CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

# Hardware/Software Co-design of an Optical Music Recognition System

A graduate project submitted in partial fulfillment of
the requirements for the degree of
Master of Science in Computer Engineering

By
Louis Alfred Tacata

May 2021

# Acknowledgments

I would like to express my sincerest gratitude to Dr. Shahnam Mirzaei for introducing me to the world of digital hardware design. From learning finite state machines, to image processing, and to programming Zedboards, to culminate my graduate experience with a project that combines my two greatest passions: music and engineering, is just fitting.

To my friends, thank you for the moral support, and for being there every step of the way. I appreciate every single one of you.

Finally, to my family, thank you for your undying support and extreme patience throughout the course of the project development. Hearing organ sounds in the middle of the night might not be funny at times, but I assure you, it is all worth it.

Table of Contents

List of Figures

List of Tables

Abstract

Hardware/Software Co-design of an Optical Music Recognition System

By
Louis Alfred Tacata

Master of Science in Computer Engineering

The objective of the project presented herein is to provide a co-design approach to the general optical music recognition (OMR) framework. The proposed system aims to be the first step towards a full-fledged hardware implementation of a music recognizer.

The OMR system is partitioned into two subsystems: software and hardware. Utilizing a combination of morphological and template matching techniques, the HOMeR software subsystem processes and converts simple monophonic, homophonic, and piano sheet music images to two numeric arrays corresponding to notes and beats through a graphical user interface. Subsequently, the hardware subsystem applies these array outputs into a custom synthesizer core written for the Zybo Z7 development board, which produces an organ-like audible equivalent of the initial input images through the onboard audio codec.

HOMeR was able to process sheet music images of different types and sizes with high accuracy and fast run times, both for pre- and post- correction.

# 1. Introduction

## 1.1 Motivation

From birds chirping in the morning, to road trip jam sessions, and parents humming lullabies to slumbering toddlers, music is a part of our everyday lives. Throughout the course of history, music has been transmitted and learned through various media. Before the existence of musical notation, oral transmission was the primary medium of preserving music, in which it is passed from one generation to another. This method was prone to variations due to its reliance on sheer memory and improvisation. In the early 11$^{th}$ century, Guido d'Arezzo, a monk and a music theorist, introduced a system that preceded the modern staff notation as it is known today. Scribes played a vital role in the dissemination and preservation of handwritten music knowledge, even with the eventual invention of the printing press [9, 11].

Until the late 1960's, the digitization of musical notation was nonexistent, when the first attempts on automatic recognition of sheet music were performed, signaling the birth of the field of research called *optical music recognition,* or OMR. For the last half a century, researchers have developed various systems to convert scanned and handwritten sheet music images into a machine-readable format, incorporating elements of artificial intelligence such as deep learning and neural networks. One of the main applications of OMR is *replayability*, or the recovery of music through an audible format, which can be taken advantage of in music education.

The motivation behind the project was driven by the idea that by combining the musical components of a punch-hole music box with the scanning capability of a Scantron machine, a complete hardware implementation of OMR can be achieved. For students with limited notation background or reliant on audible cues, a standalone OMR machine integrated in electronic music instruments can be supplemental in their learning process.

## 1.2 Objective

The project's main objective is to provide a preliminary platform towards a complete hardware implementation of a replayable OMR system. The prototype should be able to process solo voice and simple piano sheet music images and produce an audible version of the written music in one attempt. Finally, the system must adopt a co-design methodology, with the software subsystem handling music recognition, and the hardware subsystem handling audio synthesis.

## 1.3 Thesis outline

In the following chapter, the fundamentals of music, OMR, image processing, and audio synthesis will be discussed. In Chapter 3, the proposed adaptation of the optical music recognition framework pipeline is explained, while Chapter 4 describes the synthesizer subsystem. The overall implementation of the entire OMR system is given in Chapter 5, and the results are discussed in Chapter 6. The thesis finishes with Chapter 7, where conclusions, improvements, and future work will be addressed. The list of the test sheet music and their composers, and the detection results are found in Appendix A, while the subsequent Appendices B, C, and D provide the source codes utilized by the project prototype.

# 2. Fundamentals

## 2.1 Music notation theory

The following section contains the musical symbols and elements that will be utilized by the proposed OMR system.

### 2.1.1 Common symbols

#### Staff

Used as reference by all other musical symbols, a staff consist of five horizontal parallel lines and four spaces. It is typically described from bottom to top, that is, the bottom line is the *first line,* and the top line is the *fifth line*. By itself, a staff does not represent pitch; it requires a placement of a clef to have an absolute pitch reference. A type of staff, called *grand staff*, is a combination of two staves intended to be played simultaneously. Both individual and grand staves are supported by the project.

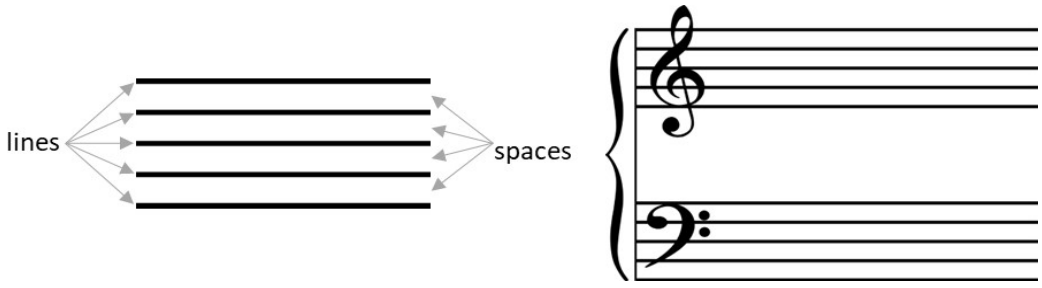



***Figure 2.1-1:*** *(left) Staff and its parts; (right) a typical grand staff [15]*

#### Ledger and bar lines

Like the staff, ledger lines are also pitch reference lines. Located either above or below staff lines, these small horizontal lines, and the spaces they created, indicate pitches that are out of a staff's range. On the other hand, bar lines are vertical lines that divide a staff into *measures*, music notation's unit of time based on time signature. Single and double bar lines indicate a new measure or section, while a bold double bar line typically indicates the end of a notation.

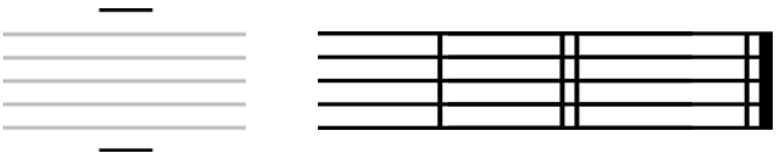


***Figure 2.1-2**: (left) Ledger lines; (right) single, double, and bold double bar lines [15]*

**Notes and rests**

The building blocks of musical notations, notes represent the pitch and duration of sounds, while rests represent the duration of silence in music notations. Notes are divided in three parts: *noteheads* that indicate the pitch location, *flags* that indicate the type of note, and *stems* that connect the two other parts together. In the case of consecutive notes, they can be connected by using horizontal or diagonal lines called *beams.* These parts are all relevant to the detection of notes in the subsequent section.

Two note classifications based on notehead shape are used in this paper: open and closed. Open notes consist of whole and half notes, while closed notes consist of the quarter, eighth, sixteenth, thirty-second, and sixty-fourth notes. Their relative duration values are defined in Table 2.1-1.

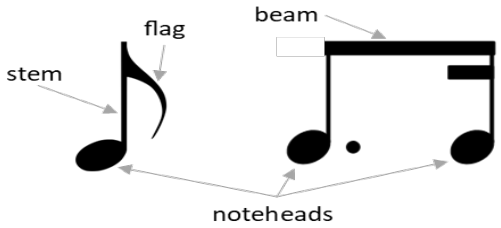



***Figure 2.1-3:** Parts of individual and consecutive notes*

**Clefs**

A clef defines what pitches are being represented by the lines and spaces of a given staff. By assigning a clef to a staff, one of the staff lines serve as a reference point for the rest of the lines

and spaces. The most common clefs used today are the treble, bass, and alto clefs. The reference points mentioned earlier are defined in Table 2.1-2.

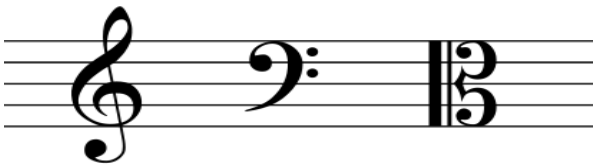


***Figure 2.1-4:*** *(left to right) Treble, bass, and alto clefs [15]*

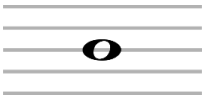
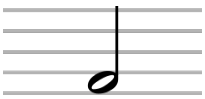
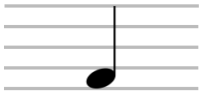
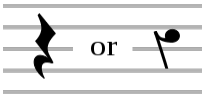
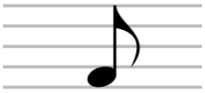
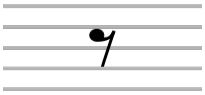
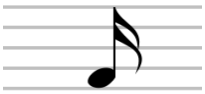
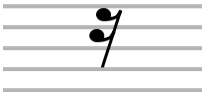
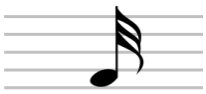
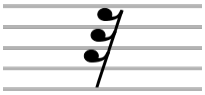
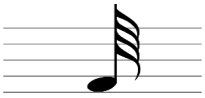
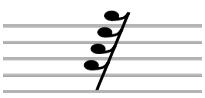
| Name | Note | Rest | Value |
|---|---|---|---|
| Whole |  |  | 1 |
| Half |  |  | $\frac{1}{2}$ |
| Quarter |  |  | $\frac{1}{4}$ |
| Eighth |  |  | $\frac{1}{8}$ |
| Sixteenth |  |  | $\frac{1}{16}$ |
| Thirty-second |  |  | $\frac{1}{32}$ |
| Sixty-fourth |  |  | $\frac{1}{64}$ |

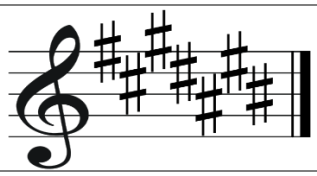
***Table 2.1-1:*** *Notes and rests names, and their relative duration values [15]*

| Clef type | Reference line | Pitch | Middle C location |
|---|---|---|---|
| Treble | 2nd | G4 | 1st ledger line below staff |
| Bass | 3rd | C4 | 3rd staff line |
| Alto | 4th | F3 | 1st ledger line above staff |

***Table 2.1-2:*** *List of relevant clef parameters*

**Key signatures**

Final reference pitches are assigned by key signatures, which are sets of *sharps* or *flats* that indicates that affected staff lines or spaces will be played one *semitone* higher (sharp) or lower (flat). Located just on the right of a clef, a maximum of seven sharps or flats can be present in a key signature. A notation's *key* is defined by the number of sharps or flats, as shown in Table 2.1-3.

| **Sharp** | | |
|---|---|---|
|  | | |
| *Number of #* | *Added #* | *Key (major/minor)* |
| 0 | - | C/A |
| 1 | F# | G/E |
| 2 | C# | D/B |
| 3 | G# | A/F# |
| 4 | D# | E/C# |
| 5 | A# | B/G# |
| 6 | E# | F#/D# |
| 7 | B# | C#/A# |

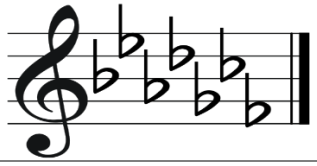
| **Flat** | | |
|---|---|---|
|  | | |
| *Number of ♭* | *Added ♭* | *Key (major/minor)* |
| 0 | - | C/A |
| 1 | B♭ | F/D |
| 2 | E♭ | B♭/G |
| 3 | A♭ | E♭/C |
| 4 | D♭ | A♭/F |
| 5 | G♭ | D♭/B♭ |
| 6 | C♭ | G♭/E♭ |
| 7 | F♭ | C♭/A♭ |

***Table 2.1-3:*** *Key definitions based on number of sharps and flats [15]*

**Time signature**

The number of *beats* in each measure is determined by the notation's time signature. Typically consisting of two fraction-like numbers, the top number indicates how many beats are in a measure, while the bottom number tells which note receives one beat. For instance, in a 4/4-time signature, there are four beats in a measure, and a quarter note receives one beat, whereas in a 6/8-time signature, the measure contains six beats, with the eighth note receiving one beat.

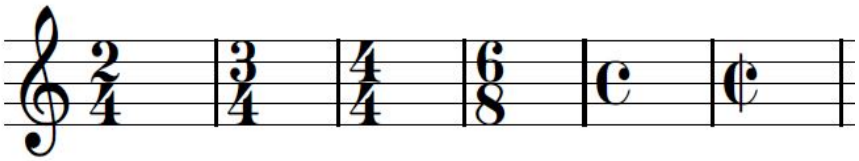


***Figure 2.1-5:*** *Commonly used time signatures; the last two are called common (4/4) and cut (2/2) times, respectively*

**Tie and slur**

Note relationships are important in music notations. When two or more notes are connected by a tie, it indicates that these notes have the same pitch, and must be played as a single note. As shown in Figure 2.1-6, two quarter notes tied together is played as a half note. On the other hand, notes that are bounded by a slur must be played smoothly and connected. While ties and slur might look alike, the difference lies within the start and end notes, in which in slurs, the end notes have different pitches, as also shown in Figure 2.1-6.

**Chords**

A group of two or more notes of the same type played simultaneously is called a *chord*. While typically seen in the bass-clef side of a grand staff, the project can process chords of up to eight combined notes, regardless of its staff.

***Figure 2.1-6:*** *Note relationships (left to right): tie, slur, chord [15]*

### 2.1.2 Musical elements

**Pitch**

The simplest way to define pitch: it is the classification of how high or low a note is. Each note is represented by a *frequency*, the higher the frequency, the higher the note, and the same case goes for low frequencies. In the project, the pitches are denoted using *scientific pitch notation*, in

which each pitch is defined as a combination of a letter name and an *octave* number, as shown in Figure 2.1-7.

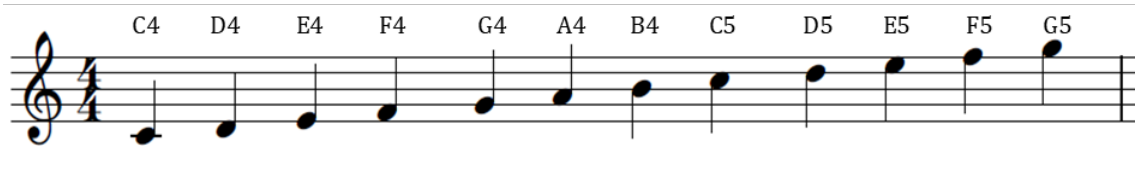



***Figure 2.1-7:*** *Visualization of the C major scale, with the corresponding pitch names*

**Accidentals**

A *semitone* is the smallest interval in musical notations. The pitch of a given note can be modified using accidentals. The most common accidentals in use today are *flat*, *sharp*, and *natural*. A sharp accidental increases the current note pitch one semitone higher, while a flat accidental decreases the pitch one semitone lower. On the other hand, a natural accidental cancels the effect of a previous accidental. For instance, assuming a G-clef is attached to the staff in Figure 2.1-8, the first space's pitch is F4. By applying a sharp accidental, the note's pitch changes to F#4. After applying a natural accidental, the pitch returns to its original value, F4. Lastly, a flat accidental attached to the note changes the pitch to F♭4, which is equivalent to E4.

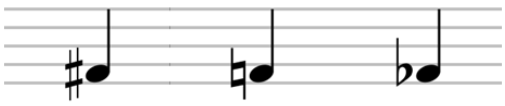


***Figure 2.1-8:*** *(left to right) Sharp, natural, and flat accidentals [15]*

**Duration and articulation**

Another important musical element is *duration*, the amount of time a note, rest, or a composition lasts. With duration also comes *articulation*, which determines how and how long to play a given note. While notes and rests have their individual relative values, certain symbols can prolong or cut these values. When a dot is added on the righthand side of a note, its relative value

must be increased by half. Also, when a dot is placed on top or bottom of a note, it is called *staccato*, and it reduces the note's duration in half. Finally, a *fermata,* placed on top or bottom of a note, indicates that the note must be played twice as long.

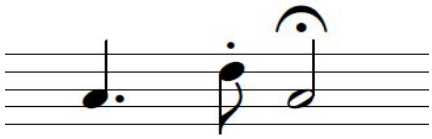


***Figure 2.1-9:*** *(left to right) Augmentation dot, staccato, and fermata*

### 2.1.3 Notation types

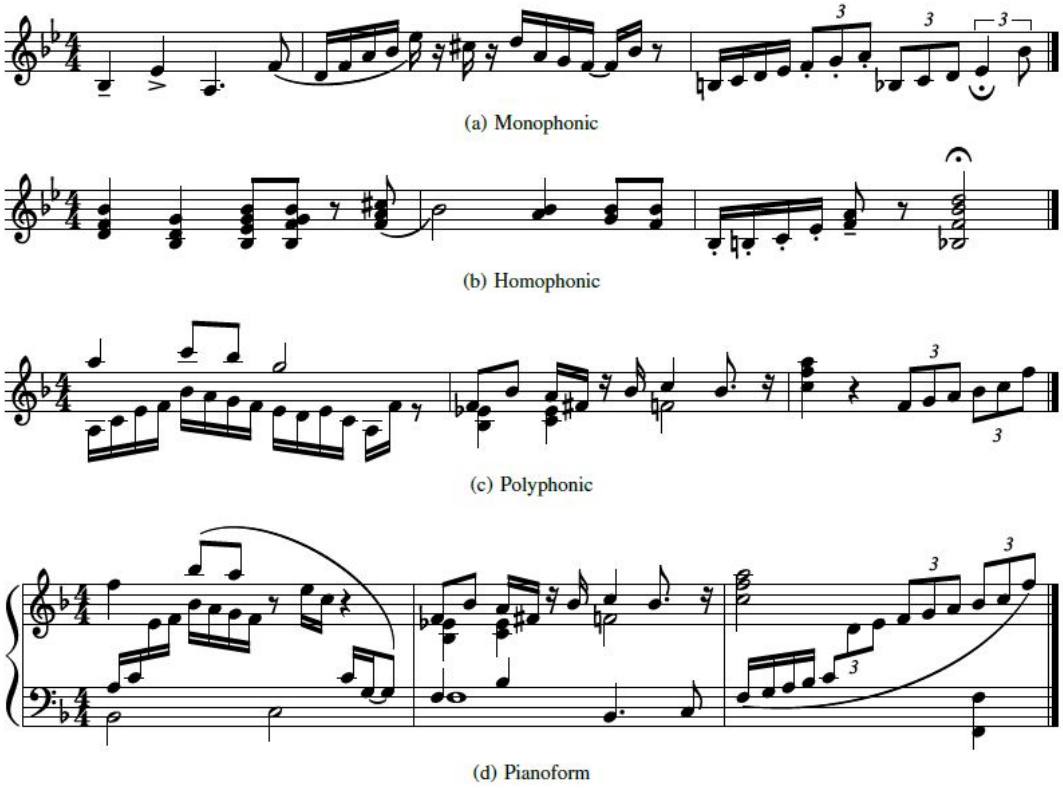



***Figure 2.1-10:*** *Four common types of music notation [6]*

Byrd and Simonsen provided a characterization of sheet music inputs complexity based on the number of *voices* present in a staff, and the presence of multiple staves [4]. Calvo-Zaragoza, et. al however, saw the multiple staves' presence as a nonfactor, thus, providing a revised characterization of OMR inputs: [6]

a) *Monophonic*: single voice in a single staff; played one note at a time

b) *Homophonic*: variation of monophonic; presence of chords but remains single voice

c) *Polyphonic*: multiple voices in a single staff, typically two voices

d) *Pianoform*: multiple voices in multiple staves; cannot be separated into individual staves

In its current version, the prototype can read simple monophonic, harmonic, and pianoform notations. It has not been tested on polyphonic sheet music images, but in theory, the project must be able to read simple polyphonic notations.

