threaded then we can execute operations on staves concurrently and this will increase the speed of execution.

APPENDIX A

Source Code

A.1 Package : omtengine

A.1.1 DivideBlocks.java

```
/*STEP 2 __________ Divides the given stave into blocks of notes -> Filtering out
empty spaces (level -1 segmentation_____*/

package omtengine;
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import javax.imageio.ImageIO;
import drawimage.DisplayImage;
import drawimage.DrawRectangle;

        public class DivideBlocks {

                private BufferedImage buffimage;
                int xPro[];
                int thres;
                int startend[];
                int coun;
                int stavearray[][];
                int sp[];
                int stavno;
                double position;
                int sharp;
                int flat;
                int transpose;
                public DivideBlocks(double position,BufferedImage buffimg,int thres,
int coun,int sp[],int stavearray[][],int startend[],int stavno,int sharp,int flat,
                                                                int transpose)
                {
                        this.buffimage = buffimg;
                        this.thres=thres;
                        this.startend=startend;
                        this.stavearray=stavearray;
                        this.coun=coun;
                        this.stavno=stavno;
                        this.position=position;
                        this.sp=sp;
                        this.sharp=sharp;
                        this.flat=flat;
                        this.transpose=transpose;

                }



                public void divideb(DrawRectangle drawblk,PitchDuration pd)
                //division into bars //parameter to draw rectangles
                {
                        int count=0, height=buffimage.getHeight();
                        int width = buffimage.getWidth();
                        xPro= new int[width];
                        int count1=1;
                        XProjection xpro= new XProjection(buffimage);
                        xpro.calcXProjection(0, height, 0,width);
                        xPro=xpro.getXProjection();
                        int tx1=0,tx2=0;for(int i=0;i<width;i++)
                        {
                        if(xPro[i]>thres) // is something other than stave present here.
                        {
                        tx1=i-2;
                        for(;xPro[i]>thres && i<width;i++);
                                tx2=i+2;
                        if(tx2!=tx1 && tx2>tx1 ) count++;
                        }
                        }
                BufferedImage img[]= new BufferedImage[count];
                BufferedImage imgcopy[]= new BufferedImage[count];
```

```
 int tx1tx2[][]=new int[count][2];
 int flag=0;
 count=0;
 for(int i=0;i<width;i++)
 {
 if(xPro[i]>thres) // is something other than stave present here.
 {
 {
        tx1=i-2;
        for(;xPro[i]>thres && i<width;i++);
        tx2=i+2;
        if(tx2!=tx1 && tx2>tx1)
        {
        img[count] = new BufferedImage(tx2-tx1, buffimage.getHeight(),1);
        imgcopy[count] = new BufferedImage(tx2-tx1, buffimage.getHeight(),
                                                                       1);
        // draws the image chunk
        Graphics2D gr = img[count].createGraphics();
        tx1tx2[count][0]=tx1;
        tx1tx2[count][1]=tx2;
        Graphics2D gr1 = imgcopy[count++].createGraphics();
        gr.drawImage(buffimage, 0, 0, tx2-tx1, buffimage.getHeight(),
                        tx1,0, tx2, buffimage.getHeight(), null);
        gr1.drawImage(buffimage, 0, 0, tx2-tx1, buffimage.getHeight(),
                        tx1,0, tx2, buffimage.getHeight(), null);
        if(tx2<width-.01*width)// to prevent out of cordinate bounds
                                        //while drawing
        if(tx2>=width)
                drawblk.drawfig( tx1, startend[0]+2, tx2,startend[1]-2,
                                                        Color.GREEN);
        else
                drawblk.drawfig( tx1+2, startend[0]+2, tx2,startend[1]-2,
                                                        Color.GREEN);
        gr.dispose();
        gr1.dispose();
        }
 }
 }
 }

//writing mini images into image files
for (int i = 0; i < count; i++)  /// calling next step ie glyph division
{
try {
ImageIO.write(img[i], "png", new File("blocktest" + (coun)+(i+1) + ".png"));
SegmentNotes seg=new SegmentNotes(img[i],sp[1],sp[0]);
int posit[][]= seg.divideseg();
 DivideNotes divno=new DivideNotes(position,img[i],imgcopy[i],posit,thres,
                coun,i,stavearray[coun][0],stavearray[coun][1],sp,tx1tx2[i],startend,
                stavno,sharp,flat,transpose);
                divno.dividen(drawblk,pd);
               position=divno.returnposition();
                } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
                }
                }
        }
public double returnposition()
{
return position;
}
public int[] returnbars()
{
return xPro;
}
}
```

A.1.2 DivideImage.java

```
/*STEP 1_____ Divides a given notation in different staves_____*/
package omtengine;
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import javax.imageio.ImageIO;

import drawimage.*;

public class DivideImage {

        private BufferedImage buffimage;
        int a,b,c,d,mid;
        int stavearray[][];
        int startend[][];

        public DivideImage(BufferedImage buffimg, int stavearray[][])
        {
                this.stavearray= new int[stavearray.length][2];
        this.buffimage = buffimg;
        this.stavearray = stavearray;
}

public BufferedImage[] dodivide(DrawRectangle drawrec) //number of staves
                                                  //parameter to draw image
{
        BufferedImage img[]= new BufferedImage[stavearray.length];
        int startend[][]=new int[stavearray.length][2];
                                // start end cordinated of calculated blocks
        this.startend=startend;
        int count=0, height;
        float midt1=0,midt2=0;
        int width = buffimage.getWidth();
        for(int i=0;i<stavearray.length;i++)
        {
        {
        height = stavearray[i][1]-stavearray[i][0];
        //Initialize the image array with image chunks
        if(i==0 && i==stavearray.length-1)
        {
        if(stavearray[i][0]<0.75*height)
                midt1=0;
        else
                midt1=(float) (0.75*height);
        if(stavearray[i][1]+0.75*height>buffimage.getHeight())
                midt2=buffimage.getHeight()-stavearray[i][1];
        else
                midt2=(float)0.75*height;
        }
        else if(i==0)
        {
                if(stavearray[i][0]<0.75*height)
                        midt1=0;
                else
                        midt1=(float)0.75*height;
                        midt2=(float)0.75*height;
        }
        else if(i==stavearray.length-1)
        {
                if(stavearray[i][1]+0.75*height>buffimage.getHeight())
                        midt2=buffimage.getHeight()-stavearray[i][1];
                else
                        midt2=(float)0.75*height;
                midt1=(float)0.75*height;
        }
        else
        {
                midt1=(float) (0.75*height);
                midt2=(float) (0.75*height);
                }
img[count] = new BufferedImage(buffimage.getWidth(), height+(int)midt1+(int)midt2,1);
             startend[count][0]=stavearray[i][0]-(int)midt1;
             startend[count][1]=stavearray[i][1]+(int)midt2;
             // draws the image chunk
             Graphics2D gr = img[count++].createGraphics();
             gr.drawImage(buffimage, 0, 0, width, height+(int)midt1+(int)midt2,
        0,stavearray[i][0]-(int)midt1, width, stavearray[i][1]+(int)midt2, null);
             drawrec.drawfig((int)(width*0.01),stavearray[i][0]-(int)midt1,
        (int)(width-width*0.01), stavearray[i][1]+(int)midt1, Color.MAGENTA);
              gr.dispose();
        }
        }
        //writing mini images into image files
        for (int i = 0; i < img.length; i++) {
        try {
        ImageIO.write(img[i], "png", new File("imgtest" + i + ".png"));
        } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        }
        }
                return img;

        }
        int[][] returnstartend()
        {
                return startend;
        }

}
```