## 2.2 Optical music recognition framework

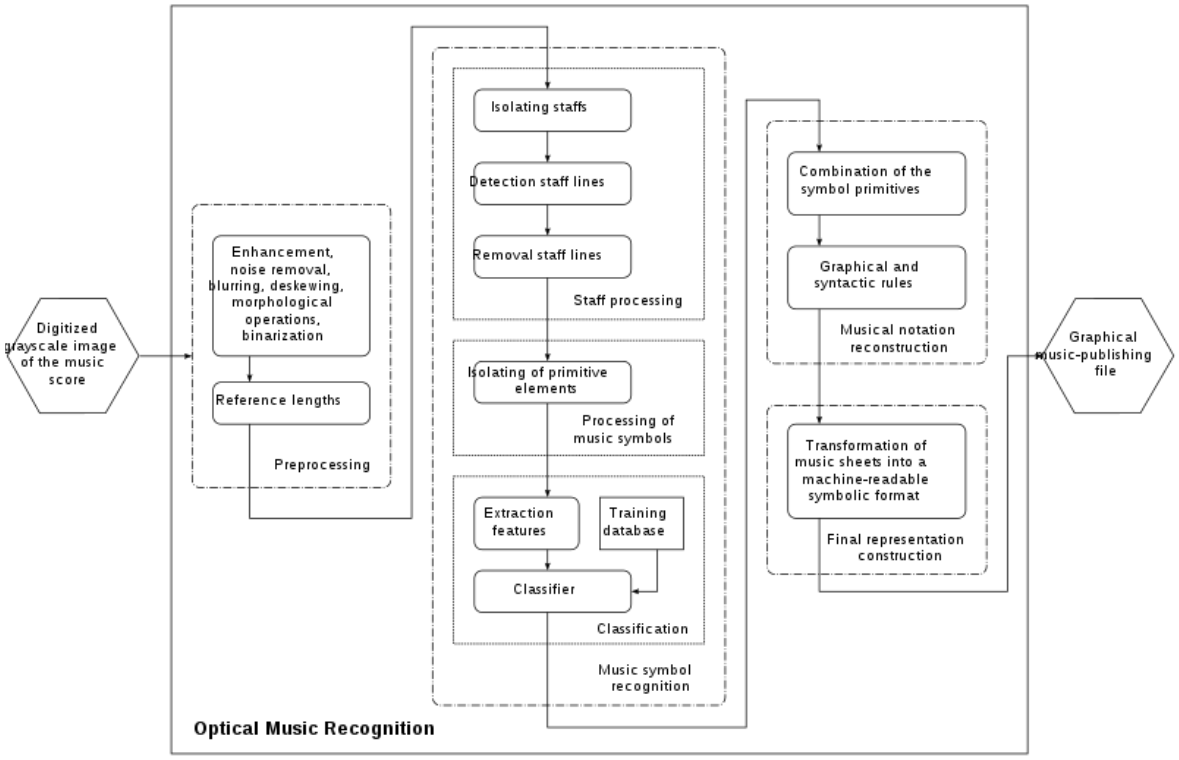



***Figure 2.2-1:*** *Framework of a typical OMR system [13]*

Bainbridge and Bell's OMR framework proposal consisted of a four-stage pipeline: staff line identification, musical object location, musical feature classification, and music semantics, with image enhancements as intermediate steps in between stages [3]. Rebelo provided a

modification to the existing framework, integrating the image enhancement step in a preprocessing stage, and the staff processing included in the music symbol recognition phase [*12*]. An overview of the modified OMR framework is as follows:

a) *Preprocessing*: rotation, cropping, and binarization of input image, obtaining reference lengths

b) *Music symbol recognition*: staff detection and removal, primitive elements detection

c) *Musical notation reconstruction:* notation recreation using the detected symbols and application of graphical and semantic rules

*d) Final representation construction:* encoding of sheet music into a machine-readable format

The last two stages can be combined in one step, as encoding can be performed as the notation is being reconstructed, as is the case with the project prototype. The project's three-stage pipeline culminates to two C header files containing note and beat arrays, respectively.

## 2.3 Image processing

The success of the proposed OMR system relies on the application of various image processing techniques. Listed below are relevant techniques utilized by the project.

### 2.3.1 Connected component analysis

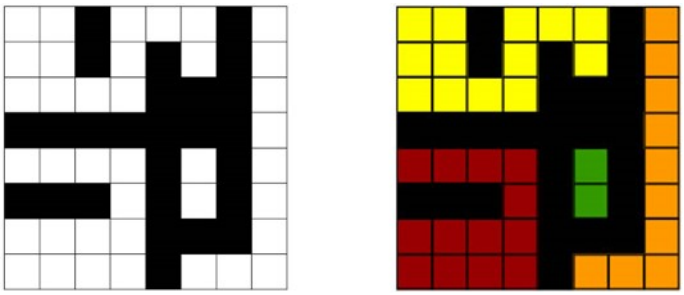


***Figure 2.3-1:*** *(left) Binary image; (right) connected components indicated by colored pixels*

In binary images, *connected components*, or CC, are sets of pixels that form a connected group. The analysis of connected components consists of giving detected these components unique

labels. Properties such as area, eccentricity, and centroids, among others, can be obtained for each labeled component. These properties can be used as references in performing *image segmentation*.

**Eccentricity and circularity**

The eccentricity and circularity properties are helpful in analyzing rounded CCs. *Eccentricity* can be defined as the ratio of the distance between the ellipse's foci and its center. A circle has an eccentricity value of 0, while a line segment, 1. Similarly, *circularity* refers to the roundness of a component, that is, the closer an object is to a perfect circle, the closest the value goes to 1. These properties can be used in detecting open note holes.

**Bounding boxes and centroids**

Two other properties that are of vital importance are bounding boxes and centroids. By obtaining the smallest rectangle that can fit a given component, the bounded segment can be isolated from the original image. From the bounding box parameters, the centroid can be obtained by calculating the means of the X and Y endpoints of the box. Centroid calculations are significant in calculating distances between staff lines and note heads during pitch assignments.

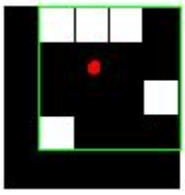


***Figure 2.3-2:*** *Visualization of a bounding box and centroid of a discontiguous object*

### 2.3.2 Morphological operations

Binary morphology is defined by probing a binary image with a predefined second binary image, called a *structuring element*, or SE, and depending on how the SE fits or misses the primary image, records the resultant image. The morphological operators utilized in the project will be defined and described.

**Erosion**

In the morphological context, erosion is synonymous to shrinking. For each foreground pixel, if the superimposed SE matches the foreground pixels underneath it, the pixel directly below the SE center is unchanged, otherwise, the pixel turns into a background pixel. The middle image in Figure 2.3-3 is the result of the erosion of the left image using a line SE with height of 3, rotated 90°.

**Dilation**

The *dual* of erosion, dilation is synonymous to expanding. For each foreground pixel, the background pixels directly underneath the superimposed SE are turned to foreground pixels. Using Figure 2.3-4 as an example, the middle image is the result of superimposing a 3x3 square SE on the original image.

**Opening**

The opening operation is defined as the dilation of the erosion of a given binary image, using the same SE for both operations. The rightmost image in Figure 2.3-3 is the opening of the original image, as it is the dilation of the middle image, which is the erosion result. This operation is helpful in isolating certain objects and removing salt noise in a binary image. For this example, vertical lines were the resultant of the image opening.

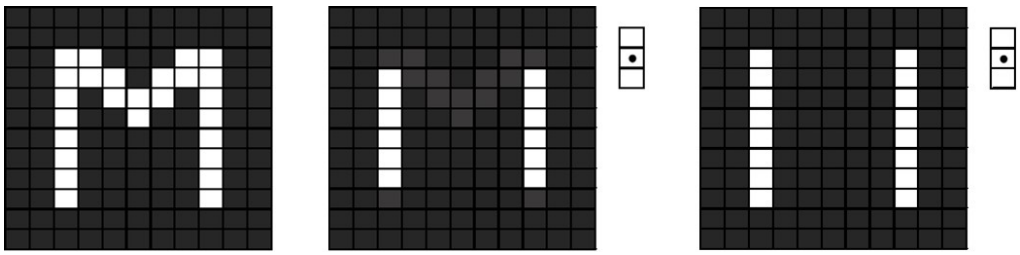


***Figure 2.3-3:*** *(left) Original image; (middle) erosion result of the original image; (right) dilation of the middle image, resulting in the opening of the original image*

**Closing**

The closing operation is the dual of opening; it is the erosion of the dilation of a given image. Using the example in Figure 2.3-4, the rightmost image is the result of closing the original image using the same 3x3 square SE. This operation is beneficial in removing gaps in small objects and removing pepper noise in binary images.

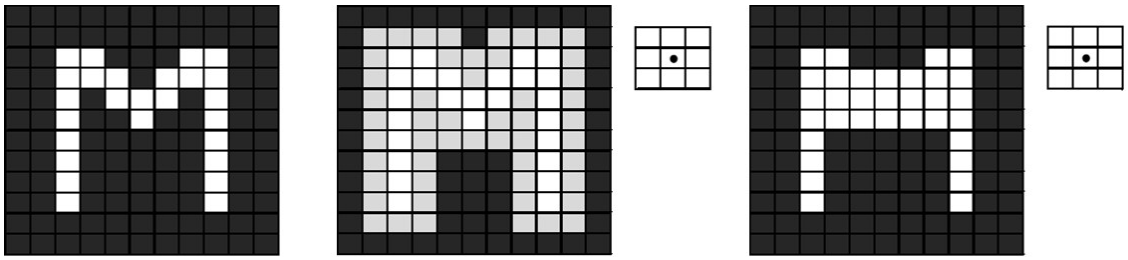


***Figure 2.3-4:*** *(left) Original image; (middle) dilation result of the original image; (right) erosion of the middle image, resulting in the closing of the original image*

### 2.3.3 Template matching

The method of searching and finding a copy of a template image within an input image is called *template matching.* A modification of this method is performed in the project prototype, in which the correlation between an isolated potential symbol and the resized template image is calculated. If the correlation factor is equal or greater than the specified threshold, then the potential symbol and the template are deemed matching.

## 2.4 Audio synthesis

Audio synthesis is the process of producing sound without the presence of an acoustic source. Two of the most common sound synthesis methods that are used in the project prototype are discussed in this section.

### 2.4.1 Direct digital synthesis

Signal generation is made possible by creating arbitrary waveforms from a single reference clock through the direct digital synthesis methodology. It comprises of four parts: a frequency

control word (FCW) generator, a numerically-controlled oscillator (NCO), a phase-to-amplitude converter, which can be replaced by a lookup table of a desired waveform, and a digital-to-analog converter to produce the desired audio signal.

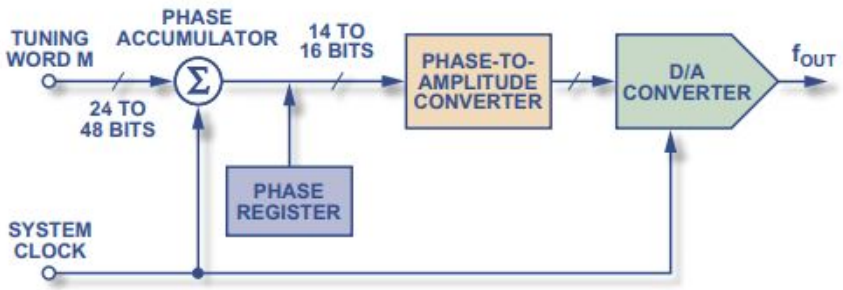



***Figure 2.4-1**: Schematic of a digital direct synthesizer [2]*

### 2.4.2 Additive synthesis

From the name itself, additive synthesis is a technique that combines waveforms, typically sine waves, to create a certain timbre. In Figure 2.4-2, the top three tracks represent the notes A4, C#5, and E5, respectively. By calculating the sum of these waveforms, the A major chord waveform is generated, as shown in the bottom track.

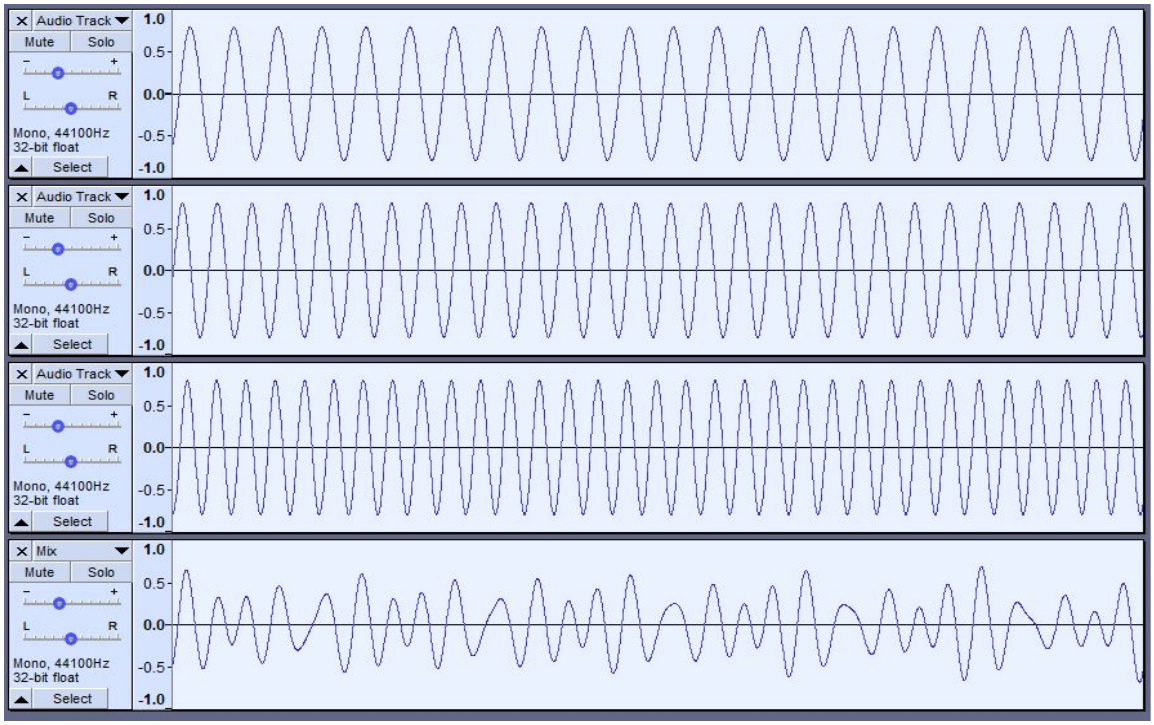



***Figure 2.4-2:** Visualization of additive synthesis using the software Audacity*

# 3. OMR Framework Adaptation

This chapter defines the project adaptation of the current OMR framework. The preprocessing stage is described in one section, while the music symbol recognition and reconstruction stages are further broken down into subsequent sections.

## 3.1 Input image parameters

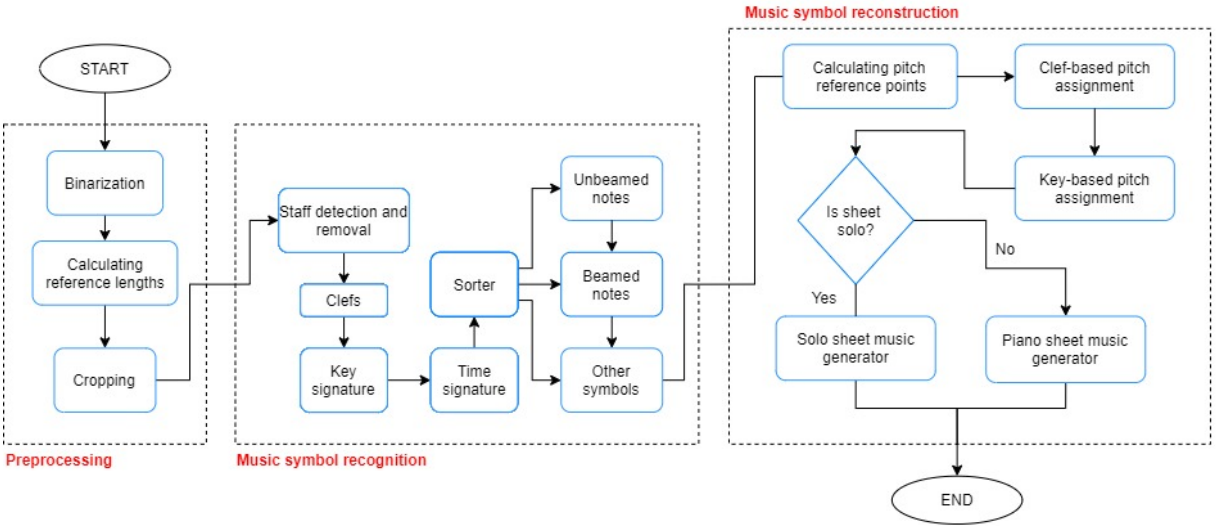



***Figure 3.1-1:*** *Flowchart of the proposed OMR framework adaptation*

The proposed system was designed around forty grayscale and color input images with low to medium-high music symbol density. Some of the scores have multiple pages, and size dimensions vary in every image. Also, based on the quality grades presented by Byrd and Simonsen, all images are of Grade 1 quality, that is, “near flawless, containing totally contiguous symbols, and with no line skewing” [4]. The list of the input titles, composers, and notation type is found in Appendix A.

## 3.2 Image preprocessing

Before any detection is performed in an input image, it must undergo preprocessing. The necessary steps described in the section contribute to a better recognition in the following stages: binarization, calculating reference lengths, and cropping.

### 3.2.1 Binarization

Binarization is the process of converting a grayscale or color image into a logical equivalent. This process alone has been a separate topic among OMR researchers as they attempt to develop binarization methods specifically for musical notations. In this implementation however, a global thresholding algorithm is used, called *Otsu's method*. Otsu argued that by minimizing intra-class variance within the image's gray-level histogram, a single intensity threshold can be obtained. [10]

### 3.2.2 Reference lengths

The implementation of the proposed OMR system heavily relied on two values: staff space height (SH) and line height (LH). Fujinaga explained that these two values can be estimated in a notation image by performing *run-length encoding*, a lossless data compression method that stores data into single counts. [8] This process is performed in every single column of the binary image, and the results are recorded in two histograms, one for white runs, and another for black runs. The most common white run represents the estimated space height, while the most common black run, line height. Throughout the rest of the paper, space heights and line heights will be referred to as SH and LH, respectively.

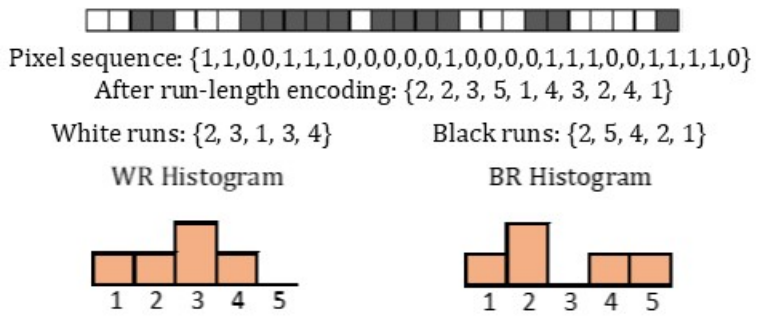



***Figure 3.2-1:*** *An example of run-length encoding in a binary strip*

### 3.2.3 Cropping

To eliminate unnecessary information in scores such as score titles and copyrights, cropping is a must. Calculating the sum of black pixels in the image columns, called *horizontal projection,* is performed in the binarized image, and any y-location with a projection greater than half of the maximum projected value will be recorded. The top and bottom bounds of the cropped images are the y-locations $(5 * (SH + HL))$ away from the minimum and maximum locations from the previous step. As for the left and right bounds, they are the x-locations 20 pixels away from the leftmost and rightmost foreground pixels.