A.1.3 DivideNotes.java

```
/*_________ Divides the given stave into blocks of notes -> Filtering out
empty spaces (level -1 segmentation_____*/

package omtengine;

        import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import javax.imageio.ImageIO;

import drawimage.DisplayImage;
import drawimage.DrawRectangle;

        public class DivideNotes {

                private BufferedImage buffimage;

                int xPro[];
                int thres;
                int coun;
                int c;
                int posi[][];
                int sp[];
                int startend[];
                int stavestart;
                int staveend;
                int startstv;
                int endstv;
                int stavno;
                int tx1tx2[];
                int sharp;
                int flat;
                double position;
                int transpose;
                private BufferedImage org;

                public DivideNotes(double position,BufferedImage buffimg,
BufferedImage org,int pos[][],int thres,int coun,int c,int stavestart,int staveend,
int sp[],int tx1tx2[],int startend[],int stavno,int sharp,int flat,int transpose)
                {
                        this.buffimage = buffimg;
                        this.posi=pos;
                        this.position=position;
                        this.startend=startend;
                        this.stavno=stavno;
                        this.sp=sp;
                        this.org=org;
                        //this.stavestart=0;
                        this.startstv=stavestart;
                        //this.staveend=0;
                        this.tx1tx2=tx1tx2;
                        this.endstv=staveend;
                        this.thres=thres;
                        this.c=c;
                        this.coun=coun;
                        this.sharp=sharp;
                        this.flat=flat;
                        this.transpose=transpose;

                }



                public void dividen(DrawRectangle drawblk,PitchDuration pd)
                {
                int count=0, height=buffimage.getHeight();
                int tx1=0,tx2=1;
                int width=posi.length;
                System.out.println("width"+width);
                BufferedImage img[]= new BufferedImage[width];
                BufferedImage imgorg[]= new BufferedImage[width];
                count=0;
                for(int i=0;i<width;i++)
                {
                tx1=posi[i][0];
                tx2=posi[i][1]+1;
                img[count] = new BufferedImage(tx2-tx1, buffimage.getHeight(),1);
                imgorg[count] = new BufferedImage(tx2-tx1, org.getHeight(),1);
                drawblk.drawfig(tx1tx2[0]+tx1+1, startend[0]+4,tx1tx2[0]+tx2-2,
                                                startend[1]-4, Color.BLUE);
                // draws the image chunk
                Graphics2D gr = img[count].createGraphics();
                Graphics2D gr1 = imgorg[count++].createGraphics();
                gr.drawImage(buffimage, 0, 0, tx2-tx1, buffimage.getHeight(),
                                        tx1,0, tx2, buffimage.getHeight(), null);
                gr1.drawImage(org, 0, 0, tx2-tx1, org.getHeight(), tx1,0,
                                        tx2, org.getHeight(), null);
                gr.dispose();
                }
                stavestart=buffimage.getHeight()*3/10;
               //writing mini images into image files
                for (int i = 0; i < width; i++)
                {
                try {
                ImageIO.write(imgorg[i],"png",new File("glyph"+coun+c+(i+1)+".png"));
                ImageIO.write(img[i],"png",new File("glyph00"+coun+c+(i+1)+".png"));
                YProjection ypro=new YProjection(img[i]);
                ypro.calcYProjection(0, img[i].getHeight(), 0, img[i].getWidth());
                int Ypro[]=ypro.getYProjection();
                Notedetection notedetec=new Notedetection(position,coun,img[i],
        imgorg[i],Ypro,sp,stavestart,tx1tx2[0]+posi[i][0]+1, startend[0]+4,stavno,sharp,
        flat,transpose);
```

```
                notedetec.Recognizenotes(drawblk,pd);
                position=notedetec.returnposition();
                } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
                }
                }
                }
                public double returnposition()
                {
                        return position;
                }
}
```