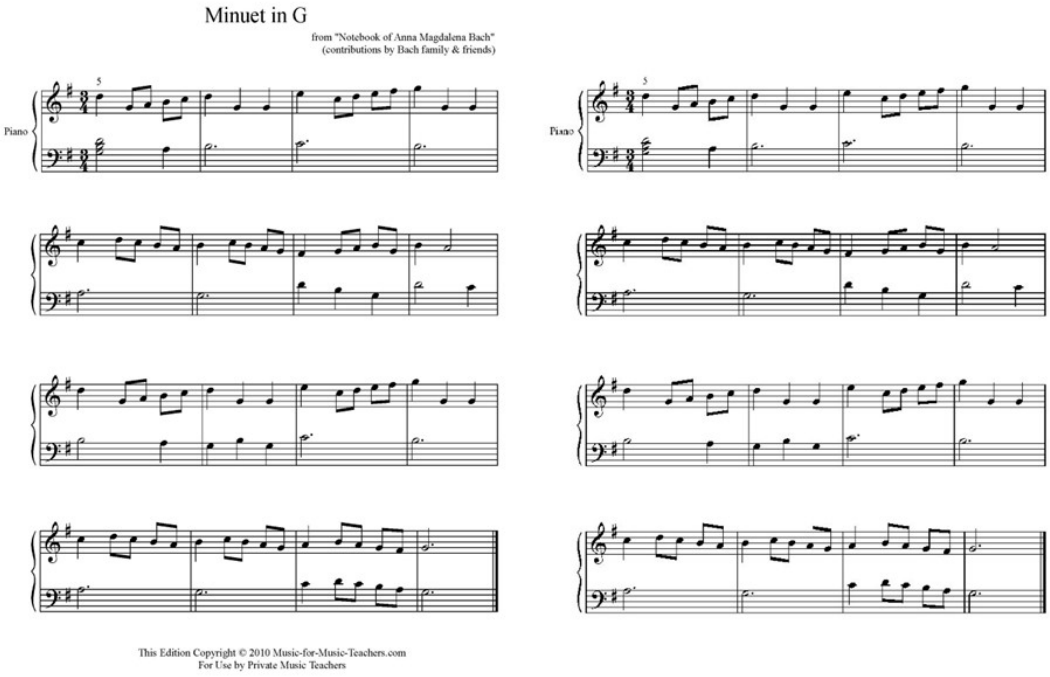



***Figure 3.2-2:*** *(left) Minuet in G before cropping, (right) same image after cropping*

## 3.3 Staff line localization and removal

Like binarization, the localization and subsequent removal of staves is a major subject among OMR researchers. While some OMR systems prefer to leave the staves [5], most of the existing systems contain staff removal algorithms. The removal process described in this section is based on horizontal projections and connected component analysis.

### 3.3.1 Horizontal projection

The horizontal projection process described here is like the projection performed in Section 3.2.3. In this case however, the threshold value for a potential staff line value is increased to $2/3$ of the cropped image width. Once these y-locations are isolated, run-length encoding is performed to determine the actual height of each staff line before its removal.

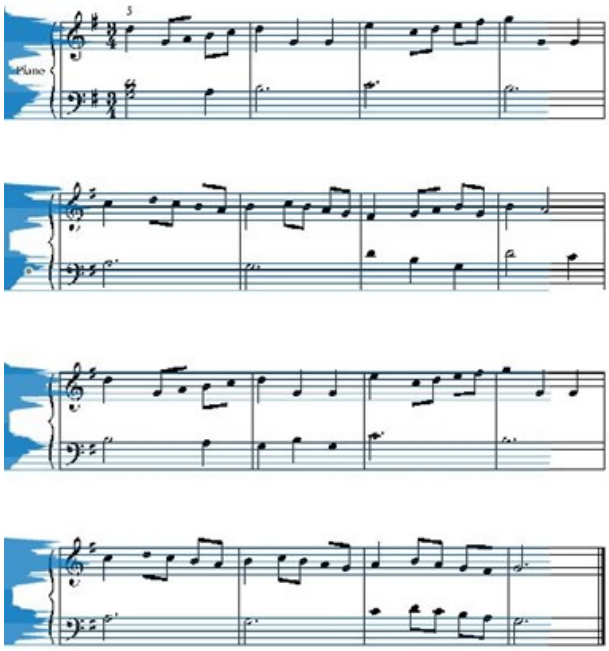


***Figure 3.3-1:*** *Overlay of the cropped score and the horizontal projection*

### 3.3.2 Obtaining segmentation locations

Since the project is designed to process individual staff or grand staves, segmentation locations are calculated in this step. If only one staff exists in the image, the dividers are set to the y-locations $(4 * (SH + LH))$ away from the first and fifth lines. Otherwise, a two-step process ensues.

Initially, each set of ten staff lines is treated as 'grand staff', and the first divider set is calculated as the midpoint between the $10^{th}$ line of the current set and the $1^{st}$ line of the following set. Afterwards, these sets are checked to see if they are indeed grand staves. As shown in Figure

2.1-1, a distinct feature of this staff type is the presence of bar lines extending to both staves. Therefore, if these lines are not detected, a divider is added between the 5th and 6th staff lines. Finally, the divider locations are sorted to ensure proper section segmentation.

### 3.3.3 Removing the staff lines

By outright removing the staff lines without any symbol segment attached on either top or bottom, it risks the degradation of symbols, especially open notes bounded by two staff lines. To avoid such problem, the image undergoes a three-step process. First, the cropped image is inverted to have a white foreground and eroded using a disk SE of radius 1. After labeling the connected components present, if condition (a) and either condition (b) or (c) is satisfied, the CC is saved:

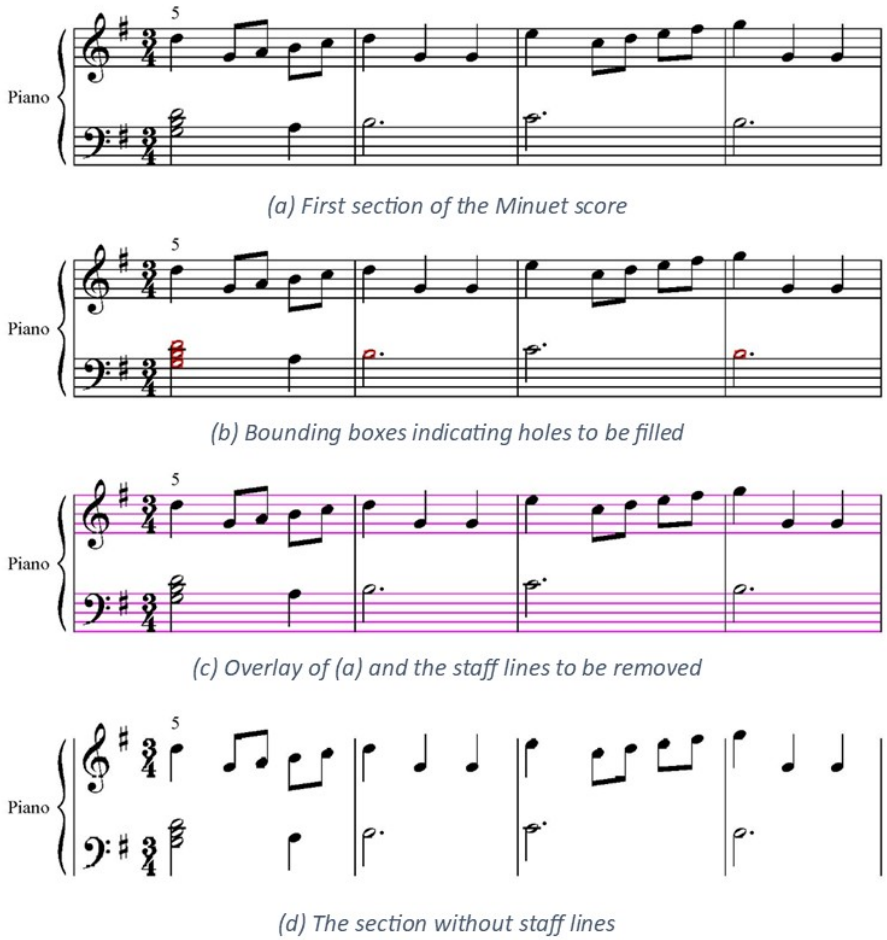



***Figure 3.3-2:*** *Visualization of the staff removal process*

a) *Area*: width between $(0.7 * SH)$ and $(1.75 * SH)$, and height between $(0.7 * SH)$ and $SH$

b) *Potential half note holes*: Eccentricity factor between 0.9 and 1, and circularity factor between 0.6 and 0.9

c) *Potential whole note holes*: Eccentricity factor between 0.6 and 0.675, and circularity factor between 1.05 and 1.2

After dilating the image containing the CCs using a disk SE of radius 1, the same image is integrated into the cropped image, effectively filling holes. Each staff line will be traversed horizontally, and if there is no symbol segment attached to either top or bottom of the current staff line, the staff segment is removed. Once the traversal is complete, the holes are put back, resulting in an image without staff lines.

## 3.4 Initial symbol classification

After segmenting the score into individual sections, the initial music symbol classification takes place. In this section, the utilized dataset will be discussed, as well as detection of clefs, time and key signatures, and the sorting of the remaining undetected symbols.

### 3.4.1 Dataset importation

| Symbol Type | Sample Template | Number |
|---|---|---|
| Clef |  | 15 |
| Time signature |  | 26 |
| Whole note |  | 8 |
| Rest |  | 7 |
| Dot |  | 7 |
| Accidental |  | 5 |
| Fermata |  | 4 |
| Slur |  | 4 |
| Tie |  | 15 |

***Table 3.4-1:*** *Description of symbols in the utilized dataset*

Ninety-one total bitmap images of musical symbols are used as templates, with the majority being time signatures and clefs. These symbols are extracted from sheet music images created using the score-writer software MuseScore. Every single symbol has a corresponding set of parameters which are extracted during the template matching process.

### 3.4.2 Clefs

In the prototype implementation, it is assumed that clefs only appear in the left-hand side of sheet music images, which is true for most of the current existing scores. Therefore, the *region of interest,* or ROI in this detection part is the section bounded by the leftmost edge of the image and the x-location $(30 * SH)$ to its right. A closing of the image with a rectangular SE with a height-width pair of [5,6] is performed to ensure the connection of the dots in bass clefs.

After labeling the resultant image, any component with height between $(3 * SH)$ and $(9.5 * SH)$ and width between $(2 * SH)$ and $(4.5 * SH)$ undergoes template matching with the clef dataset. If the correlation factor of 0.5 or greater is met, the clef parameters are saved, and the reference points defined in Table 2.1-2 are assigned.

### 3.4.3 Key signatures

There are two ROIs in this detection phase. In the key signature detection of the first section, the ROI is between the right edge of the detected clef(s) and the x-location $(10 * SH)$ to its right. A preliminary closing of the section with a vertical line SE of length 3 is performed to ensure the integrity of the accidental shapes. If the labeled symbol's height is between $(2 * SH)$ and $(3 * SH)$ and width between $(0.75 * SH)$ and $(2 * SH)$, it undergoes template matching with the key dataset. If the correlation factor of 0.5 or greater is met, the key parameters are saved, and the rightmost edge of the detected key is recorded. If a CC has a height between $(1.75 * SH)$ and $5.5 * SH)$ and width between $(1.5 * SH)$ and $(3 * SH)$, it will be assumed as a 'time signature',

and its left bound is recorded as the next stage bound. In the case of no keys present, the next stage bound is the same as the initial bound.

For the subsequent sections, the ROI is between the clef's right edge and the bound value obtained from the first section. The detection algorithm is the same as defined above. The total keys are counted; if the staff is a grand staff, the total keys are divided by two. Finally, based on this count, the key signature is defined using Table 2.1-3.

### 3.4.4 Time signatures

Time signature detection is much like the clef time signature. The ROI is between the bound obtained from the key signature detection stage, and $(4 * SH)$ to its right. A closing with a rectangular SE with size [6, 8] is performed to connect the segments forming a potential time signature symbol.

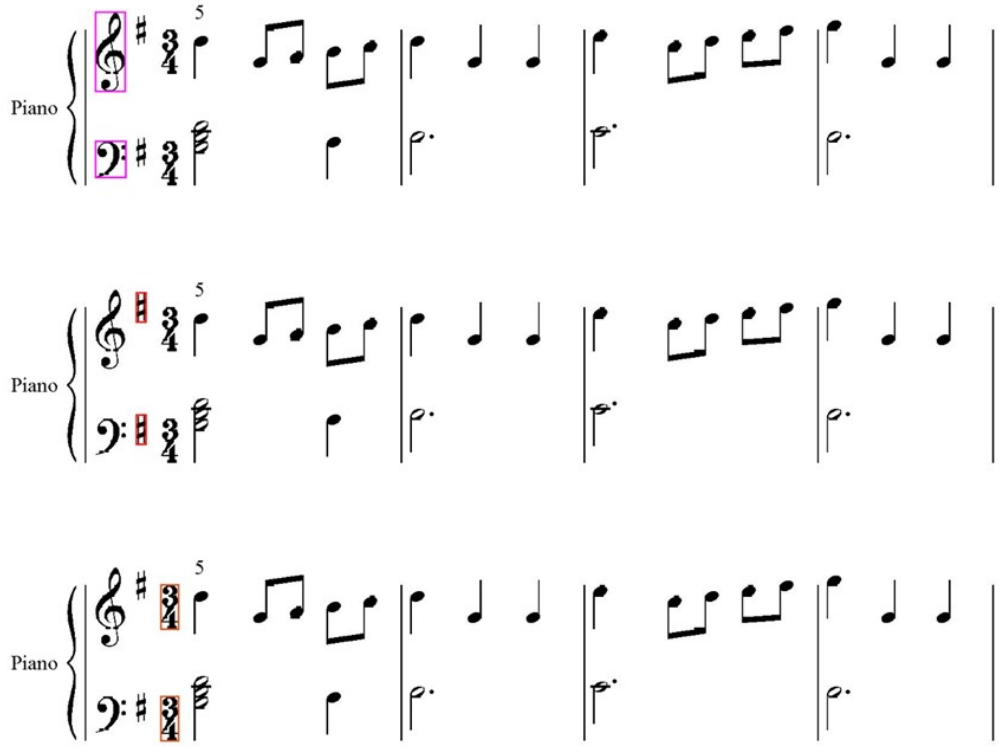



***Figure 3.4-1:*** *(top to bottom) Results of the clef, key signature, and time signature detections*

Any labeled component with height between $(1.75 * SH)$ and $(6 * SH)$ and width between $(1.375 * SH)$ and $(4 * SH)$ undergoes template matching with the time signature dataset. If the correlation factor of 0.6 or greater is met, the time signature parameters are saved, its rightmost edge is recorded for the next stage.

### 3.4.5 Note sorting

The ROI is between the time signature's right edge and the section's right edge. After labeling the image, for the first two sorting passes, each of the requirements stated in Table 3.4-2 must be met.

| Type | Boundary requirement |
|---|---|
| Unbeamed notes | $(2 * SH) \leq h \leq (15 * SH)$ and $(1.375 * SH) \leq w \leq (3.5 * SH)$, or $(SH \leq h \leq (1.5 * SH)$ and $((1.375 * SH) \leq w \leq (3.25 * SH))$ and $(circularity > 0.2))$ |
| Other symbols | $(h < (3 * SH)$ and $w < (20 * SH))$, or $(h < (3 * SH)$ and $w < (20 * SH))$, or $(h \leq SH)$ |

***Table 3.4-2:*** *Boundary requirements for unbeamed notes and other symbols*

The first sorting pass detects potential unbeamed notes. An opening using a vertical line SE of height $(1.75 * SH)$ is performed to detect stems on the current component. If a stem is detected, or the circularity parameter is met, the CC is saved. All potential unbeamed notes are removed from the original image to avoid re-classification in the next pass.

The second pass detects other symbols such as rests, bar lines, and dots, among others. In this case, if the boundary requirement is met, the symbol is saved and removed from the original image.

The third pass does not automatically save the entire image as a beamed note section, because some symbols that are missed by the previous passes might be misclassified. Since beamed notes have at least two stems, symbols with stem counts other than two are sent to the

other symbol section. Figure 3.4-2 shows the sections that are generated by the sorting algorithm overlaid to the original section.

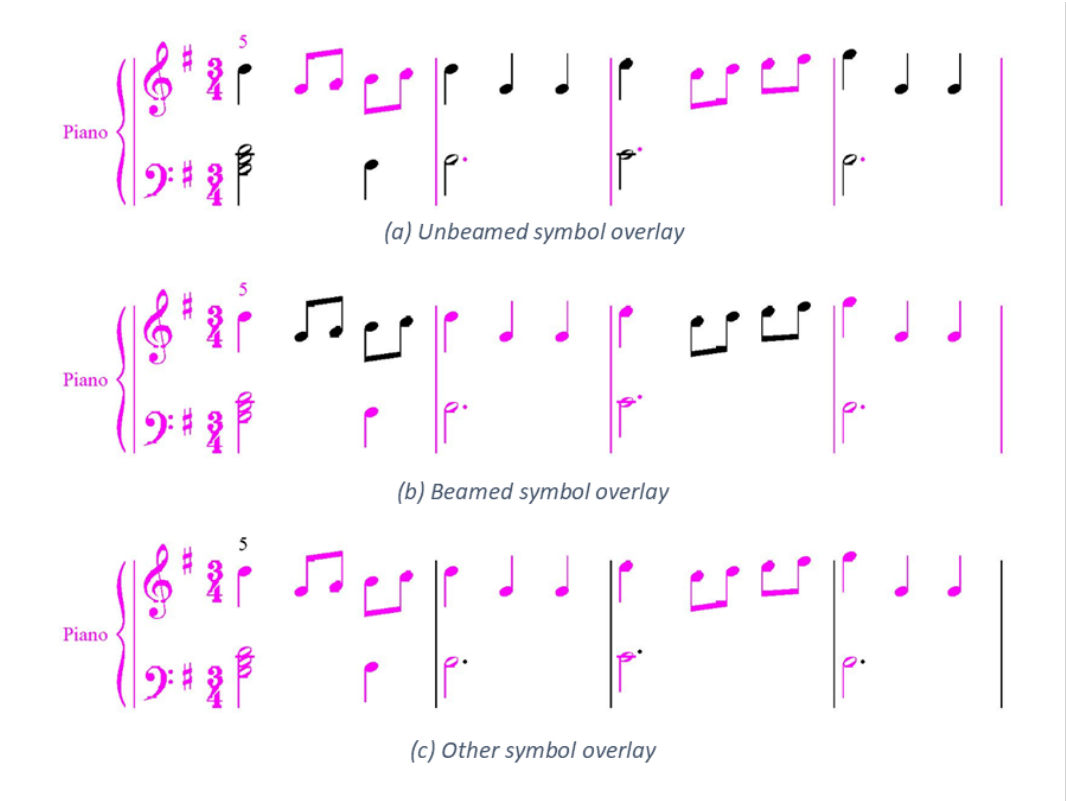



***Figure 3.4-2:*** *Resultant images of the sorting process*

## 3.5 Unbeamed notes

From the name, any notes or chords that are not connected by beams fall under the *unbeamed notes* category. Other than potential single whole notes, the detection in this stage is purely morphological.

### 3.5.1 Ledger line detection and removal

A closing using rectangular SE with size $[2, (2 * SH)]$ is performed to the unbeamed section isolate potential ledger lines. Since the closing might produce unwanted lines from adjacent notes, if the area of the resultant component is less than $(12 * SH)$, it is an assumed ledger line, and its y-location is recorded. The process for removing ledger lines is like the staff removal

process described in Section 3.3.3. In this case, however, only a partial removal is performed. Since ledger lines protrude from notes or chords, only the lines outside of the main notehead body is eliminated, as shown in Figure 3.5-1(b).

### 3.5.2 Closed notes

After the ledger lines have been removed, each connected component is isolated into a blank image, then undergoes a three-step process to verify it falls under the closed note category. Two main splits: solo whole notes and stemmed notes.

**Stem detection**

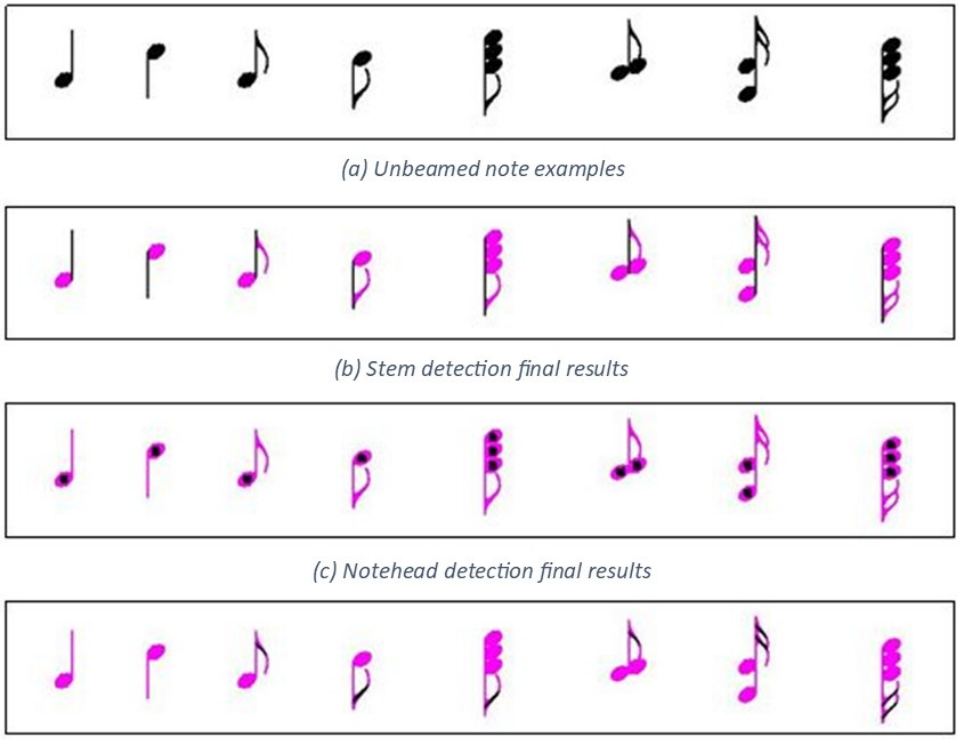
An opening is applied to the image using a vertical line SE with height $((3 * SH) + (2 * LH))$, isolating the component's stem. However, in the case of chords with at least three notes, the opening might yield two stems. Therefore, two more openings are applied to the potential note image, with vertical line SEs of length $(4.25 * SH)$ and $(1.75 * SH)$, respectively. If the stem yield is more than one, the CC is likely a whole note chord.