A.1.4 Notedetection.java

```
package omtengine;

import java.awt.image.BufferedImage;
//values to be received:start pixel of the 1st stave line & distance b/w stave lines.
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import javax.imageio.ImageIO;

import recogniseglyph.CompareImage;

import drawimage.DrawRectangle;

public class Notedetection
{

        BufferedImage buffImag;
        int threshold;
        int YProjection[];
        int thresblack;
        int staren[][]=new int[1][2];
        int startstave;
        int xstart;
        double position;
        BufferedImage org;
        int ystart;
        int stavno;
        int sharp=0;                            // values from Gui
        int flat=0;
        int transpose;
        int coun;
        int countt;

public Notedetection(double position,int coun,BufferedImage buffImages,
BufferedImage org,int yp[],int thres[],int startstave,int xstart, int ystart,
int stavno,int sharp,int flat,int transpose)
{
        this.coun=coun;
        this.buffImag=buffImages;
        this.YProjection=yp;
        this.startstave=startstave;
        this.threshold=thres[1];
        this.org=org;
        this.thresblack=thres[0];
        this.xstart=xstart;
        this.stavno=stavno; //staveno+1
        this.ystart=ystart;
        this.sharp=sharp;
        this.flat=flat;
        this.position=position;
        this.transpose=transpose;

}
public void Recognizenotes(DrawRectangle drawblk,PitchDuration pd)
{
        int i,count=0;
        int start[] =new int[20],end[]=new int[20];
        int height;
        int midilist[]=new int[10];
        int staveflag=1;
        int no=0;
        int temp,h;
        int pos[] =new int[20];
        int midi = 0;

        for(i=0;i<buffImag.getHeight();i++)
        {
                if(YProjection[i]>threshold-2*thresblack)
                {
                        if(count==0)
                        {
                                start[no]=i;
                        }
                        count++;
                }
                else
                {
                        if(count>=threshold-2*thresblack)
                        {
                                height=i-start[no];
                                temp=height/(threshold-thresblack);
                                end[no]=start[no]+threshold;
                                pos[no]=(start[no]+end[no])/2;
                                if(temp>=2)
                                {
                                for(h=1;h<temp;h++)
                                {
                                start[++no]=end[no-1]+1;
                                end[no]=start[no]+threshold;
                                pos[no]=(start[no]+end[no])/2;
                                }
                                }
                                no++;
                        }
                        count=0;
                }
        }
        System.out.println((no)+" Notes Detected ");
        int beg=0,nocop=no;
        beg=startstave;
        int dist=thresblack+threshold;
        int s1=beg+thresblack/2;
        int s2=s1+dist;
        int s3=s2+dist;
        int s4=s3+dist;
        int s5=s4+dist;
        int s6=s5+dist;
        int s7=s6+dist;
        int s8=s7+dist;
        int s20=s1-dist;
        int s19=s20-dist;
        int s18=s19-dist;
        int s17=s18-dist;


        int d0=(s20+s1)/2;//-((thresblack/2)+(threshold/2));
        int d1=(s1+s2)/2;
        int d2=(s2+s3)/2;
        int d3=(s3+s4)/2;
        int d4=(s4+s5)/2;
        int d5=(s5+s6)/2;
        int d6=d5+(s6+s7)/2;
        int d7=d6+(s7+s8)/2;
        int d8=d7+((thresblack/2)+(threshold/2));
        int d19=(s19+s20)/2;
        int d18=(s18+s19)/2;
        int d17=(s17+s18)/2;
        int lim=threshold/3;

        for(i=0;i<nocop;i++)
        {
        no=i;
        if(staveflag==1)
        {
        if(pos[no]>d18-lim && pos[no]<d18+lim)
        {
                midi=74;
                System.out.println("D5");
        }
        else if(pos[no]>s19-lim && pos[no]<s19+lim)
        {
                midi=72;
                System.out.println("C5");
        }

        else if(pos[no]>d19-lim && pos[no]<d19+lim)
        {
                midi=71;
                System.out.println("B4");
        }
        else if(pos[no]>s20-lim && pos[no]<s20+lim)
        {
                midi=69;
                System.out.println("A4");
        }
        else if(pos[no]>d0-lim && pos[no]<d0+lim)
        {
                midi=67;
                System.out.println("G4");
        }
        else if(pos[no]>s1-lim && pos[no]<s1+lim)
        {
                midi=65;
                System.out.println("F4");
        }
        else if(pos[no]>d1-lim && pos[no]<d1+lim)
        {
                midi=64;
                System.out.println("E4");
        }

        else if(pos[no]>s2-lim && pos[no]<s2+lim)
```

```
{
        midi=62;
        System.out.println("D4");
}
else if(pos[no]>d2-lim && pos[no]<d2+lim)
{
        midi=60;
        System.out.println("C4");
}

else if(pos[no]>s3-lim && pos[no]<s3+lim)
{
        midi=59;
        System.out.println("B3");
}
else if(pos[no]>d3-lim && pos[no]<d3+lim)
{
        midi=57;
        System.out.println("A3");
}
else if(pos[no]>=s4-lim && pos[no]<=s4+lim)
{
        midi=55;
        System.out.println("G3");
}
else if(pos[no]>d4-lim && pos[no]<d4+lim)
{
        midi=53;
        System.out.println("F3");
}
else if(pos[no]>s5-lim && pos[no]<s5+lim)
{
        midi=52;
        System.out.println("E3");
}
else if(pos[no]>d5-lim && pos[no]<d5+lim)
{
        midi=50;
        System.out.println("D3");
}
else if(pos[no]>s6-lim && pos[no]<s6+lim)
{
        midi=48;
        System.out.println("C3");
}
else if(pos[no]>=d6-lim && pos[no]<d6+lim)
{
        midi=47;
```

```
                System.out.println("B2");
        }
        else if(pos[no]>s7-lim && pos[no]<s7+lim)
        {
                midi=45;
                System.out.println("A2");
        }
        else if(pos[no]>d7-lim && pos[no]<d7+lim)
        {
                midi=43;
                System.out.println("G2");
        }
        else if(pos[no]>s8-lim && pos[no]<s8+lim)
        {
                midi=41;
                System.out.println("F2");
        }
        else if(pos[no]>d8-lim && pos[no]<d8+lim)
        {
                midi=40;
                System.out.println("E2");
        }
        else
                System.out.println(" Code yet to be written ");

        }
        else
        {



                if(pos[no]>s17-lim && pos[no]<s17+lim)
                {
                        midi=59;
                        System.out.println("B3");
                }
                else if(pos[no]>d17-lim && pos[no]<d17+lim)
                {
                        midi=57;
                        System.out.println("A3");
                }
                else if(pos[no]>s18-lim && pos[no]<s18+lim)
                {
                        midi=55;
                        System.out.println("G3");
                }
                else if(pos[no]>d18-lim && pos[no]<d18+lim)
                {
                        midi=53;
                        System.out.println("F3");
                }
                else if(pos[no]>s19-lim && pos[no]<s19+lim)
                {
                        midi=52;
                        System.out.println("E3");
                }

                else if(pos[no]>d19-lim && pos[no]<d19+lim)
                {
                        midi=50;
                        System.out.println("D3");
                }
                else if(pos[no]>s20-lim && pos[no]<s20+lim)
                {
                        midi=48;
                        System.out.println("C3");
                }
                else if(pos[no]>d0-lim && pos[no]<d0+lim)
                {
                        midi=47;
                        System.out.println("B2");
                }
                else if(pos[no]>s1-lim && pos[no]<s1+lim)
                {
                        midi=45;
                        System.out.println("A2");
                }
                else if(pos[no]>d1-lim && pos[no]<d1+lim)
                {
                        midi=43;
                        System.out.println("G2");