**Notehead detection**

The notehead detection step classifies a note whether it is closed or open. Using a line SE with length $SH$, rotated 30°, an erosion is applied to the image. Since there can be size differences between sheet music images, two cases of enhancement are applied to the resultant erosion. If the sheet SH is $\leq 10$, it is opened using a disk SE with radius 2, otherwise, the radius is changed to 3. Once enhanced, the notehead's area is calculated, and if its value is greater than $(8 * SH)$, it is possible that there are more than one notehead in the component. Thus, the stem detected from the previous step serves as a divider between the noteheads, which yields the correct notehead amount in the ROI.

**Flag detection**

Like the previous step, the flag detection step relies on the overall note geometry. After the noteheads are temporary eliminated in the image, an opening of the stem/flag image follows suit using a line SE with height $(1.25 * SH)$, rotated at $-45°$. If there are no flags detected in the first pass, another opening is performed on the same image using the same SE, with the angle of rotation set at 45°. The total flag count is obtained after the second pass, and it determined the type of closed note from quarter (0 flags) to sixty-fourth (4 flags).



*(a) Unbeamed note examples*

*(b) Stem detection final results*

*(c) Notehead detection final results*

*(d) Flag detection final results*

***Figure 3.5-1:*** *Overall unbeamed note detection process*

### 3.5.3 Open notes

Potential open notes are CCs that have stem counts not equal to one, or no notehead was detected from the previous section. The overall detection is divided into two parts: half notes and whole note chords, and individual whole notes.

**Half notes and whole note chords**

By just filling out holes and performing a logical XOR between the resultant and the original images, because of the ledger line presence, the image must undergo multiple hole-counting passes. The first pass isolates the complete holes from the first hole-filling process; that is, holes with area $\geq (6 * SH)$ for sheets with $(SH > 15)$ and area $\geq (4.5 * SH)$ for smaller sheets.

The second pass on the other hand, ensures that ledger lines that pass through the notes are small enough to connect the smaller holes detected by the first pass. The image is opened using a line SE with length $(1.375 * SH)$, then eroded the resultant with another line SE, this time with length $(0.9 * SE)$, leaving a very small segment of the ledger line. To ensure that the new holes are saved, CCs must have an area $\geq (5 * SH)$ for sheets with $(SH > 10)$ and area $\geq (2 * SH)$ for smaller sheets.

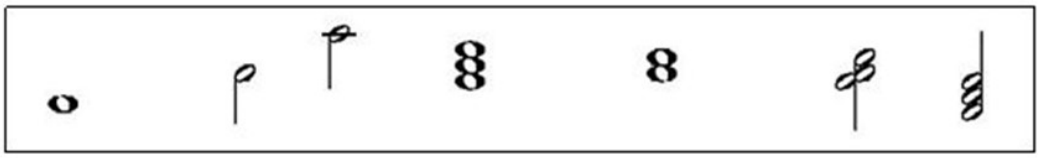
Lastly, some small horizontal lines might go undetected during the second pass. Therefore, a final pass is performed. Any saved CCs whose area is $\geq (4 * SH)$ are the final set of note holes.

**Individual whole notes**

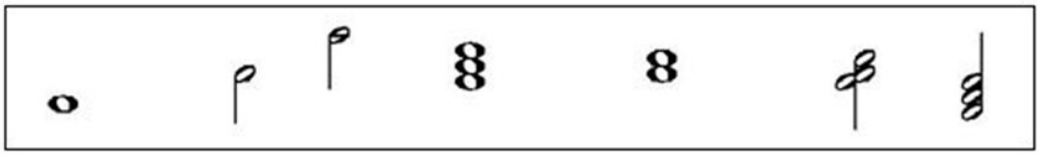
Unlike all other unbeamed notes, the single whole note is the only note that undergoes template matching. However, it still goes through the open notehead detection algorithm explained above, but only needs one pass to obtain the desired hole.

### 3.5.4 Final unbeamed note parameters

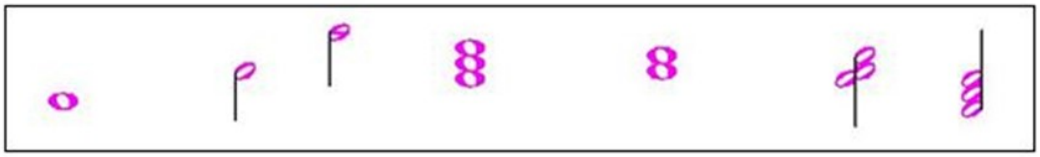
The total count of unbeamed notes is the sum of the detected note heads and note holes, with the classification based on flag counts. Compared to the previous detection stages, the bounding boxes only contain the heads or holes, as shown in Figure 3.5-3. Finally, the unbeamed note parameters are sorted based on their x-locations.



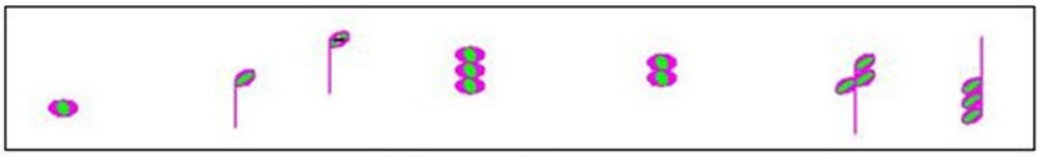
(a) Beamed note examples



(b) Ledger line removal



(c) Stem detection final results



(d) Hole detection final results

***Figure 3.5-2:*** *Overall open note detection process*

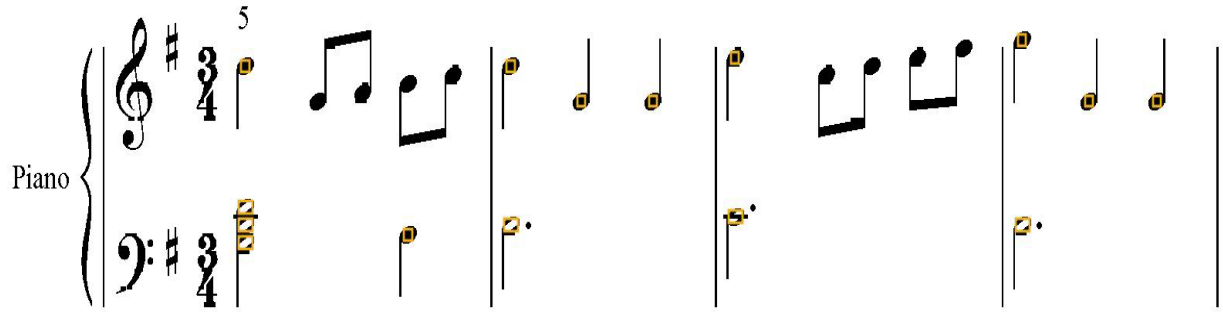



***Figure 3.5-3:*** *Unbeamed note detection results from the Minuet section*

## 3.6 Beamed notes

The presence of beams added more complexity to the note detection process as multiple notes are grouped. Therefore, the section is divided in two parts: beam segmentation and type classification. Some operations such as ledger removal and notehead detection were defined in the previous section, thus, it will only be mentioned briefly.

### 3.6.1 Beam segmentation

The first step towards complete beam segmentation is the temporary separation of the noteheads and the rest of the beamed notes. It is done in three consecutive morphological operations: opening using a diamond SE with radius ($\lfloor 0.5 * \mathrm{SH} \rfloor$), dilation using a square SE with radius 1, and another opening using a vertical line SE with length ($\lfloor 0.5 * \mathrm{SH} \rfloor$). After noise removal, the resultant image consists of only the beams and stems, as shown in magenta in Figure 3.6-1(b). By performing a logical XOR using the previous image and the original section, only the noteheads remain, as shown in green in Figure 3.6-1(b).

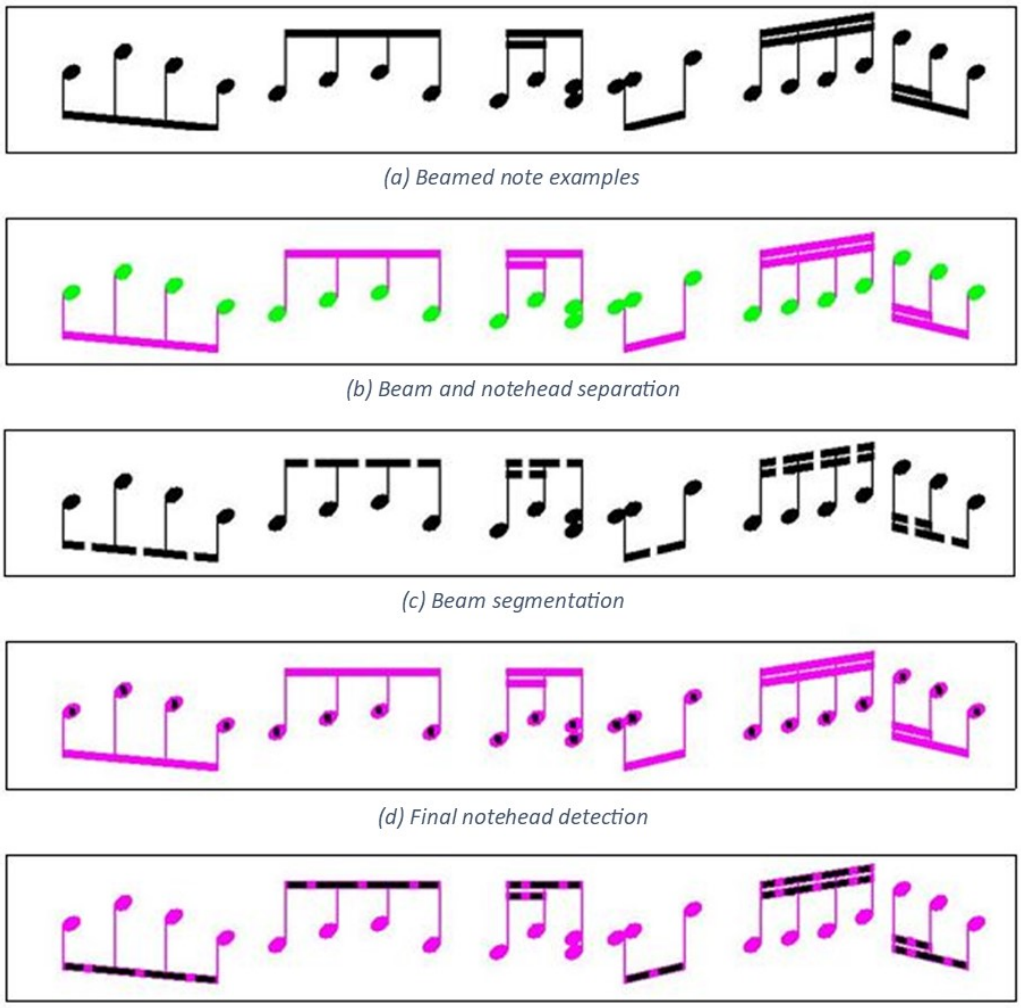
Afterwards, each beam/stem group is processed individually. A temporary opening of the stems using a vertical line SE of length $SH$ is performed, yielding only the stems. The segmentation points are calculated by obtaining the midpoints between two stem pairs. Finally, 3 pixels from each side of the midpoint value is 'cut', producing a segmented group of notes, as shown in Figure 3.6-1(c).

### 3.6.2 Type classification

After labeling each individual note, the classification process is like its unbeamed counterpart, with the difference in flag detection. To ensure disconnect between adjacent beams, the image is eroded using a disk SE of radius 1. After removing the stems, gaps in between beams might exist, therefore, a closing using a line SE with length ($\lfloor 0.5 * \mathrm{SH} \rfloor$) is performed.

### 3.6.3 Final beamed note parameters

The total count of beamed notes is the sum of the detected note heads and note holes, with the classification based on beam counts. The beamed note parameters are also sorted based on x-locations



(a) Beamed note examples

(b) Beam and notehead separation

(c) Beam segmentation

(d) Final notehead detection

(e) Final beam detection

***Figure 3.6-1:*** *Overall beamed note detection process*

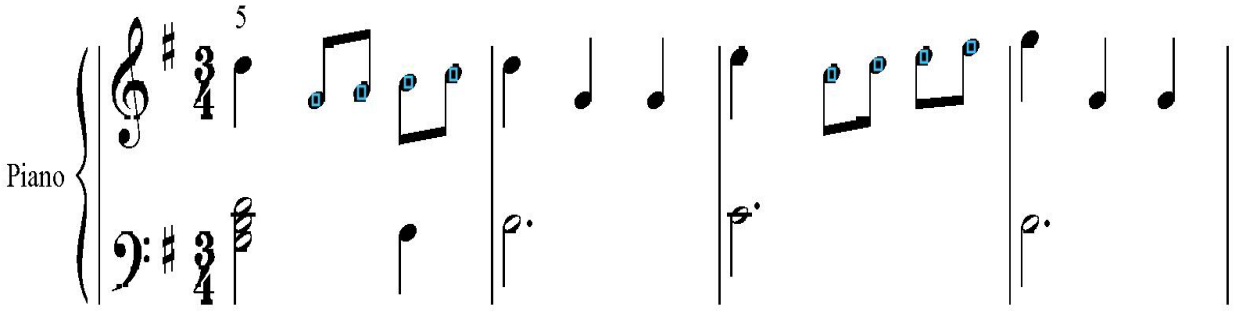



***Figure 3.6-2:*** *Beamed note detection results from the Minuet section*

## 3.7 Other symbols

Any non-note symbol is considered *other* symbol, such as rests, measure numbers, and chord names, among others. The detection process is divided in three parts: vertical symbols, horizontal symbols, and ledger/bar lines.

### 3.7.1 Vertical symbols

The first step before any detection occurs is the staff height calculation, which is the difference of the maximum and minimum staff x-locations. Afterwards, a preliminary bar line removal is performed through the opening of the section using a rectangular SE with size $[(staffheight - 5), 2]$.

While not necessarily the detection of only vertical symbols, it is called as such because of the closing of the non-bar line image using a vertical SE of length $SH$. If the current CC's bounding box parameters meet one of the boundary requirements, it will undergo template matching with the corresponding dataset (dots or combined).

| Type | Boundary requirement |
|---|---|
| Dots (augmentation/staccato) | $(3 \leq h, w \leq (0.9 * SH)$ and $\lvert h - w \rvert = 0 \; or \; 1$ |
| Rests, accidentals, fermatas | $(3 \leq h \leq (3.5 * SH)$ and $(3 \leq h \leq (3.5 * SH))$ |

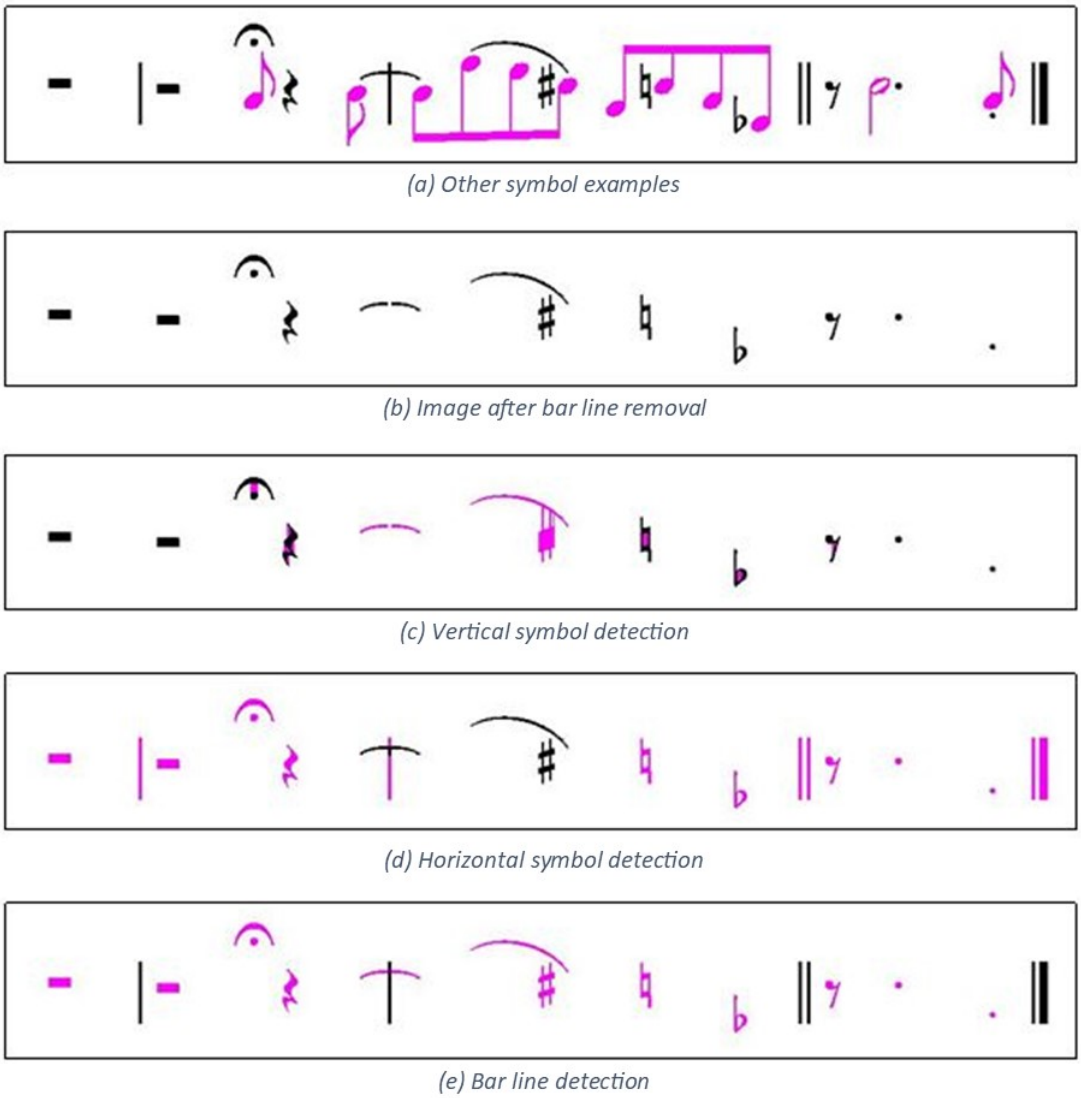
***Table 3.7-1:*** *Boundary requirements for vertical symbols*

If the potential dot correlation factor is $\geq 0.5$, or $0.6$ for other symbols, the CC in question is saved and removed from the original image. Figure 3.7-1(c) shows an example of this detection step.

### 3.7.2 Horizontal symbols

Once the vertical symbols are removed, all that remains are symbols that might have been disconnected due to bar line removal. Therefore, run-length encoding is performed in the image with only bar lines to determine the maximum bar line width. Afterwards, a temporary closing of

the disconnected section is performed using a line SE with length $(barline + 3)$. By performing a logical AND to the isolated bar line and closed images, only the gaps remain. Finally, the gap is integrated into the disconnected section to re-connect the components.



*(a) Other symbol examples*

*(b) Image after bar line removal*

*(c) Vertical symbol detection*

*(d) Horizontal symbol detection*

*(e) Bar line detection*

***Figure 3.7-1:*** *Overall other symbol detection process*

| Type | Boundary requirement |
|---|---|
| Ties and slurs | $h > (2 * SH)$ |
| Dots (augmentation/staccato) | $(3 \leq h, w \leq (0.9 * SH)$ and $\lvert h - w \rvert = 0 \; or \; 1$ |
| Rests, accidentals, fermatas | $(3 \leq h \leq (3.5 * SH)$ and $(3 \leq h \leq (3.5 * SH))$ |

***Table 3.7-2:*** *Boundary requirements for horizontal symbols*

The overall detection of horizontal symbols is the same as the vertical symbols, with the addition of tie and slur dataset. If the potential tie/slur correlation factor is $\geq 0.5$, the CC is saved and removed from the original image. Figure 3.7-1(d) shows an example of this detection step.

### 3.7.3 Ledger and bar lines

The remaining relevant symbols to be detected are ledger and bar lines. Any CC that has an area $\leq (20 * SH)$ is removed to avoid false detections. The resultant image is opened using a line SE of length $(2 * SH)$, and their y-locations are recorded. In the case of bar lines, a closing using a line SE of length $SH$ is performed to separate single from double bar lines. This final step is demonstrated in Figure 3.7-1(e).

### 3.7.4 Final non-note symbol parameters

The symbols classified and detected from all three detection stages are sorted based on x-locations. The bounding boxes for the classified symbol contain the actual symbols, except for the flat accidental, wherein the bounding boxes surround the hole.

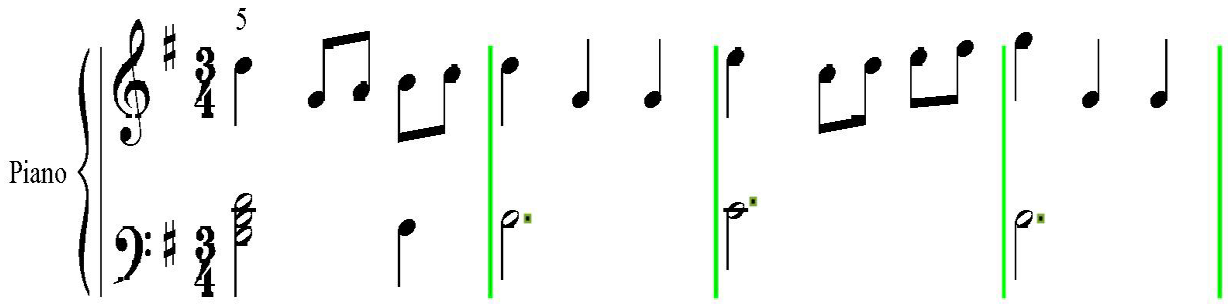



*Figure 3.7-2: Non-note symbols detection results in the Minuet section*

## 3.8 Initial notation reconstruction

The first part of the music symbol reconstruction stage is the initial reconstruction, which consists of the reference points calculation and staff pitch assignments. This section serves as the preprocessing for the subsequent music generation.

### 3.8.1 Reference points calculation

Temporary images were filled using the staff and ledger line locations (if present) from the music symbol detection stage for centroid calculation. Afterwards, the staff space centroids are obtained by calculating the midpoints between two staff line pairs, including the spaces above and below the staff or ledger lines (if present).

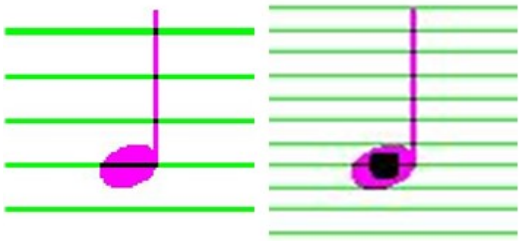


***Figure 3.8-1:*** *(left) Staff centroids in green; (right) Combined staff/ledger/space centroids*

### 3.8.2 Staff pitch assignment

The overall staff pitch assignment is a two-step process: clef-based assignment that provides the base pitches, and key-based assignment that provides the final pitch values. Before the assignment phase, however, a descending note array based on natural pitches is generated, with the first element the highest note on a piano, C8.

**Clef-based pitch assignment**

First, the reference pitch and line location are obtained from the clef parameters, shown in Figure 3.8-2(a), Afterwards, the staff location is cross-referenced between the natural pitch array and centroid arrays to generate the initial pitch assignment index: $(pitch\ array\ index - centroid\ array\ index + 1)$. Finally, the natural pitches are assigned during centroid traversal, as shown in Figure 3.8-2(b). The pitches and its note equivalent are stored as reference points for the next pitch assignment step.

### Key-based pitch assignment

The distance between the key centroid and the centroid array is calculated. The index of the minimum calculated distance represents the line where the key lies above. Depending on the key type, all notes belonging to the line's pitch class is either increased or decreased by one semitone. Looking at Figure 3.8-2, the first key is a sharp located at the fifth staff line. Therefore, all instances of F are sharped, that is, F4 becomes F#4 and F5 becomes F#5. Similarly, the second key lies on staff space 3, therefore, C5 becomes C#5, as shown in Figure 3.8-2(c).

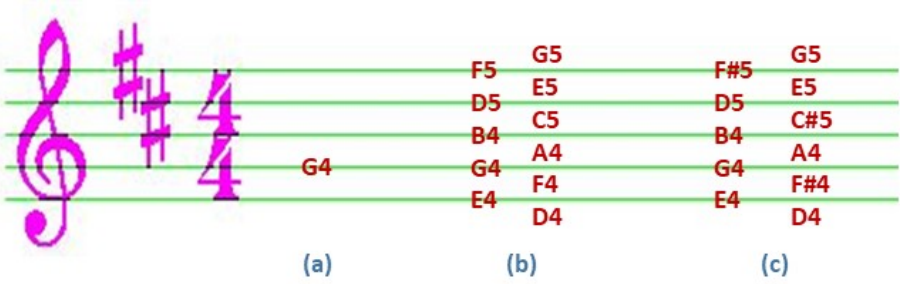



***Figure 3.8-2:*** *Step-by-step visualization of the staff pitch assignment process*

### Pitch assignment organization

The assignment process described only works for individual staves. That said, in the case of grand staves, the top staff is processed first, followed by the bottom staff. The final pitch and note assignments are concatenated to generate the complete pitch and note sets.

## 3.9 Music generation

Before applying the musical rules to the complete set of detected symbols, their centroids are first calculated. Afterwards, the set is segregated in two main subsets: the subset containing notes, rests, and bar lines, and the subset containing the rest. While both solo and piano scores share most of the processes such as note/beat pitch assignment and tie/slur locator, some processes are limited to one score type.

### 3.9.1 Rest preprocessing

Applicable only to piano scores, the preprocessing of whole and half rests achieves two things: ensures the correct assignment of silence and perform value correction. Looking at Figure 3.9-1, there are three possible whole/half cases present in each measure: one whole and one half, two halves, or two wholes. Initially, because of template matching, either half or whole rest is detected as whole, therefore, a correction is needed based on staff location and rest type.

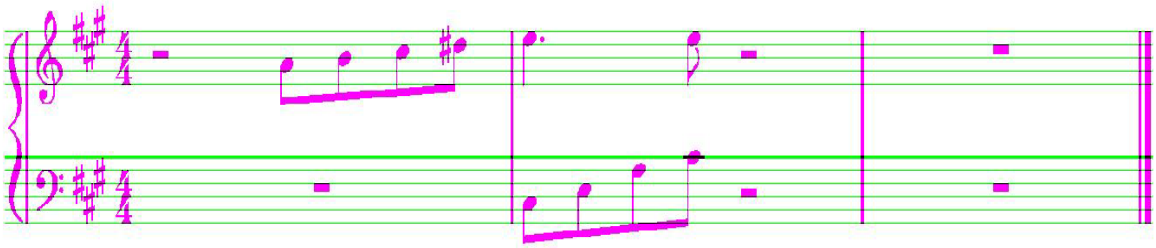


***Figure 3.9-1:*** *Sample section used for rest preprocessing*

In the case of one whole rest and one half rest, the half rest is assigned the value 0.5, while the whole rest is considered 'visited', that is, it will not be processed further in the subsequent steps. On the other hand, if two aligned half rests are present, only the beat values are modified. Finally, in the case of two whole rests existing, one of them will be counted as 'visited', and the remaining rest will have a value based on the time signature.

### 3.9.2 Note/rest/bar line processing

After the preprocessing step, all unvisited notes, rests, and bar lines will be processed based on their x-locations. If the current symbol is a note, the pitch and beat are assigned based on minimum centroid distance and default values, respectively.

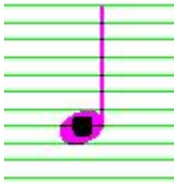


***Figure 3.9-2:*** *Quarter note in a treble clef; pitch G4 and beat 0.25*

On the other hand, if the current symbol is a rest, the pitch is set to 0, and the beat is based on the values obtained from the dataset. Finally, if a bar line is detected, it cancels the effects of accidentals in the previous measure. An example of the processing is shown in Figure 3.9-2.

### 3.9.3 Accidental locator

Accidentals are always located on the left-hand side of notes, therefore, any non-note symbol that is located within $(5 * SH)$ to the left and $(7 * LH)$ from either top or bottom of the note centroid is checked. If the symbol is indeed an accidental, the staff pitch is modified based on the accidental type and location.

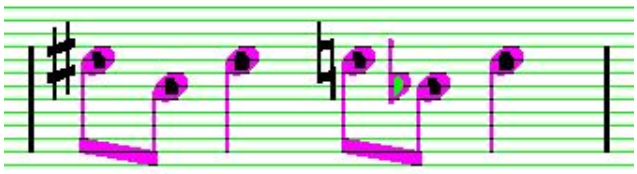


***Figure 3.9-3:*** *Visualization of the accidental locator process*

### 3.9.4 Augmentation dot locator

Augmentation dots are typically located on the right-hand side of notes or rests. Any non-note symbol within $(2.5 * SH)$ to the right, $(0.75 * SH)$ to the top, and $(0.25 * SH)$ to the bottom of the note/rest centroid is checked. If the symbol category is detected as a "dot", the duration of the symbol is increased by half.

### 3.9.5 Vertical symbol locator

Two main types of vertical symbols are observed: articulation symbols and other notes. Solo sheets support both types, while piano sheets only support other notes.

#### 3.9.5.1 Articulation symbols

The symbols that fall under this category are fermatas and staccato dots. Any non-note symbol within $(3 * SH)$ of either top or bottom and $(3 * LH)$ of either left or right of the note centroid. If the symbol located is a staccato dot, the note's duration is decreased by half, followed

by silence with the same duration, while a fermata increases the duration by twice its value. In the case of homophonic sheets, if either symbol appears in a chord, the entire chord is played based on the symbol type.

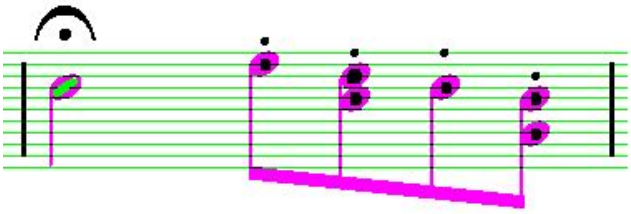


***Figure 3.9-4:*** *Beats after symbol locations: 1, followed by four sets of (0.0625, 0.0625) representing corresponding pitch and silence, respectively*

#### 3.9.5.2 Other notes and rests

Any notes or rests that are on top or below the note being processed and within $(2 * SH)$ on either side are considered part of a "chord". To avoid redundancies in processing, once the notes are located, they will be processed based on the presence of accidentals and augmentation dots and will be marked 'visited'. In the case of rests, the pitch value will be assigned as 0, and is combined with the rest of the chord pitch values.

***Figure 3.9-5:*** *Chord samples, with ties and slurs present*

### 3.9.6 Tie/slur locator

A parameter called *notebreak* determines if a short pause is to be added to a note during playback, and by default, it is set to 1. If a tie or slur edge is detected within $(0.5 * SH)$ to the left or right of the note centroid, notebreak is set to 0. In addition, if a note is caught in between a slur, the notebreak at that moment is also set to 0.

### 3.9.7 Note and beat array transcription

The final outputs of the proposed OMR are two distinct arrays, one for pitches and one for beats. The following section demonstrates how the symbols will be translated to C arrays that will be read by the synthesizer. For the solo staff case, a maximum of three notes can be played together, while for the grand staff case, eight total notes.

#### 3.9.7.1 Single staff

While mentioned in the previous section that the program supports chords in solo staffs, most of the solo input scores are strictly monophonic, hence, the transcription is straightforward, as shown in Figure 3.9-6. Each measure is one group in the note array, and one line in the beat array.

```
#include "notes.h"

int noteArray[][9] =
{
    {G4, 0, 0, 0, 0, 0, 0, 0, 1},

    {G4, 0, 0, 0, 0, 0, 0, 0, 1},
    {G4, 0, 0, 0, 0, 0, 0, 0, 1},
    {G4, 0, 0, 0, 0, 0, 0, 0, 1},
    {F4, 0, 0, 0, 0, 0, 0, 0, 1},
    {E4, 0, 0, 0, 0, 0, 0, 0, 1},
    {G4, 0, 0, 0, 0, 0, 0, 0, 1},
    {C5, 0, 0, 0, 0, 0, 0, 0, 1},
    {D5, 0, 0, 0, 0, 0, 0, 0, 1},

    {E5, 0, 0, 0, 0, 0, 0, 0, 1},
    {E5, 0, 0, 0, 0, 0, 0, 0, 1},
    {E5, 0, 0, 0, 0, 0, 0, 0, 1},
    {D5, 0, 0, 0, 0, 0, 0, 0, 1},
    {C5, 0, 0, 0, 0, 0, 0, 0, 1},
    {C5, 0, 0, 0, 0, 0, 0, 0, 1},
    {B4, 0, 0, 0, 0, 0, 0, 0, 1},

    {A4, 0, 0, 0, 0, 0, 0, 0, 1},
    {A4, 0, 0, 0, 0, 0, 0, 0, 1},
    {A4, 0, 0, 0, 0, 0, 0, 0, 1},
    {B4, 0, 0, 0, 0, 0, 0, 0, 1},
    {C5, 0, 0, 0, 0, 0, 0, 0, 1},
    {B4, 0, 0, 0, 0, 0, 0, 0, 1},
    {C5, 0, 0, 0, 0, 0, 0, 0, 1},
    {A4, 0, 0, 0, 0, 0, 0, 0, 1},
```

```
double beatArray[] =
{
    0.250,
    0.188, 0.063, 0.188, 0.063, 0.125, 0.125, 0.125, 0.125,
    0.063, 0.188, 0.188, 0.063, 0.250, 0.125, 0.125,
    0.188, 0.063, 0.188, 0.063, 0.188, 0.063, 0.188, 0.063,
```

***Figure 3.9-6:*** *Note and beat arrays of the song "Battle Hymn of the Republic" generated by the OMR system*

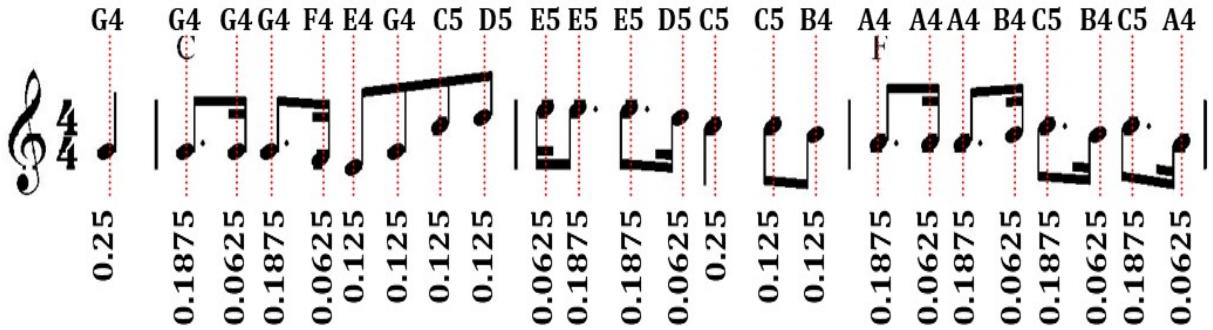



*Figure 3.9-7: Actual pitch and beats of the first section of "Battle Hymn of the Republic"*

### 3.9.7.2 Grand staff

Transcription of grand staves is a little more complex compared to its solo counterpart. While the pitch assignment is based on the staff lines and spaces, the beat assignment is based on an invisible vertical line running from left to right, denoted by broken lines in Figure 3.9-8.

Looking at the first aligned notes, in the bottom is a half note chord and at the top, a quarter note. If the aligned notes are of different types, the beat that will be assigned at that line will be equal to the beat from the lesser note, which in this case, 0.25. This process continues until the beat in each line is equal throughout, as shown in Table 3.9-1, denoted in red. This also works if the note with longer duration is on the top staff, or if rests are involved instead. Also, in terms of the *notebreak* parameter, if the aligned symbols are of the same type at a given time, it is set to 1, otherwise, it is equal to 0.

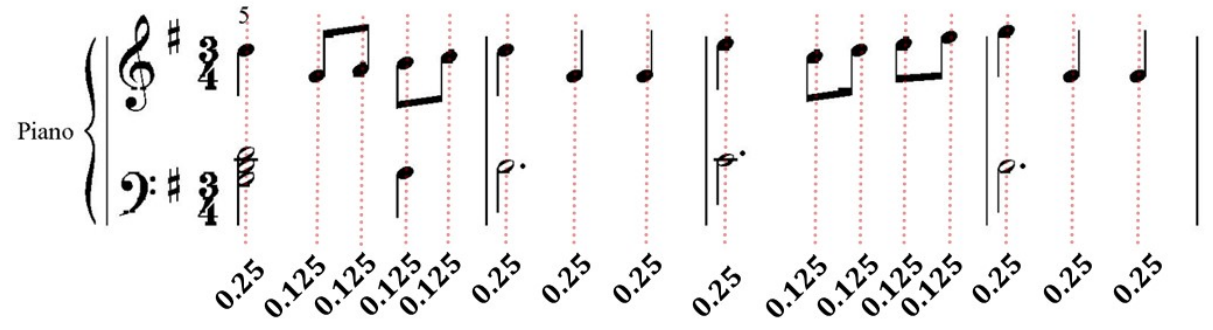



*Figure 3.9-8: Beats of the first section of "Minuet in G"*

| Line Number | 1 | 2 | 3 |
|---|---|---|---|
| Top staff beat | 0.25 | 0.125 | 0.125 |
| Bottom staff beat | 0.5 - 0.25 | 0.25 - 0.125 | 0.125 - 0.125 |
| Note break value | 0 | 0 | 1 |

***Table 3.9-1:*** *Tabular explanation for beat assignment; if the beat just before reduction (red) is equal to the rest of the aligned beats, balancing is complete*

```
#include "notes.h"

int noteArray[][9] =
{
    {D4, B3, G3, D5, 0, 0, 0, 0, 0},
    {D4, B3, G3, G4, 0, 0, 0, 0, 0},
    {D4, B3, G3, A4, 0, 0, 0, 0, 1},
    {A3, B4, 0, 0, 0, 0, 0, 0, 0},
    {A3, C5, 0, 0, 0, 0, 0, 0, 1},

    {B3, D5, 0, 0, 0, 0, 0, 0, 0},
    {B3, G4, 0, 0, 0, 0, 0, 0, 0},
    {B3, G4, 0, 0, 0, 0, 0, 0, 1},

    {C4, E5, 0, 0, 0, 0, 0, 0, 0},
    {C4, C5, 0, 0, 0, 0, 0, 0, 0},
    {C4, D5, 0, 0, 0, 0, 0, 0, 0},
    {C4, E5, 0, 0, 0, 0, 0, 0, 0},
    {C4, Fs5, 0, 0, 0, 0, 0, 0, 1},

    {B3, G5, 0, 0, 0, 0, 0, 0, 0},
    {B3, G4, 0, 0, 0, 0, 0, 0, 0},
    {B3, G4, 0, 0, 0, 0, 0, 0, 1},
```

```
double beatArray[] =
{
    0.250, 0.125, 0.125, 0.125, 0.125,
    0.250, 0.250, 0.250,
    0.250, 0.125, 0.125, 0.125, 0.125,
    0.250, 0.250, 0.250,
```

***Figure 3.9-9:*** *Pitch and beat arrays generated by the OMR system for the tune "Minuet in G"*

# 4. Synthesizer Adaptation

This chapter defines the adaptation of the OMR system's other half, the synthesizer subsystem. The inspiration behind the design and the steps involved to produce tones and player controls will be discussed.