                }

                else if(pos[no]>s2-lim && pos[no]<s2+lim)
                {
                        midi=41;
                        System.out.println("F2");
                }
                else if(pos[no]>d2-lim && pos[no]<d2+lim)
                {
                        midi=40;
                        System.out.println("E2");
                }

                else if(pos[no]>s3-lim && pos[no]<s3+lim)
                {
                        midi=38;
                        System.out.println("D2");
                }
                else if(pos[no]>d3-lim && pos[no]<d3+lim)
                {
                        midi=36;
                        System.out.println("C2");
                }
                else if(pos[no]>=s4-lim && pos[no]<=s4+lim)
                {
                        midi=35;
                        System.out.println("B1");
                }
                else if(pos[no]>d4-lim && pos[no]<d4+lim)
                {
                        midi=33;
                        System.out.println("A1");
                }
                else if(pos[no]>s5-lim && pos[no]<s5+lim)
                {
                        midi=31;
                        System.out.println("G1");
                }
                else if(pos[no]>d5-lim && pos[no]<d5+lim)
                {
                        midi=29;
                        System.out.println("F1");
                }
                else if(pos[no]>s6-lim && pos[no]<s6+lim)
                {
                        midi=28;
                        System.out.println("E1");
                }
                else if(pos[no]>=d6-lim && pos[no]<d6+lim)
                {
                        midi=26;
                        System.out.println("D1");
                }
                else if(pos[no]>s7-lim && pos[no]<s7+lim)
                {
                        midi=24;
                        System.out.println("C1");
                }
                else if(pos[no]>d7-lim && pos[no]<d7+lim)
                {
                        midi=23;
                        System.out.println("B0");
                }
                else if(pos[no]>s8-lim && pos[no]<s8+lim)
```

```
                {
                        midi=21;
                        System.out.println("A0");
                }
                else if(pos[no]>d8-lim && pos[no]<d8+lim)
                {
                        midi=19;
                        System.out.println("G0");
                }
                else
                        System.out.println(" Code yet to be written ");







        }
        midilist[i]=midi;  //pass this
        }
        countt=i;
        for(i=0;i<nocop;i++)
                System.out.println(midilist[i]);
        CompareImage cmpimg;

        if(midi!=0)
        {
                if(sharp!=0)
                {
                if(sharp==1)
                {
                        if(midi==29 ||midi==41||midi==53||midi==65||midi==77)
                                midi=midi+1;
                }
                else if(sharp==2)
                {
                        if(midi==29 ||midi==41||midi==53||midi==65||midi==77||
                                midi==24||midi==36||midi==48||midi==60||midi==72)
                                midi=midi+1;
                }
                else if(sharp==3)
                {
                        if(midi==29 ||midi==41||midi==53||midi==65||midi==77||
                                midi==24||midi==36||midi==48||midi==60||midi==72||
                                midi==19||midi==31||midi==43||midi==55||midi==67||
                                midi==79)
                                midi=midi+1;
                }
                else if(sharp==4)
```

```
{
        if(midi==29 ||midi==41||midi==53||midi==65||midi==77||
                midi==24||midi==36||midi==48||midi==60||midi==72||
                midi==19||midi==31||midi==43||midi==55||midi==67||
                midi==79||midi==26||midi==38||midi==50||midi==62||
                midi==74)
                midi=midi+1;
}
else if(sharp==5)
{
        if(midi==29 ||midi==41||midi==53||midi==65||midi==77||
                midi==24||midi==36||midi==48||midi==60||midi==72||
                midi==19||midi==31||midi==43||midi==55||midi==67||
                midi==79||midi==26||midi==38||midi==50||midi==62||
                midi==74||midi==21||midi==33||midi==45||midi==57||
                midi==69)
                midi=midi+1;
}
else if(sharp==6)
{
        if(midi==29 ||midi==41||midi==53||midi==65||midi==77||
                midi==24||midi==36||midi==48||midi==60||midi==72||
                midi==19||midi==31||midi==43||midi==55||midi==67||
                midi==79||midi==26||midi==38||midi==50||midi==62||
                midi==74||midi==21||midi==33||midi==45||midi==57||
                midi==69||midi==28||midi==40||midi==52||midi==64||
                midi==76)
                midi=midi+1;
}
else if(sharp==7)
{
        if(midi==29 ||midi==41||midi==53||midi==65||midi==77||
                midi==24||midi==36||midi==48||midi==60||midi==72||
                midi==19||midi==31||midi==43||midi==55||midi==67||
                midi==79||midi==26||midi==38||midi==50||midi==62||
                midi==74||midi==21||midi==33||midi==45||midi==57||
                midi==69||midi==28||midi==40||midi==52||midi==64||
                midi==76||midi==23||midi==35||midi==47||midi==59||
                midi==71||midi==29||midi==41||midi==53||midi==65)
                midi=midi+1;
}
}
else
{
        if(flat!=0)
        {
                if(flat==1)
                {
```

```
if(midi==23||midi==35||midi==47||midi==59||midi==71)
        midi=midi-1;
}
if(flat==2)
{
if(midi==23||midi==35||midi==47||midi==59||midi==71||
midi==28||midi==40||midi==52||midi==64||midi==76)
        midi=midi-1;
}
if(flat==3)
{
if(midi==23||midi==35||midi==47||midi==59||midi==71||
midi==28||midi==40||midi==52||midi==64||midi==76||
midi==21||midi==33||midi==45||midi==57||midi==69)
                midi=midi-1;
}
if(flat==4)
{
if(midi==23||midi==35||midi==47||midi==59||midi==71||
midi==28||midi==40||midi==52||midi==64||midi==76||
midi==21||midi==33||midi==45||midi==57||midi==69||
midi==26||midi==38||midi==50||midi==62||midi==74)
                midi=midi-1;
}
if(flat==5)
{
if(midi==23||midi==35||midi==47||midi==59||midi==71||
midi==28||midi==40||midi==52||midi==64||midi==76||
midi==21||midi==33||midi==45||midi==57||midi==69||
midi==26||midi==38||midi==50||midi==62||midi==74||
midi==19||midi==31||midi==43||midi==55||midi==67||
midi==79)
                midi=midi-1;
}
if(flat==6)
{
if(midi==23||midi==35||midi==47||midi==59||midi==71||
midi==28||midi==40||midi==52||midi==64||midi==76||
midi==21||midi==33||midi==45||midi==57||midi==69||
midi==26||midi==38||midi==50||midi==62||midi==74||
midi==19||midi==31||midi==43||midi==55||midi==67||
midi==79||midi==24||midi==36||midi==48||midi==60||
midi==72)
                midi=midi-1;
}
if(flat==7)
{
if(midi==23||midi==35||midi==47||midi==59||midi==71||
                                        midi==28||midi==40||midi==52||midi==64||midi==76||
                                        midi==21||midi==33||midi==45||midi==57||midi==69||
                                        midi==26||midi==38||midi==50||midi==62||midi==74||
                                        midi==19||midi==31||midi==43||midi==55||midi==67||
                                        midi==79||midi==24||midi==36||midi==48||midi==60||
                                        midi==72||midi==29||midi==41||midi==53||midi==65||
                                        midi==77)
                                                midi=midi-1;
                                        }
                                }
                        }
                        for(i=0;i<countt;i++)
                                midilist[i]=midilist[i]+transpose;
                        midi+=transpose;

                        cmpimg=new CompareImage(position,org,midilist,countt,pd);
                        position=cmpimg.returnposition();
                        System.out.print("note detection : "+position);

        }

}
public double returnposition()
{
        return position;
}
}
```