## 4.1 Overview

Invented by Laurens Hammond and John M. Hanert, the Hammond organ is an electric organ that mimics the *timbre* of a pipe organ through electromechanical sound generation. Rotating tone wheels near an electronic pickup produces a signal like a sine wave. *Drawbars,* as shown in Figure 4.1-1, determine how a single key press will sound using a fundamental (third drawbar from left) and overtones, hence, the sound synthesis is additive.

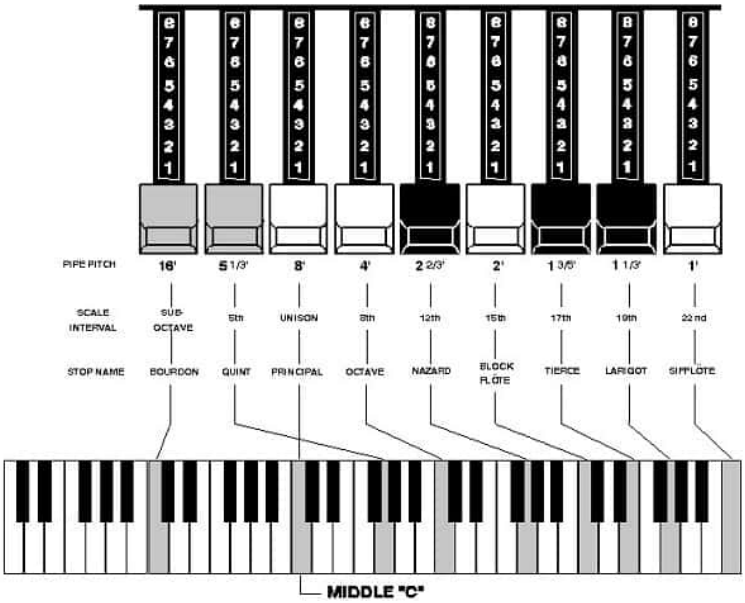



***Figure 4.1-1:*** *Drawbars of a Hammond organ*

## 4.2 Harmonizer

In its simplest definition, a harmonizer is a device that combines notes to produce *harmony*, one of the fundamental music elements. In that context, an organ is considered a harmonizer. Therefore, the proposed synthesizer design aims to mimic the tone generation of Hammond organs

using an additive digital direct synthesis approach. In this implementation however, due to its inherent odd harmonics and bright sound, the triangle wave is used as base instead of the typical sine wave..

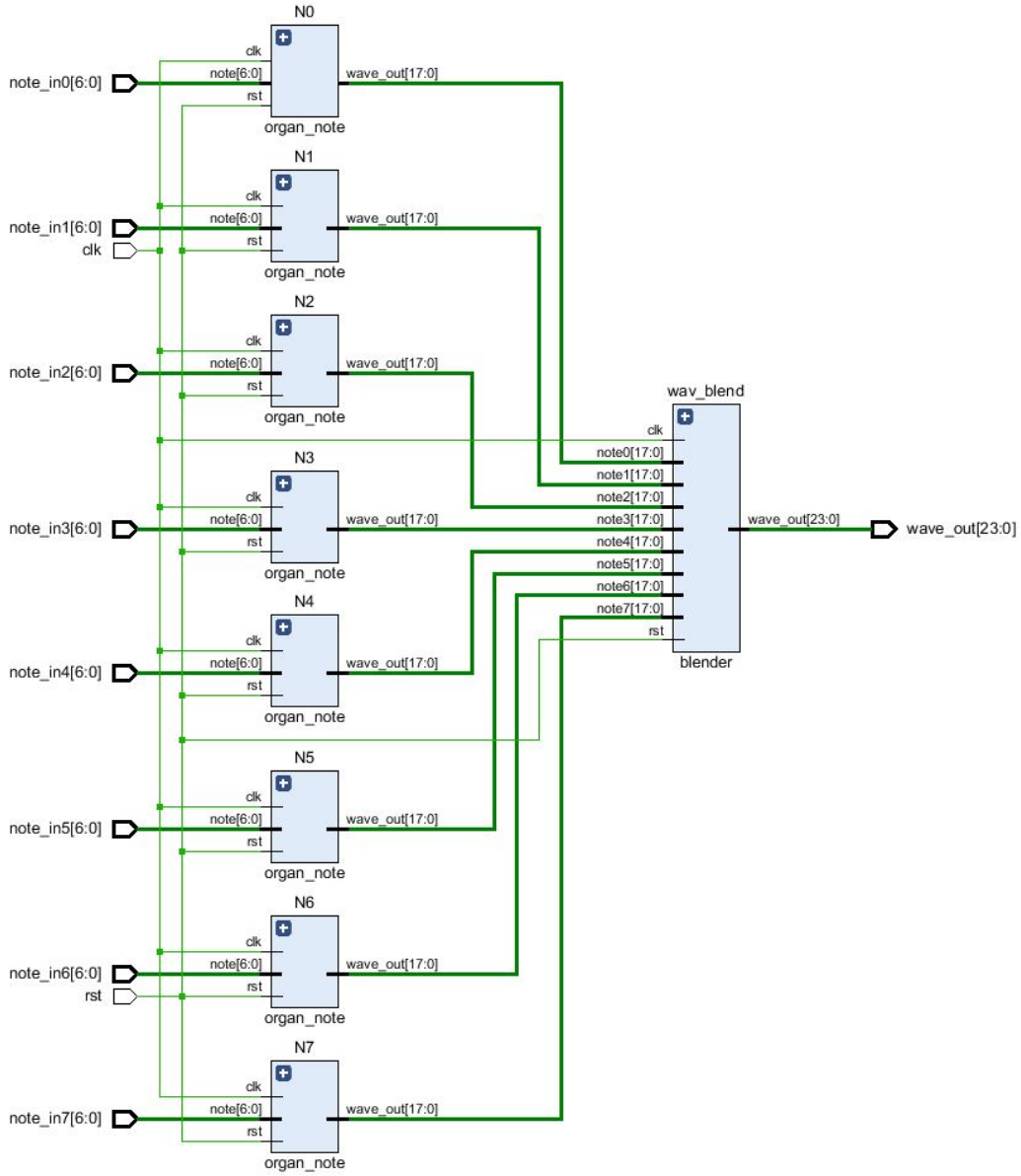



***Figure 4.2-1:*** *Top-level view of the proposed harmonizer*

### 4.2.1 Frequency control

The *frequency control word,* or FCW in a digital design synthesizer is a tuning word that divides the scaled system clock into the desired output frequency.

$$f_{out} = \frac{(FCW)(f_{ref})}{2^M} \tag{4.1}$$

As shown in Equation 4.1, the output frequency is a function of the reference clock, the FCW, and $2^M$, where $M$ is the width of the phase accumulator. From this, the equation for the FCW can be derived:

$$FCW = \frac{(f_{out})(2^M)}{f_{ref}} \tag{4.2}$$

For instance, the output frequency of the pitch A4 is 440 Hz, the phase accumulator used in the harmonizer is 24 bits, and the reference frequency is the audio sampling frequency of 48 kHz, therefore, the FCW of the pitch is:

$$FCW = \frac{(440)(2^{24})}{48000} \approx \mathbf{153791}$$

### 4.2.2 Numerically-controlled oscillator

The numerically-controlled oscillator, or NCO, is the workhorse of a DDS. Based on the input FCW and the system clock, the phase accumulator produces a sawtooth-like waveform. The truncated value is sent to a phase-to-amplitude converter, which in turn, outputs an amplitude value to be sent to a digital-to-analog converter. In the synthesizer implementation, the PAC is represented by a $2^{12}\ x\ 16$ triangle waveform lookup-table, and the DAC is contained inside an audio codec.

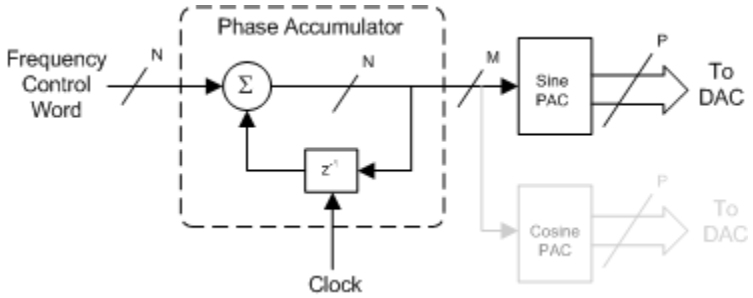



***Figure 4.2-2:*** *Generic block diagram of an NCO [14]*

### 4.2.3 Individual note generator

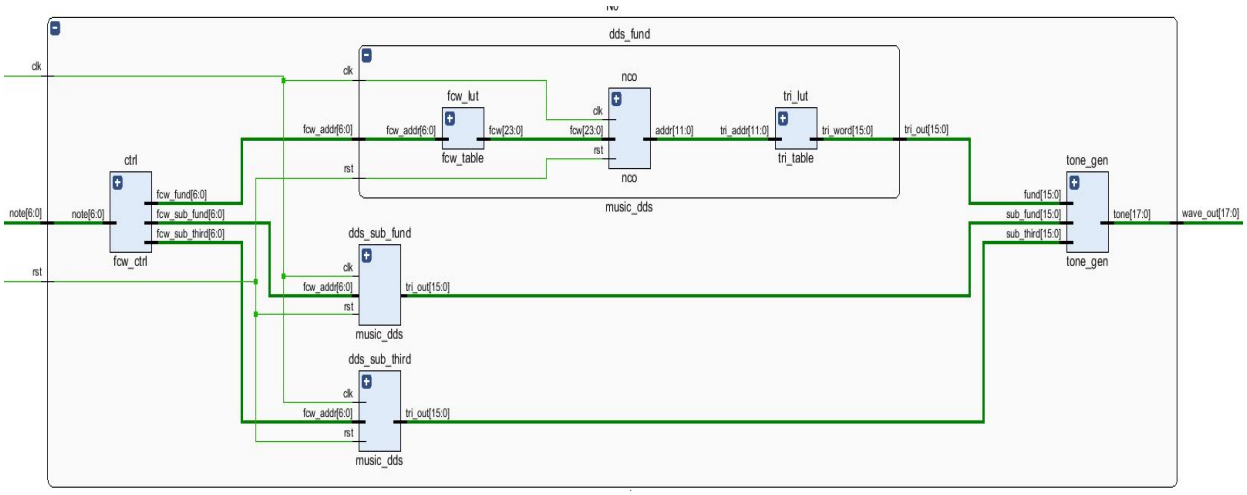



***Figure 4.2-3:*** *Top-level view of the individual note generator*

To simulate the effect of drawbars in a Hammond organ while keeping the hardware utilization to nominal levels, the three leftmost drawbars from Figure 4.1-1 are used, representing sub-fundamental, sub-third, and fundamental frequencies, respectively. The tone generator consists of three DDS instantiations and two controllers for both frequency control word and tone.

In terms of pitch, the *sub-fundamental* of a note is a pitch 12 semitones below the pitch in question, while the *sub-third*, or perfect fifth, is seven semitones above it. The FCW controller takes the one input note, and in turn, outputs three FCW equivalents based on this concept. To avoid unnecessary noise during playback, if the pitch is lower than A1, the sub-harmonic is set to 0, and if the pitch is higher than F7, the sub-fifth is set to 0.

| Input note | Sub-fundamental | Sub-third | Fundamental |
|---|---|---|---|
| A4 | A3 | E5 | A4 |
| F#7 | F#6 | 0 | F#7 |
| G1 | 0 | D2 | G1 |

***Table 4.2-1:*** *List of sample input notes and its under/over tones*

On the other side of the generator is the tone control. To ensure the quality of the output the individual DDS outputs are equalized before blending. That is, the fundamental pitch is shifted 1 bit to the left, while both sub-fundamental and sub-third are shifted 4 bits to the right. As a result,

the blended output will sound just like the fundamental note, but with the faint presence of undertones and overtones. Figures 4.2-4 and 4.2-5 provide a visual representation of the note generator process with input pitch C4. The figures show that the blended waveforms are almost identical to the fundamental waveform.

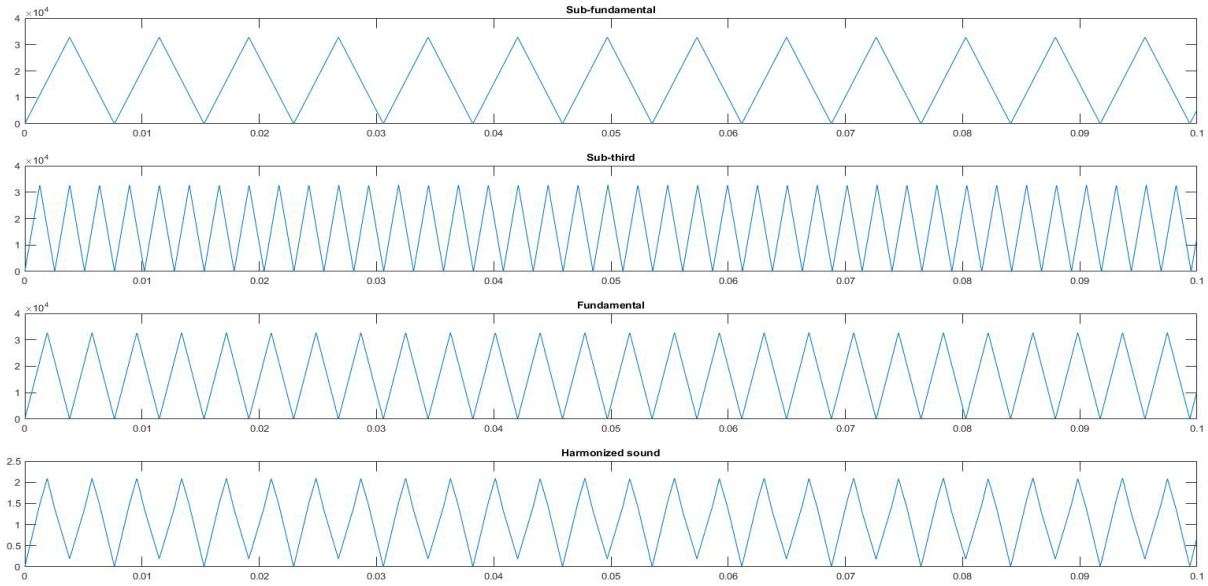



***Figure 4.2-4:*** *MATLAB simulation of the note generator for the input pitch C4*

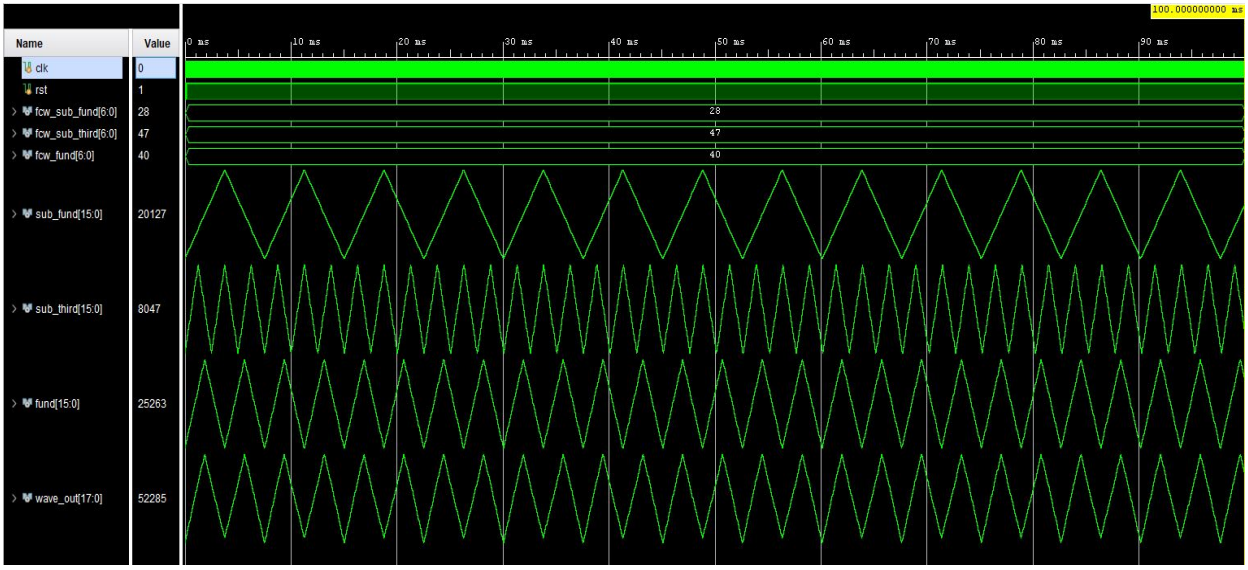



***Figure 4.2-5:*** *Vivado equivalent of the MATLAB simulation for the pitch C4*

### 4.2.4 Harmonized tone generator

Figure 4.2-1 shows the block diagram of the harmonizer. It consists of eight instantiations of the individual note generator and a blender module. The blender module calculates the sum of all eight notes, then it is shifted 3 bits to the left to increase overall volume. Figures 4.2-7 and 4.2-8 demonstrates the simulations for the C major chord.

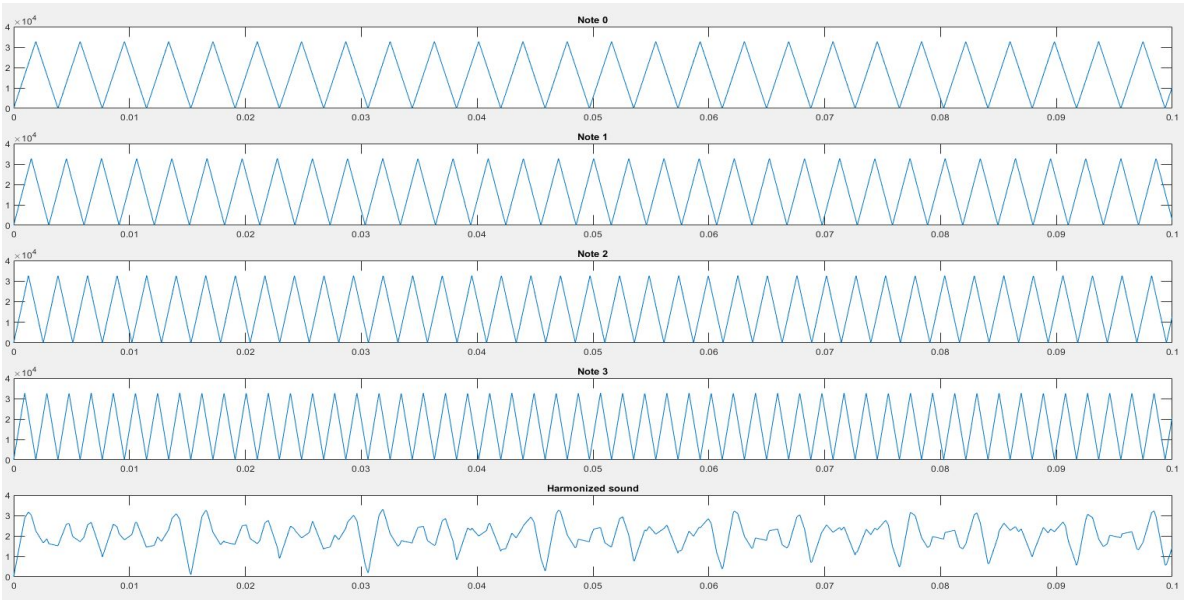



***Figure 4.2-6**: MATLAB simulation of the C major chord and its individual notes*

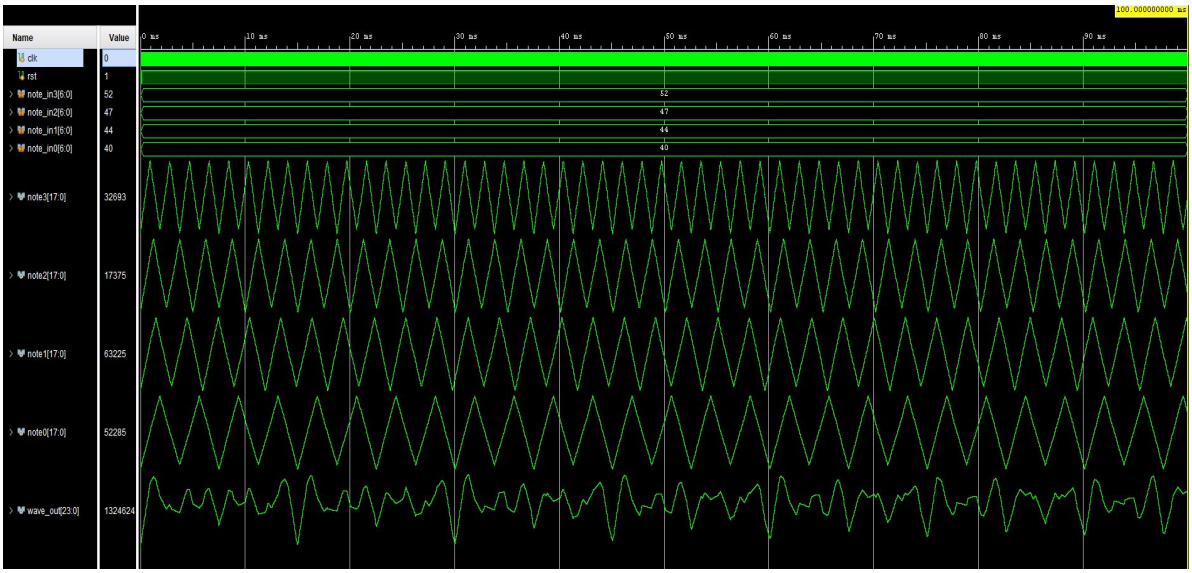



***Figure 4.2-7:** Vivado simulation of the C major chord and its individual notes*

## 4.3 Audio codec configuration

The authors of the Zynq tutorial book [7] provides a controller IP for the SSM2603 audio codec by Analog Devices. [1] Onboard the Zybo Z7 development board, it is configured through and $I^2C$ bus, while audio data is transferred through the $I^2S$ protocol. To produce the desired sound, utilizing a timer, the harmonizer output is sent to the left and right channels of the codec. In addition, to ensure proper functionality, a second fabric clock with frequency 12.288 MHz is added to the overall design to allow a 48 kHz sampling rate.