A.1.5 OpenMusicTranscriptor.java

```
package omtengine;

import java.awt.image.BufferedImage;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;

import javapy.CreateMid;

import javax.imageio.ImageIO;
import drawimage.DrawRectangle;
import drawimage.DisplayImage;

public class OpenMusicTranscriptor {



        public OpenMusicTranscriptor(String filename,String vol,String shp,
                String flt,String trp,String tmp,String ofl,int bw,int typeflag)
        {
        int starH,enH,starW,enW;
        int sp[]=new int[2];
        double startposition=0.0,endposition=0.0;
        int volume=(int)Double.parseDouble(vol) ;//Integer.parseInt(vol);
        int sharp=(int)Double.parseDouble(shp);
        int flat=(int)Double.parseDouble(flt);
        int transpose=(int)Double.parseDouble(trp);
        int tempo=(int)Double.parseDouble(tmp);
        String opfile=ofl;
        double position=0.0;
        BufferedImage buffImage;
        try{
        File file = new File(filename);
        FileInputStream fis= new FileInputStream(file);
        buffImage=ImageIO.read(fis);
        System.out.println(bw);
        DoBlackandWhite dobw=new DoBlackandWhite(buffImage);
        if(bw==1)
        {
                System.out.println("BW");
                buffImage=dobw.doBW();
        }
        new DisplayImage(buffImage);
        StaveParameters sparam= new StaveParameters(buffImage);
        starH=1;
        starW=1;
        enH=buffImage.getHeight();
        enW=buffImage.getWidth();
        sparam.calcStaveParameters(starH, enH, starW, enW);
    sp=sparam.findStaveParameters();
    StaveDetection sdetec= new StaveDetection(buffImage);
    sdetec.tb=sp[0];
    sdetec.tw=sp[1];
    int arr[][]=new int[sdetec.stavelineno][2];
    arr=sdetec.DetectStave(starH, enH, starW, enW,sp[0]);
    PitchDuration pd=new PitchDuration();
    DrawRectangle drawblk=new DrawRectangle(filename); //Declare an image to draw
                                                       // rectangles
    BufferedImage img[]= new BufferedImage[arr.length];
```

```
DivideImage imgs= new DivideImage(buffImage, arr);  //arr-> Stave array
img=imgs.dodivide(drawblk);                        //parameter to draw rectangles
int startend[][]=new int[arr.length][2];
startend=imgs.returnstartend();
int thres=5*sdetec.tb+2;  // 2 set to divide slurs
 System.out.println("thereshld "+thres);
int k=0;
 int tempflag=0;

 for(k=0;k<arr.length;k++)
 {
     if(tempflag==0)
     {
    startposition=position;
    DivideBlocks divblo=new DivideBlocks(position,img[k],thres,k,sp,arr,
                                startend[k],k+1,sharp,flat,transpose);
    divblo.divideb(drawblk,pd);                    //parameter to draw rectangles
    position=divblo.returnposition();
    endposition=position;
    if(typeflag==1)
    {
            position=startposition;
            tempflag=1;
    }
    }
    else
    {
             DivideBlocks divblo=new DivideBlocks(position,img[k],thres,k,sp,arr,
                                       startend[k],k+1,sharp,flat,transpose);

             divblo.divideb(drawblk,pd);          //parameter to draw rectangles
             position=endposition;
             tempflag=0;

     }

 }
 // the notes identified and its duration
 int noofnotes=pd.sizeoflist();
 double note[][]=new double[noofnotes][3];

 for(int i=0;i<noofnotes;i++)
 {
     note[i]=pd.valu(i);
 }

 new CreateMid(noofnotes,note,opfile,volume,tempo);
 drawblk.writeimg();
```

```
}
                catch(IOException e){
                        e.printStackTrace();
        }
        }
}
```

A.1.6 PitchDuration.java

```
package omtengine;

import java.util.ArrayList;

public class PitchDuration {

        ArrayList<Integer> pitch;
        ArrayList<Double> duration;
        ArrayList<Double> position;
        public PitchDuration(){
                pitch=          new ArrayList<Integer>();
                duration=       new ArrayList<Double>();
                position= new ArrayList<Double>();
        }

        public void add(int pit,double dur,double pos)
        {
                pitch.add(pit);
                duration.add(dur);
                position.add(pos);
        }

        public int sizeoflist()         // returns the number of notes identified.
        {
                return pitch.size();
        }

        public double[] valu(int i)     // returns in an array
        {
                double temp[][]=new double[1][3];
                temp[0][0]=pitch.get(i);
                temp[0][1]=duration.get(i);
                temp[0][2]=position.get(i);
                return temp[0];
        }
}
```

A.1.7 StaveDetection.java

```
package omtengine; /*____ Detects Staves______*/

import java.awt.Color;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import javax.imageio.ImageIO;

import projection.Projection;
import drawimage.DrawRectangle;

/*
 *
 * @author Prince Mathew
 * @version 1.0
 */

public class StaveDetection
{
        private BufferedImage buffImage;
        private int stavearray[][];
        private int height;
        int stavelineno=0;
        private int width;
        int blackp;
        int tb,tw;

        public StaveDetection(BufferedImage buffImage)
        {
                this.buffImage = buffImage;
                height = buffImage.getHeight();
                width=buffImage.getWidth();
        }


        /**
         * Cacluate the Y-Projection of the BufferedImage
         * @param startH Desired start Y-Coordinate of the BufferedImage
         * @param endH Desired end Y-Coordinate of the BufferedImage
         * @param startW Desired start X-Coordinate of the BufferedImage
         * @param endW Desired end X-Coordinate of the BufferedImage
         */

public int[][] DetectStave(int startH, int endH, int startW, int endW,int blackp)
        {

                int height = endH - startH + 1;
                this.blackp=blackp;
                this.height = height;
                int width= endW-startW+1;
                this.width= width;
                int stavethres;
                int no,carray=0;

                int yProjection[];
                int tempstavearrayno[][]=new int[20][2];
                yProjection = new int[height];

                for (int i = startH; i < endH; i += 1)
                {
                        for (int j = startW; j < endW; j += 1)
                        {
                                int color;
                                try
                                {
                                        color = buffImage.getRGB(j, i);
                                        if (color == -1) // white pixel
                                        {
                                                // Do nothing
                                        } else
                                        {
                                                yProjection[i - startH] += 1;
                                        }
                                }
                                catch (ArrayIndexOutOfBoundsException e)
                                {

                                }

                        }
                }

                new Projection(yProjection,yProjection.length,"Stave Detection");

                no=0;
                int i;
                int maxy=0;
```

```
for(i=startH;i<endH;i+=1)
{
        if(yProjection[i]>maxy)
                maxy=yProjection[i];
}
if(blackp<2)
        stavethres= 88*maxy/100; //percentage -- dynamically assigned
else if(blackp<6)
        stavethres= 77*maxy/100;//percentage -- dynamically assigned
else if(blackp<10)
        stavethres= 66*maxy/100;
else
        stavethres= 55*maxy/100;
int flag=0;

for(i=startH;i<endH;i+=1)
{
        int j;
        //lab1:
        if(yProjection[i]>stavethres)
        {

                no+=1;

                if(no==5)
                {

                        stavelineno+=1;

                        tempstavearrayno[carray][0]=i-4*(tb+tw)+1;
                        tempstavearrayno[carray][1]=i+tb;
                        no=0;
                        carray+=1;
                }
                i=i+tb+tw-2;
        }
}
System.out.println("Number of Staves Recognized");
System.out.println(stavelineno);
stavearray=new int[stavelineno][2];
for(i=0;i<stavelineno;i++)
{
        stavearray[i][0]=tempstavearrayno[i][0];
        stavearray[i][1]=tempstavearrayno[i][1];
        System.out.println(stavearray[i][0]);
        System.out.println(stavearray[i][1]);
        System.out.println("\n");
}
```

```
                return stavearray;
        }

        public int getHeight()
        {
                return height;
        }


}
```