## 4.4 Overall design

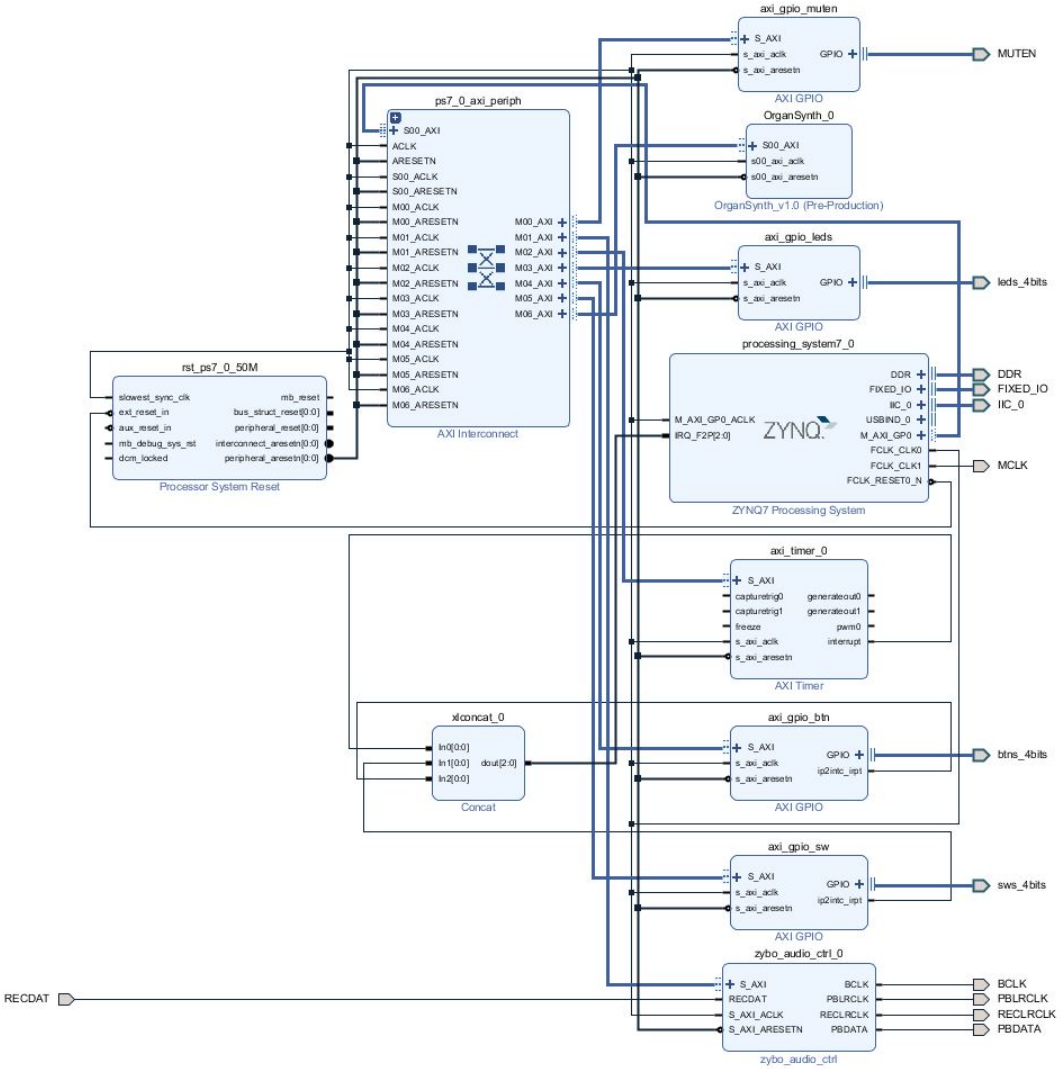



***Figure 4.4-1:*** *Complete block diagram of the synthesizer subsystem*

# 5. Implementation

## 5.1 Platform and tools

Four primary software platforms were utilized during the development of the project. MATLAB provided the necessary functions and toolboxes for the OMR user interface. The Vivado Design Suite allowed for the creation of the harmonizer module, and the control mechanisms for the synthesizer was made possible by Xilinx SDK. Finally, Github is used for version control and revision storage of all source codes.

## 5.2 Project setup

Figure 5.2-1 shows the physical setup for the replayable OMR system. In the middle of the screen is the HOMeR user interface which performs the music recognition, while the background application is the SDK window that allows for the control of the synthesizer board. Audio can be heard through a speaker connected to the board using a 3.5mm auxiliary cord.

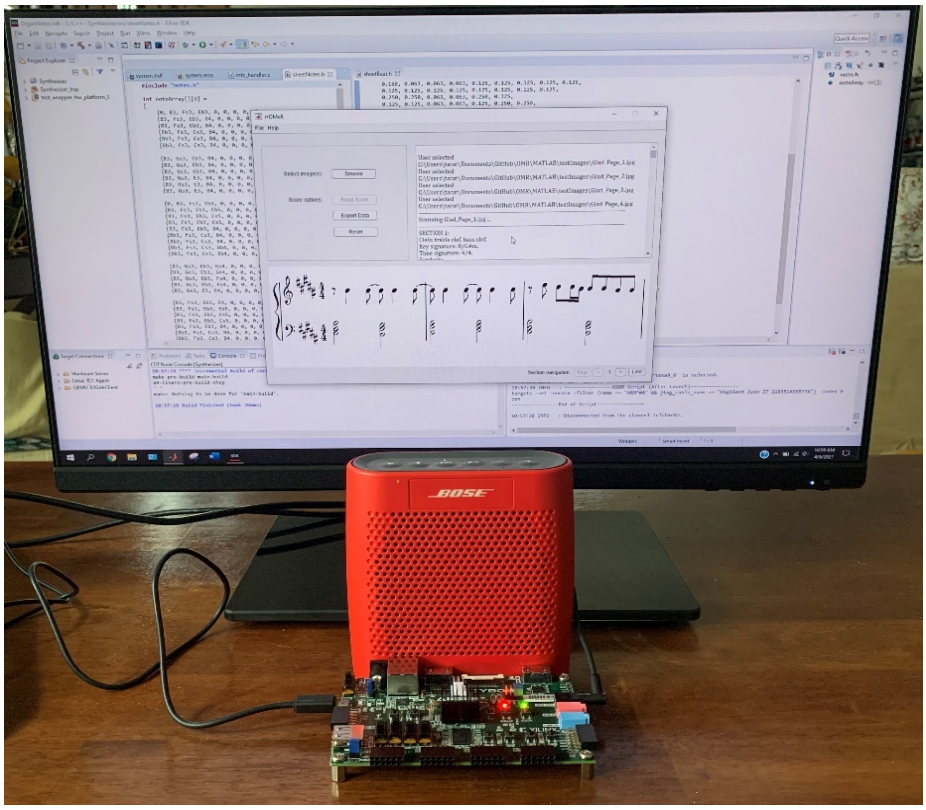



***Figure 5.2-1:*** *Project setup with the GUI and synthesizer board*

## 5.3 HOMeR

To allow for a seamless user experience, the OMR system is wrapped in a graphical user interface, *HOMeR*, an acronym for "Hardware Optical Music Recognizer". *HOMeR* can also be regarded as a play on the word *hammer*, the sound-producing part of a piano.

### User Interface

Figure 5.3-1 shows the HOMeR user interface. It consists of three primary sections: control (top left), status window (top right), and section navigation (bottom). The presence of a UI allows the user to go through the entire music recognition process with only a few clicks.

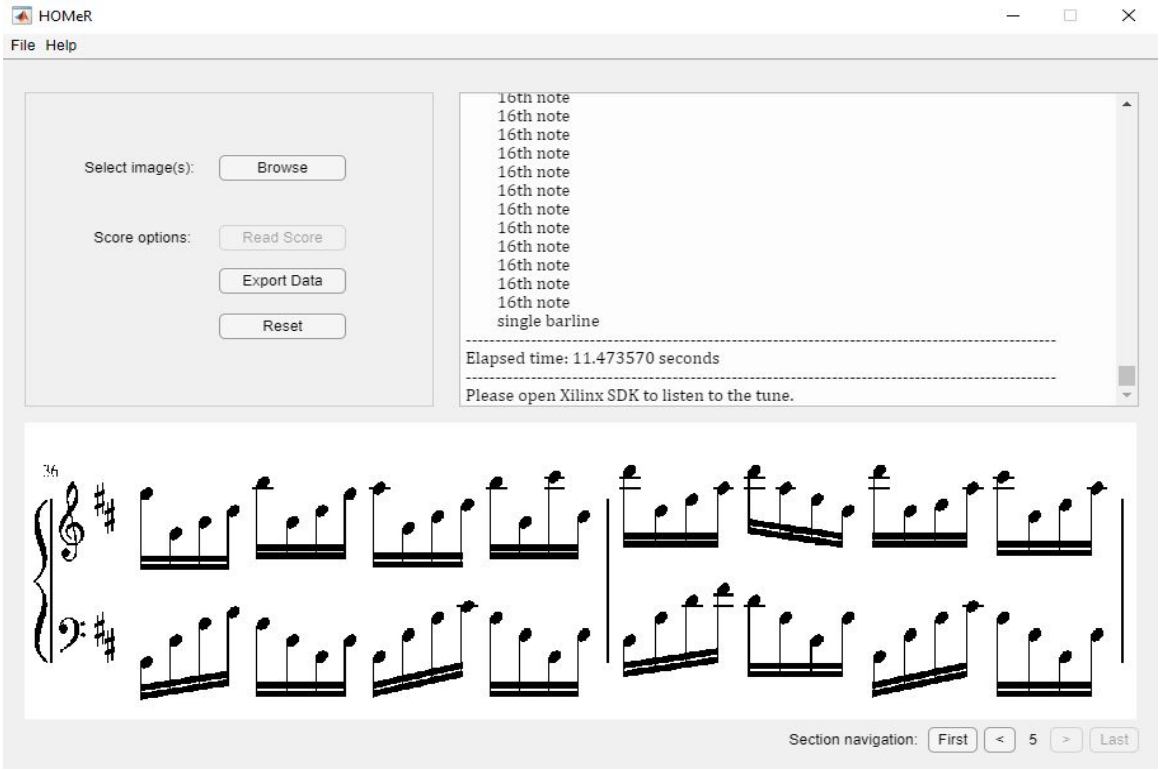



***Figure 5.3-1:*** *HOMeR user interface*

### Features

HOMeR enables the user to process single or a batch of images. Multipage scores are also allowed to be processed. Once processed, the user can navigate through both the section and status

windows to check all the symbols detected by the system. Finally, the detection results generated by the software can be saved using the 'Export Data' button.

## 5.4 Hardware

The system was tested using a laptop with an 11th Generation Intel processor. While not included in the primary objectives, the run times were recorded during the performed tests to determine the speed of the music recognition process.

### Development board

The Zybo Z7-20 development board is the hardware platform used for the project. Through the Xilinx All Programmable System on Chip (AP SOC), a dual-core ARM Cortex-A9 processor is integrated with the Zynq-7020 field programmable field array (FPGA) logic. The harmonizer module and the other peripherals such as buttons and switches are located on the board's *programming logic* side, while being controlled on its *processing system*.

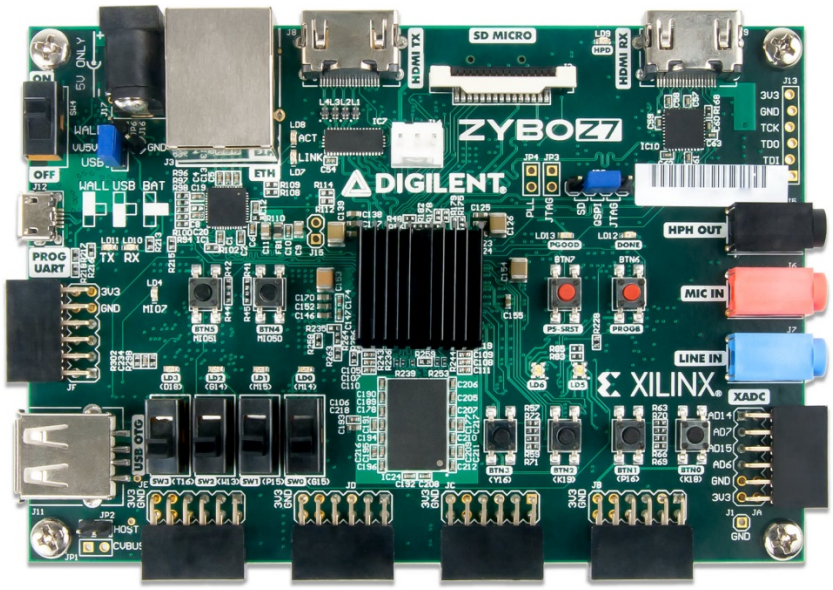



***Figure 5.4-1:*** *Zybo Z7-20 board used as the synthesizer*

### Control mechanism

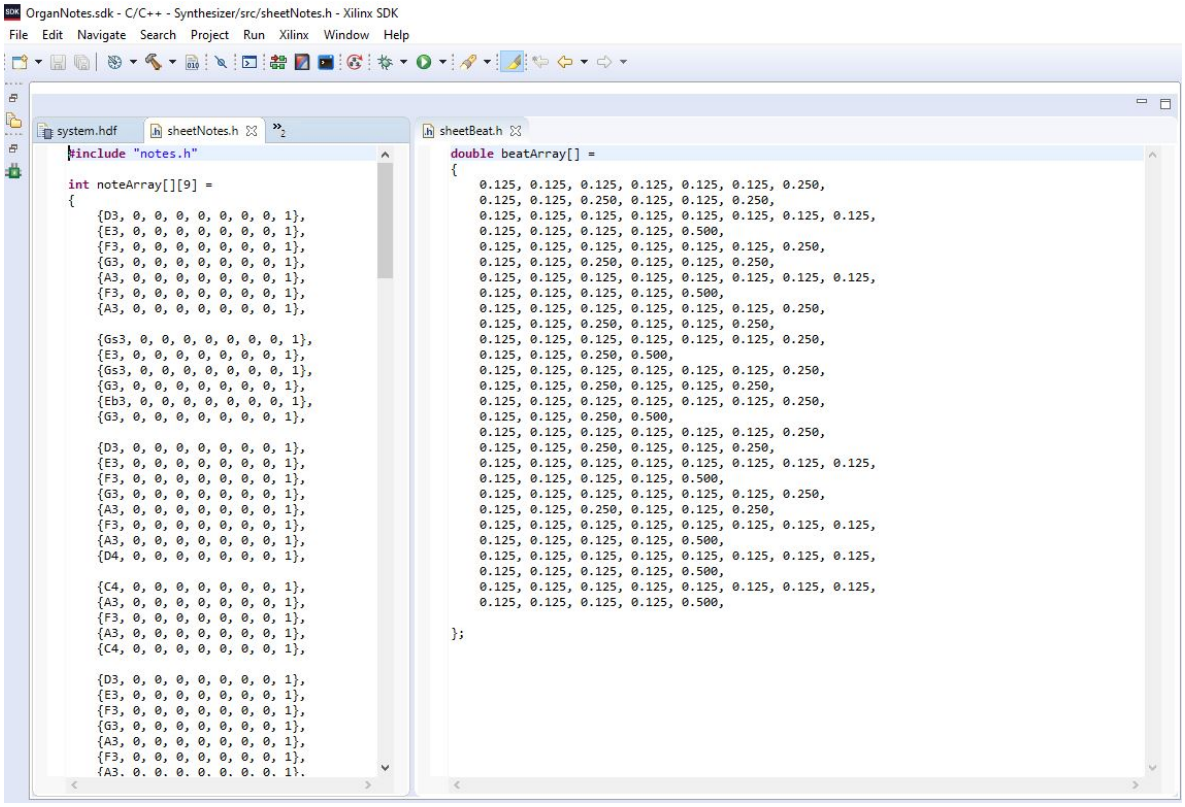



***Figure 5.4-2:*** *SDK window after the sheet music processing*

Once the HOMeR user interface finishes its transcription, the note and beat arrays will be loaded into the Xilinx SDK application, as shown in Figure 5.4-2. After flashing the synthesizer system into the Zybo board, the audible version of the sheet music is ready to be heard.

The two main controls in the synthesizer are *playback* and *tempo.* The four buttons onboard represent the following functions (in order): restart, pause, play, and fast forward. Similarly, the four switches can vary the playback speed in the following multiplier: 0.5x, 0.7x, 1.25x, and 1.5x, respectively. Since the current implementation lacked volume control, for best experience, use a speaker with adjustable volume.