A.2 Package : recogniseglyph

A.2.1 CompareImage.java

```
package recogniseglyph;

import java.awt.Color;
import java.awt.image.BufferedImage;
import java.awt.image.RenderedImage;
import java.awt.image.renderable.ParameterBlock;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import javax.imageio.ImageIO;
import javax.media.jai.InterpolationNearest;
import javax.media.jai.JAI;
import javax.media.jai.iterator.RandomIter;
import javax.media.jai.iterator.RandomIterFactory;

import omtengine.PitchDuration;
import drawimage.DisplayImage;

import java.util.ArrayList;




class FindSymbol {


    // The reference image "signature" (25 representative pixels, each in R,G,B).
        // We use instances of Color to make things simpler.
```

```
private Color[][] signature;

// The base size of the images.
private static final int baseSize = 300;
private int count=0;
int imgno;



public FindSymbol(int[] reference, BufferedImage srcimage, String note,
                        int imgno) throws IOException
{
        RenderedImage src=(RenderedImage)srcimage;

        // Put the reference, scaled,
        RenderedImage ref = rescale(src);

// Calculate the signature vector for the reference.
        signature = calcSignature(ref);
        int[][] otherimg=new int[5][5];
        this.imgno=imgno;

    // For each image, calculate its signature and its distance from the
    // reference signature.
    double[] distances = new double[imgno];
    int k=0;
    for (int o = 0; o < imgno; o++)
    {

        for(int x=0;x<5;x++)
                for(int y=0;y<5;y++)
                {
                        otherimg[x][y]=reference[k];
                        k++;
                }
        distances[o] = calcDistance(otherimg);

    }



    for (int o = 0; o < imgno; o++)
    {

        if(distances[o]==0)
                count=count+1000+1000*((1/imgno)*10);  //Got exact same
                                                      //image in training set
        if(distances[o]<100)
                count+=100+100*((1/imgno)*10);
        else if(distances[o]<500)
```

```
                        count+=25+25*((1/imgno)*5);
                else if(distances[o]<1000)
                        count+=10+10*((1/imgno));
                else if (distances[o]<2000)
                        count+=3+(3*((1/imgno)));

            }

    }

    public int weight()
    {
        return count;
    }



        private RenderedImage rescale(RenderedImage i)
        {
             float scaleW = ((float) baseSize) / i.getWidth();
             float scaleH = ((float) baseSize) / i.getHeight();
                 // Scales the original image
             ParameterBlock pb = new ParameterBlock();
             pb.addSource(i);
             pb.add(scaleW);
             pb.add(scaleH);
             pb.add(0.0F);
             pb.add(0.0F);
             pb.add(new InterpolationNearest());
             // Creates a new, scaled image and uses it on the DisplayJAI component
             return JAI.create("scale", pb);
        }

        /*
         * This method calculates and returns signature vectors for the input image.
         */
        private Color[][] calcSignature(RenderedImage i)
        {
                // Get memory for the signature.
            Color[][] sig = new Color[5][5];
            // For each of the 25 signature values average the pixels around it.
            // Note that the coordinate of the central pixel is in proportions.
            float[] prop = new float[]{1f / 10f, 3f / 10f, 5f / 10f, 7f/10f,9f/10f};
            for (int x = 0; x < 5; x++)
                for (int y = 0; y < 5; y++)
                {
                        sig[x][y] = averageAround(i, prop[x], prop[y]);
                }
             return sig;
         }

         /*
          * This method averages the pixel values around a central point and return the
          * average as an instance of Color. The point coordinates are proportional to
          * the image.
          */
         private Color averageAround(RenderedImage i, double px, double py)
         {

                 // Get an iterator for the image.
                 RandomIter iterator = RandomIterFactory.create(i, null);
                 // Get memory for a pixel and for the accumulator.
                 double[] pixel = new double[3];
                 double[] accum = new double[3];
                 // The size of the sampling area.
                 int sampleSize = 15;
                 int numPixels = 0;
                 // Sample the pixels.
for (double x = px * baseSize - sampleSize; x < px * baseSize + sampleSize; x++)
{
 for (double y = py * baseSize - sampleSize; y < py * baseSize + sampleSize; y++)
 {
         iterator.getPixel((int) x, (int) y, pixel);
         accum[0] += pixel[0];
         accum[1] += pixel[1];
         accum[2] += pixel[2];
         numPixels++;
  }
}
                 // Average the accumulated values.
                 accum[0] /= numPixels;
                 accum[1] /= numPixels;
                 accum[2] /= numPixels;
                 return new Color((int) accum[0], (int) accum[1], (int) accum[2]);
         }



         private double calcDistance(int[][] other)
     {

         // There are several ways to calculate distances between two vectors,
         // we will calculate the sum of the distances between the RGB values of
         // pixels in the same positions.
                 double dist = 0;

                 for (int x = 0; x < 5; x++)
                         for (int y = 0; y < 5; y++)
```

```
                        {
                                int r1 = signature[x][y].getRed();
                                int g1 = signature[x][y].getGreen();
                                int b1 = signature[x][y].getBlue();
                                int r2 = (other[x][y]>>16) & 0xFF;
                                int g2 = (other[x][y]>>8) & 0xFF;
                                int b2 = (other[x][y]) & 0xFF;
                                double tempDist = Math.sqrt((r1 - r2) * (r1 - r2) +
                                        (g1 - g2)* (g1 - g2) + (b1 - b2) * (b1 - b2));
                                dist += tempDist;


                        }
                return dist;
        }

}

public class CompareImage {

        double position;
        int n;
        public CompareImage(double position,BufferedImage image,int midilist[],int n,
                                                        PitchDuration pd)

        {
                //new DisplayImage(image);
               this.n=n;
                this.position=position;
                int treble[]={-1,-6316129,-7500403,-5263441,-1,
                             -1,-2302756,-8224126,-3355444,-1,
                             -65794,-6579301,-7105645,-3487030,-1,
                              -197380,-1710619,-8026747,-11842741,-1,
                              -197380,-1710619,-3026479,-1710619,-1};
                int bass[]={-1,-2829100,-1118482,-1118482,-1,
                              -1,-1184275,-6052957,-1118482,-1,
                              -1,-6908266,-12829636,-1118482,-1,
                              -1,-4802890,-7303024,-1118482,-1,
                              -1,-1118482,-1118482,-1118482,-1,};
                int n2[]={-1,-1710619,-1710619,-3618616,-1,
                              -1,-1710619,-1710619,-7829368,-1,
                              -1,-1710619,-1710619,-7829368,-1,
                              -1,-1710619,-1710619,-7303024,-1,
                              -1710619,-3223858,-3223858,-3223858,-1710619};
                int n4[]={-1,-1710619,-6645094,-5723992,-1,
                              -1,-1710619,-11645362,-1710619,-1,
                              -1,-1710619,-13816531,-1710619,-1,
                              -1,-1710619,-14342875,-1710619,-1,
                              -1,-1710619,-8158333,-1710619,-1,};
                int n8[]={-1,-4473925,-1710619,-1710619,-1,
                              -1,-6184543,-1710619,-4473925,-1,
                              -1,-7303024,-1710619,-15132391,-1,
                              -1,-8421505,-1710619,-15461356,-1,
                              -1,-1907998,-2236963,-2039584,-1};
                int n16[]={-1,-1118482,-1710619,-8947849,-1,
                              -1,-1118482,-9737365,-5197648,-1,
                              -1,-1118482,-7829368,-8487298,-1,
                              -1,-1118482,-9342607,-9803158,-1,
                              -1,-1118482,-7829368,-6250336,-1};
                int n32[]={-1,-1710619,-1513240,-1381654,-6118750,-1,
                              -1315861,-8882056,-12434878,-6052957,-1,
                              -1250068,-1710619,-9145228,-5855578,-1,
                              -1184275,-1250068,-9474193,-5921371,-1,
                              -1184275,-1447447,-10197916,-5460820,-1};
                int r4[]={-1,-1710619,-1710619,-1710619,-1,
                              -1,-1710619,-9474193,-1710619,-1,
                              -1,-1710619,-14211289,-3355444,-1,
                              -1,-1710619,-5131855,-1710619,-1,
                              -3947581,-5263441,-5263441,-5263441,-3947581};
                int s1[]={-1,-1710619,-15263977,-1710619,-1,
                              -1,-1710619,-13158601,-1710619,-1,
                              -1,-1710619,-1710619,-1710619,-1,
                              -1,-1710619,-10263709,-1710619,-1,
                              -1,-1710619,-8882056,-1710619,-1};



           int[] symbol = new int[9];//***WHEN ADDING A NEW SYMBOL,CHANGE LIMIT ****//
           int max=0,pos=-1;

                try{
               // to get rid of medialib error
                System.setProperty("com.sun.media.jai.disableMediaLib", "true");
               //comparison array, image to be compared, its name, no.of images
                FindSymbol streb=new FindSymbol(treble,image,"treble",1);
               symbol[0]=streb.weight();
                //only 1 image for documenting purpose. won't work with just 1
                        FindSymbol sbass= new FindSymbol(bass,image,"bass",1);
                        symbol[1]=sbass.weight();
                        FindSymbol sn8= new FindSymbol(n8,image,"1/8th",1);
                        symbol[2]=sn8.weight();

                        FindSymbol sn16= new FindSymbol(n16,image,"1/16",1);
                        symbol[3]=sn16.weight();

                        FindSymbol sn32= new FindSymbol(n32,image,"1/32",1);
```

```
symbol[4]=sn32.weight();

FindSymbol sn4= new FindSymbol(n4,image,"1/4",1);
symbol[5]=sn4.weight();
FindSymbol sn2= new FindSymbol(n2,image,"1/2",1);
symbol[6]=sn2.weight();
FindSymbol sr4= new FindSymbol(r4,image,"r/4",1);
symbol[7]=sr4.weight();
FindSymbol ss1= new FindSymbol(s1,image,"s1",1);
symbol[8]=ss1.weight();

int i=0;
for(i=0;i<symbol.length ;i++)
{
       System.out.println("cpig"+i+" "+symbol[i]);
       if(symbol[i]>max&& symbol[i]>24)
       {
               if(i==6)
                       if(symbol[i]>750)
                       {
                               max=symbol[i];
                               pos=i;
                       }
               max=symbol[i];
               pos=i;
       }
}
if(pos!=-1)
{
        switch(pos)
        {
        case 0:
                System.out.println("treble");
                break;
        case 1:
                System.out.println("bass");
                break;
        case 2:
                System.out.println("1/8th - duration 1/2");
                for(i=0;i<n;i++)
                {
                System.out.println(midilist[i]);
                pd.add(midilist[i], 0.5,position);
                }
                this.position=this.position+0.5;
                break;
        case 3:
                System.out.println("1/16th - duration 1/4");
```

```
                        for(i=0;i<n;i++)
                        {
                        System.out.println(midilist[i]);
                        pd.add(midilist[i], 0.25,position);
                        }
                        this.position=this.position+0.25;
                        break;
                case 4:
                        System.out.println("1/32th - duration 1/8");
                        for(i=0;i<n;i++)
                        {
                        System.out.println(midilist[i]);
                        pd.add(midilist[i], 0.125,position);
                        }
                        this.position=this.position+0.125;
                        break;
                case 5:
                        System.out.println("1/4th - duration 1");
                        for(i=0;i<n;i++)
                        {
                        System.out.println(midilist[i]);
                        pd.add(midilist[i], 1,position);
                        }
                        this.position=this.position+1;
                        break;
                case 6:
                        System.out.println("1/2th - duration 2");
                        for(i=0;i<n;i++)
                        {
                        System.out.println(midilist[i]);
                        pd.add(midilist[i], 2,position);
                        }
                        this.position=this.position+2;
                        break;
                case 7:
                        System.out.println("Rest 1/4");
                        this.position=this.position+1;
                        break;
                case 8:
                        System.out.println("scale");
                        break;
                }
        }
        else
       System.out.println("Cannot find anything / yet to recognize");
}
catch (IOException e){
```

```
                        e.printStackTrace();
                }




        }
        public double returnposition()
        {
                System.out.print("compare imge" +position);
                return position;
        }

}
```

A.3 Package : javapy

A.3.1 CreateMid.java

```
package javapy;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class CreateMid {

        public CreateMid(int noofnotes,double[][] note,String opfile,int volume,
                                                int tempo) throws IOException
        {
             for(int i=0;i<noofnotes;i++)
             {
                 for(int j=0;j<3;j++)
                 {
                         System.out.print(note[i][j]+ " ");
                 }
                 System.out.println("");
             }
             savef("test.jarp",opfile,volume,tempo,note);
             String cmd[]=new String[2];
             cmd[0]="python";
             cmd[1]="jesmidi.py";
             ProcessBuilder pb=new ProcessBuilder(cmd);
             pb.start();
```

```
            String cmd2[]=new String[2];
            cmd2[0]="vlc";
            cmd2[1]=opfile;
            ProcessBuilder pb2=new ProcessBuilder(cmd2);
            pb2.start();
        }


        public CreateMid(String opfile) throws IOException
        {
                String cmd[]=new String[2];
            cmd[0]="python";
            cmd[1]="jesmidi.py";
            ProcessBuilder pb=new ProcessBuilder(cmd);
            pb.start();

            String cmd2[]=new String[2];
            cmd2[0]="vlc";
            cmd2[1]=opfile;
            ProcessBuilder pb2=new ProcessBuilder(cmd2);
            pb2.start();
        }
         public static void savef(String filename,String opfile,int volume,int tempo,
                                     double[][] note) throws IOException
    {
              BufferedWriter writer =null;
              try
              {
                      writer= new BufferedWriter(new FileWriter(filename));
                      writer.write(opfile);
                      writer.newLine();
                      writer.write(String.valueOf(volume));
                      writer.newLine();
                      writer.write(String.valueOf(tempo));
                      writer.newLine();
                      for (int i=0;i<note.length;i++)
                      {
                              writer.write(String.valueOf((int)note[i][0]));
                              writer.newLine();
                              writer.write(String.valueOf(note[i][1]));
                              writer.newLine();
                              writer.write(String.valueOf(note[i][2]));
                              writer.newLine();
                      }

              }catch(IOException e){
              e.printStackTrace();
              }finally{
```

```
                        if(writer!=null){
                                writer.close();
                        }
                }
        }
}
```

A.4 Package : drawimage

A.4.1 DrawRectangle.java

```
package drawimage;

// Used to draw rectangles over the identified notes in an image

import java.awt.Color;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.awt.image.ColorConvertOp;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

import javax.imageio.ImageIO;

public class DrawRectangle
{
        BufferedImage image=new BufferedImage(10,10, BufferedImage.TYPE_INT_RGB);
        public DrawRectangle(String filename)
        {
                File file = new File(filename);

            try {
                FileInputStream fis = new FileInputStream(file);
                        image = ImageIO.read(fis);
                        //if the image is not in color space, then change it
                        if(image.getType()==0)
                        {
                        BufferedImage newImg = new BufferedImage(image.getWidth(),
                                image.getHeight(),BufferedImage.TYPE_INT_RGB);
                        ColorConvertOp op = new ColorConvertOp(null);
                        op.filter(image, newImg);
                        this.image=newImg;
                        }
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

        }

        public DrawRectangle(BufferedImage image)
        {
                //if the image is not in color space, then change it
                if(image.getType()==0)
                {
                        BufferedImage newImg = new BufferedImage(image.getWidth(),
                                image.getHeight(),BufferedImage.TYPE_INT_RGB);
                        ColorConvertOp op = new ColorConvertOp(null);
                        op.filter(image, newImg);
                        this.image=newImg;
                }
        }

public void drawfig(int x1,int y1, int x2, int y2,Color colo)
        {
                int t;

                for(int i=x1;i<=x2;i++)          //horizontal line
                {
                        for(int k=0;k<2;k++)        // k gives the thickness
                        {
                                t=i;
                                if(i>=image.getWidth())
                                        t=image.getWidth()-1;
                                if(y1+k>=image.getHeight())
                                        y1=image.getHeight()-k-5;
                                if(y2+k>=image.getHeight())
                                        y2=(image.getHeight()-5);
                        image.setRGB(t, y1+k,colo.getRGB());
                        image.setRGB(t, y2+k,colo.getRGB());
                        }
                }
for(int i=y1;i<=y2;i++)          //vertical line
                {
                        for(int k=0;k<2;k++)
                        {
                                t=i;
                                if(i>=image.getHeight())
                                        t=image.getHeight()-1;
                                if(x1+k>=image.getWidth())
                                        x1=image.getWidth()-k-1;
                                if(x2+k>=image.getWidth())
                                        x2=image.getWidth()-k-1;
                                image.setRGB(x1+k, t, colo.getRGB());
                                image.setRGB(x2+k, t, colo.getRGB());
                        }
                }

        }
        public void writeimg()
        {
                new DisplayImage(image);
        }
        public void displaytext(String str,int x,int y)
        {
                Graphics g = null;
                g.drawString(str, x, y);
                image.getGraphics();
        }
}
```

A.4.2 DisplayImage.java

```
  package drawimage;

// Used to Display image in a GUI

import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.IOException;


import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;


import java.awt.BorderLayout;
import java.awt.Dimension;


class MyCanvas extends JComponent {
private static final long serialVersionUID = 1L;
private BufferedImage image;
private JPanel canvas;
public MyCanvas(final BufferedImage image)
{
        this.image=image;
        this.canvas = new JPanel() {
                private static final long serialVersionUID = 1L;
                @Override
                protected void paintComponent(Graphics g) {
                        super.paintComponent(g);
                         g.drawImage(image,0,0, this);
                }
        };
        canvas.setPreferredSize(new Dimension(image.getWidth(), image.getHeight()));
        JScrollPane sp = new JScrollPane(canvas);
        setLayout(new BorderLayout());
        add(sp, BorderLayout.CENTER);
}
}

public class DisplayImage {
        public DisplayImage(BufferedImage image)
        {
                JFrame window = new JFrame();
                window.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
                window.setBounds(0, 0, image.getWidth(), image.getHeight());
                window.getContentPane().add(new MyCanvas(image));
                window.setVisible(true);
        }
}
```

REFERENCES

[1] Arnaud F. Desaedeleer (afd05). Reading sheet music - openomr. *Imperial College London,(University of London),http://sourceforge.net/projects/openomr/.*

[2] Richard Johnson Baugh Earl Gose. Pattern Recognition and Image Analysis. In *Pattern Recognition and Image Analysis*, volume 1, 2011.

[3] Online. Java Docs. `http://docs.oracle.com/javase/tutorial/`.

[4] Online. Stack Overflow- Java. `http://stackoverflow.com/questions/tagged/java`.

[5] Online. Note names, MIDI numbers and frequencies. `http://www.phys.unsw.edu.au/jw/notes.html`, 2005. June 2005.

[6] Online. midiutil- A Python interface for writing multi-track MIDI Files. `https://code.google.com/p/midiutil/`, 2013. December 2013.

[7] Online. PythonInMusic. `https://wiki.python.org/moin/PythonInMusic`, 2013. December 2013.

[8] Online. Note value. `http://en.wikipedia.org/wiki/Note_value`, 2014. Last Edited: 10 March 2014.