# 6. Testing and Evaluation

## 6.1 Initial detection results

The tests performed in this section are pre-correction tests, that is, the results came from only the symbol detection part of the OMR. In this case, 28 out of 40 images yielded a 100% correct detection rate in all notes, rests, and non-note symbols. In the 12 images that did not, the misidentifications came from three sources: dots, ties/slurs, and fermatas. Overall, the average initial detection rate for all 40 input images is 97.24%.

| Title | Detected | Actual | Detection Rate |
|---|---|---|---|
| Hallelujah_Page_1.jpg | 25 | 5 | -300.00% |
| saints.jpg | 1 | 1 | 100.00% |
| Glad_Page_2.jpg | 2 | 2 | 100.00% |
| Glad_Page_1.jpg | 3 | 3 | 100.00% |
| Glad_Page_3.jpg | 2 | 2 | 100.00% |
| Glad_Page_4.jpg | 3 | 3 | 100.00% |
| Hallelujah_Page_2.jpg | 26 | 12 | -16.67% |
| auldPiano.jpg | 17 | 17 | 100.00% |
| sweetheart.jpg | 30 | 20 | 50.00% |
| auld.jpg | 16 | 15 | 93.33% |
| yankee.jpg | 4 | 3 | 66.67% |
| patrol.jpg | 7 | 6 | 83.33% |

***Table 6.1-1:*** *Dot detection rates for the imperfect sheets*

| Title | Detected | Actual | Detection Rate |
|---|---|---|---|
| Hallelujah_Page_1.jpg | 1 | 1 | 100.00% |
| saints.jpg | 1 | 3 | 33.33% |
| Glad_Page_2.jpg | 5 | 8 | 62.50% |
| Glad_Page_1.jpg | 6 | 9 | 66.67% |
| Glad_Page_3.jpg | 13 | 17 | 76.47% |
| Glad_Page_4.jpg | 8 | 10 | 80.00% |
| Hallelujah_Page_2.jpg | 4 | 4 | 100.00% |
| auldPiano.jpg | 0 | 0 | 100.00% |
| sweetheart.jpg | 10 | 10 | 100.00% |
| auld.jpg | 2 | 3 | 66.67% |
| yankee.jpg | 1 | 1 | 100.00% |
| patrol.jpg | 0 | 0 | 100.00% |

***Table 6.1-2:*** *Tie/slur detection rates for the imperfect sheets*

| Title | Detected | Actual | Detection Rate |
|---|---|---|---|
| Hallelujah_Page_1.jpg | 0 | 0 | 100.00% |
| saints.jpg | 1 | 0 | 0.00% |
| Glad_Page_2.jpg | 3 | 0 | 0.00% |
| Glad_Page_1.jpg | 3 | 0 | 0.00% |
| Glad_Page_3.jpg | 4 | 0 | 0.00% |
| Glad_Page_4.jpg | 2 | 0 | 0.00% |
| Hallelujah_Page_2.jpg | 0 | 0 | 100.00% |
| auldPiano.jpg | 0 | 0 | 100.00% |
| sweetheart.jpg | 0 | 0 | 100.00% |
| auld.jpg | 0 | 0 | 100.00% |
| yankee.jpg | 0 | 0 | 100.00% |
| patrol.jpg | 0 | 0 | 100.00% |

***Table 6.1-3:*** *Fermata detection rates for the imperfect sheets*

## 6.2 Final detection results

After applying musical semantics to the input images, unnecessary symbols that were initially detected were eliminated. In this case, the misidentifications of the dots and fermatas were fixed, only six of the twelve sheets remained imperfect, however, the total detection rate in these images increased greatly. Overall, the final average detection rate for all 40 input images increased from 97.24% to 99.80%, with 6470 out of 6485 total symbols correctly detected.

| Title | Detection Rates | |
|---|---|---|
| | Pre-correction | Post-correction |
| saints.jpg | 87.18% | 98.10% |
| Glad_Page_3.jpg | 90.50% | 98.17% |
| Glad_Page_1.jpg | 89.74% | 98.71% |
| Glad_Page_2.jpg | 89.42% | 98.93% |
| auld.jpg | 96.92% | 98.94% |
| Glad_Page_4.jpg | 90.77% | 99.07% |

***Table 6.2-1:*** *Detection rate improvements on the imperfect sheets*

## 6.3 Detection speeds

Five runs were performed on each sheet music image. The fastest run, “Mary Had a Little Lamb”, lasted for 0.53 seconds, while the slowest run lasted for 5.41 seconds, for the first page of “Hallelujah”. Symbol density played a vital role in the detection speed, as the Hallelujah image consisted of at least 20 excess symbols detected, as shown in Table 6.1-1.

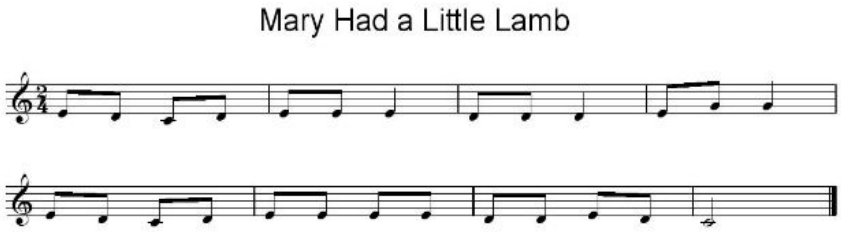



***Figure 6.3-1:*** *Mary Had a Little Lamb score, one of the input images with least symbol density*

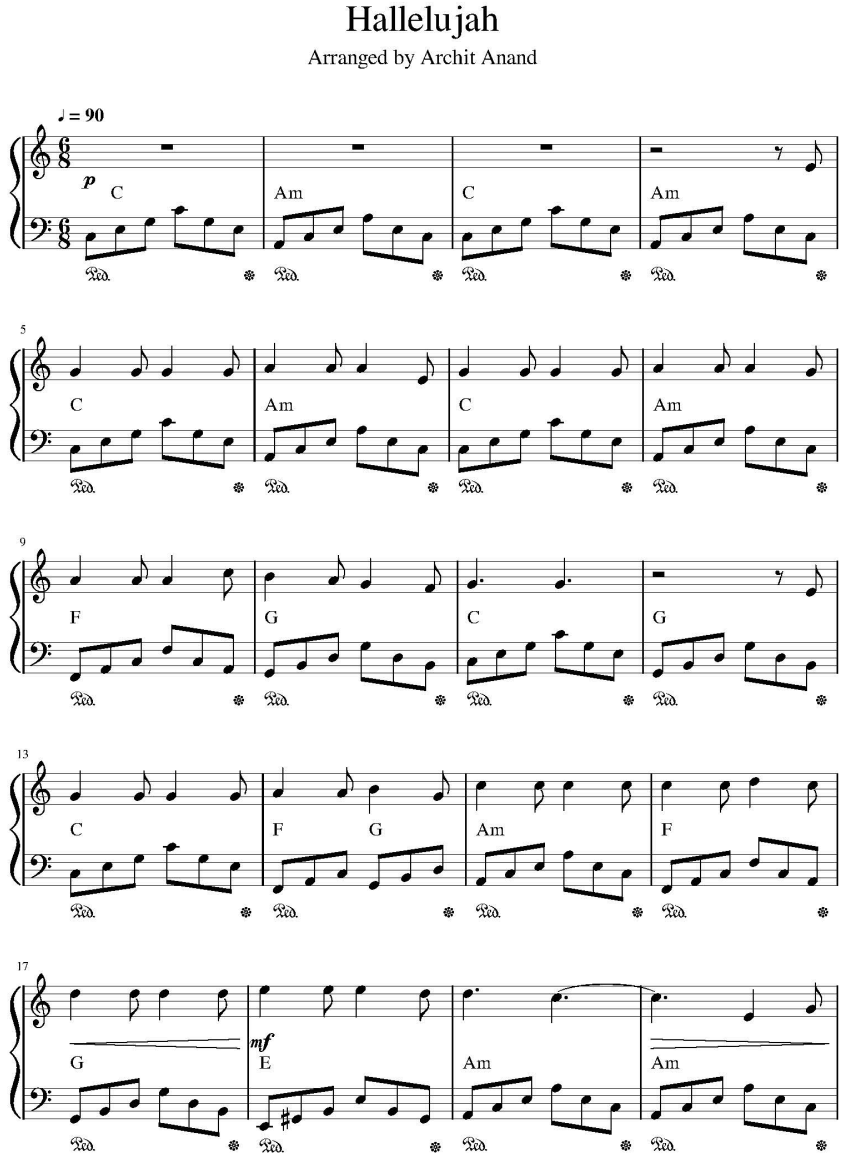



***Figure 6.3-2:*** *Page 1 of the Hallelujah score, where symbol density is much higher*

## 6.4 Note and beat assignment

Of all the 40 sheet music images, only two incorrect assignments were detected, one for pitch and one for beat. In the case of O Sole Mio in Figure 6-4.1, due to a previous accidental, the pitch of the half note below the green arrow is Eb3. Since there is a tie on top of it, the following note denoted by the red note must have the same pitch, regardless if it is in a new section. However, after the assignment, the pitch changed to E3, thus, the accidental did not propagate in the next section.

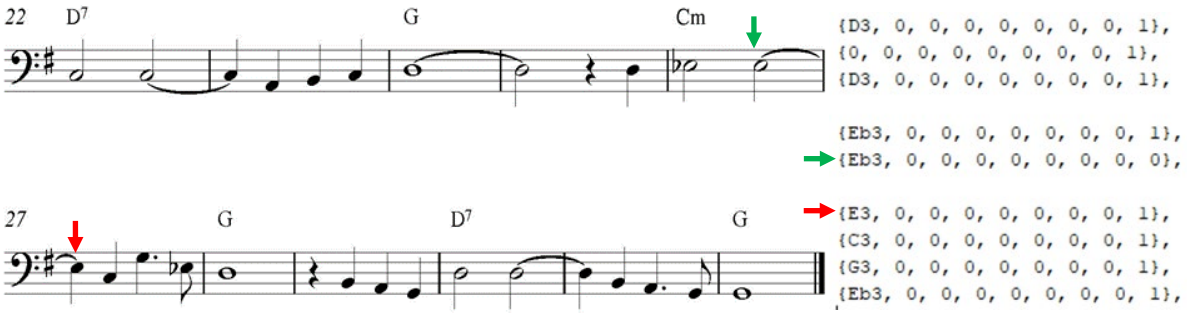



*Figure 6.4-1: Incorrect pitch assignment on the note indicated by red arrows*

The other mistake came from an incorrectly detected staccato. Coming from a non-music symbol after the staff removal part, the quarter note was played as an eighth note, followed by silence.

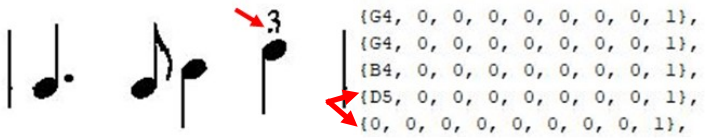



*Figure 6.4-2: Incorrect articulation indicated by red arrows*

# 7. Conclusion and Future Work

The proposed project's primary objective is to provide a preliminary framework towards an eventual hardware implementation of a replayable OMR system. After testing at least forty sheet music images of different types and dimensions, the HOMeR subsystem detected and classified musical symbols with high accuracy. Also, HOMeR was able to successfully interface with the Zybo hardware synthesizer, meeting the co-design requirement.

Because of the attempts to develop novel methods in the detection processes, near-perfect images were used as inputs as the baseline test. That said, potential additions to the OMR preprocessing stage include deskewing, rotation, and other binarization algorithms for the system to handle input scores with lesser quality.

In terms of sound quality on the other hand, during the sound tests, popping sounds can be heard during playback of some of the tunes. Upon reading through the Hammond organ's background, there have been a history of popping or clicking sounds during a single key press. While there is no correlation found between these two events, the coincidence was deemed interesting.

Finally, the potential next steps towards a complete hardware OMR are the revisions of the entire OMR software system in terms of coding for hardware compatibility. While there have been previous attempts on this during the project development, they were unsuccessful. However, as technology advances on an unprecedented manner, this goal is possible.

## References


[1] Analog Devices, "Low Power Audio Codec," SSM2603 datasheet, Feb. 2008 (Revised Jul. 2018)

[2] All About Direct Digital Synthesis | Analog Devices. (n.d.). Analog Devices. https://www.analog.com/en/analog-dialogue/articles/all-about-direct-digital-synthesis.html#

[3] Bainbridge, D., & Bell, T. (2001). The Challenge of Optical Music Recognition. Computers and the Humanities, 35(2), 95–121. https://doi.org/10.1023/a:1002485918032

[4] Byrd, D., & Simonsen, J. G. (2015). Towards a Standard Testbed for Optical Music Recognition: Definitions, Metrics, and Page Images. Journal of New Music Research, 44(3), 169–195. https://doi.org/10.1080/09298215.2015.1045424

[5] Calvo-Zaragoza, J., Barbancho, I., Tardón, L. J., & Barbancho, A. M. (2014). Avoiding staff removal stage in optical music recognition: application to scores written in white mensural notation. Pattern Analysis and Applications, 18(4), 933–943. https://doi.org/10.1007/s10044-014-0415-5

[6] Calvo-Zaragoza, J., Jr., J. H., & Pacha, A. (2020). Understanding Optical Music Recognition. ACM Computing Surveys, 53(4), 1–35. https://doi.org/10.1145/3397499

[7] Crockett, L. H., Elliot, R. A., & Enderwitz, M. A. (2015). The Zynq Book Tutorials for Zybo and ZedBoard. Strathclyde Academic Media.

[8] Fujinaga, I. Staff detection and removal. In Susan George, editor, Visual Perception of Music Notation: On-Line and Off-Line Recognition, pages 1–39. Idea Group Inc., 2004.

[9] Hanning, B. R. (2020). Concise History of Western Music (Fifth Edition, Anthology Update ed.). W. W. Norton & Company.

[10] N. Otsu. A threshold selection method from gray-level histograms. IEEE Transactions on Systems, Man and Cybernetics, 9(1):62–66, 1979.

[11] PhD, B. M. E. (2013). History of Music in Western Culture (4th Edition) (4th ed.). Pearson.

[12] Rebelo, Ana. “Robust optical recognition of handwritten musical scores based on domain knowledge.” (2012).

[13] Rebelo, A., Fujinaga, I., Paszkiewicz, F., Marcal, A. R. S., Guedes, C., & Cardoso, J. S. (2012). Optical music recognition: state-of-the-art and open issues. International Journal of Multimedia Information Retrieval, 1(3), 173–190. https://doi.org/10.1007/s13735-012-0004-6

[14] Wikipedia contributors. (2021a, February 16). Numerically-controlled oscillator. Wikipedia. https://en.wikipedia.org/wiki/Numerically-controlled_oscillator

[15] Wikipedia contributors. (2021b, April 9). List of musical symbols. Wikipedia. https://en.wikipedia.org/wiki/List_of_musical_symbols

## Appendix A: Dataset and Detection Results

| No. | Title | Composer | Type | Pages |
|---|---|---|---|---|
| 1 | American Patrol | F. W. Meacham | Solo | 1 |
| 2 | Auld Lang Syne | Robert Burns | Solo | 1 |
| 3 | Battle Hymn of the Republic | Julia Ward Howe | Solo | 1 |
| 4 | Camptown Races | Stephen Foster | Solo | 1 |
| 5 | For He's a Jolly Good Fellow | Traditional | Solo | 1 |
| 6 | Frere Jacques | Unknown | Solo | 1 |
| 7 | In the Hall of the Mountain King | Edvard Grieg | Solo | 1 |
| 8 | Let Me Call You Sweetheart | Leo Friedman | Solo | 1 |
| 9 | Mary Had a Little Lamb | Sarah Josepha Hale | Solo | 1 |
| 10 | My Bonnie | Scottish folk song | Solo | 1 |
| 11 | O Sole Mio | Eduardo di Capua | Solo | 1 |
| 12 | Silent Night | Franz Xaver Gruber | Solo | 1 |
| 13 | Twinkle, Twinkle, Little Star | English lullaby | Solo | 1 |
| 14 | Waltzing Matilda | Christina Macpherson | Solo | 1 |
| 15 | Yankee Doodle | George M. Cohan | Solo | 1 |
| 16 | You Are My Sunshine | Jimmie Davis | Solo | 1 |
| 17 | Amazing Grace | John Newton | Piano | 2 |
| 18 | America the Beautiful | Samuel A. Ward | Piano | 1 |
| 19 | Auld Lang Syne | Robert Burns | Piano | 1 |
| 20 | Burleske | Leopold Mozart | Piano | 1 |
| 21 | Canon in D | Johann Pachelbel | Piano | 4 |
| 22 | Fur Elise | Ludwig van Beethoven | Piano | 1 |
| 23 | Glad You Exist | Dan Smyers, et al. | Piano | 4 |
| 24 | Greensleeves | English folk song | Piano | 1 |
| 25 | Hallelujah | Leonard Cohen | Piano | 2 |
| 26 | Minuet in G | Christian Petzold | Piano | 1 |
| 27 | Ode to Joy | Ludwig van Beethoven | Piano | 1 |
| 28 | Old Macdonald Had a Farm | Thomas D'Urfey | Piano | 1 |
| 29 | Turkish March | Wolfgang Amadeus Mozart | Piano | 2 |
| 30 | When Johnny Comes Marching Home | Patrick Gilmore | Piano | 1 |
| 31 | When the Saints Go Marching In | James M. Black | Piano | 1 |

***Table A-1**: Complete list of the input scores*

| Source Website | Score Number(s) |
|---|---|
| Capotasto Music | 1, 3, 5, 8, 10, 11, 14, 19, 24, 28, 30 |
| Christmas Music Songs | 12 |
| Meadowlark Violin | 2 |
| Michael Kravchuk | 4, 15 |
| MuseScore | 7, 13, 16, 17, 18, 20, 21, 22, 23, 25, 27, 29 |
| Music for Music Teachers | 26 |
| Unknown | 31 |
| Violin Sheet Music | 6, 9 |

***Table A-2:*** *List of sheet music source websites*

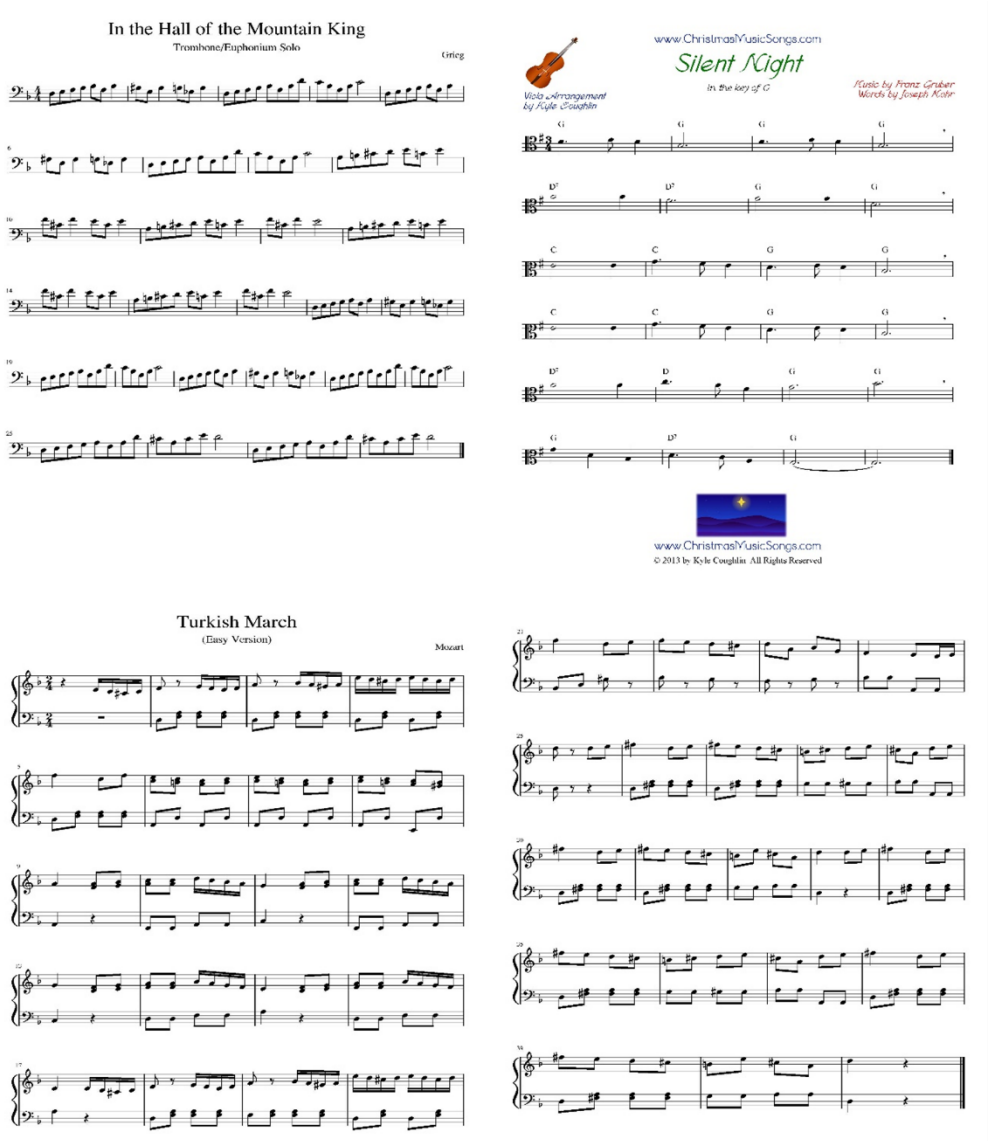



***Figure A-1:*** *Some of the input images used in the project*

| Title | Pre-correction rate | Post-correction rate |
|---|---|---|
| America2.jpg | 100.00% | 100.00% |
| auld.jpg | 96.92% | 98.94% |
| auldPiano.jpg | 92.31% | 100.00% |
| battle.jpg | 100.00% | 100.00% |
| bonnie.jpg | 100.00% | 100.00% |
| burleske.jpg | 100.00% | 100.00% |
| camptown.jpg | 100.00% | 100.00% |
| canon_d_Page_1.jpg | 100.00% | 100.00% |
| canon_d_Page_2.jpg | 100.00% | 100.00% |
| canon_d_Page_3.jpg | 100.00% | 100.00% |
| canon_d_Page_4.jpg | 100.00% | 100.00% |
| frere.jpg | 100.00% | 100.00% |
| furelise.jpg | 100.00% | 100.00% |
| Glad_Page_1.jpg | 89.74% | 98.71% |
| Glad_Page_2.jpg | 89.42% | 98.93% |
| Glad_Page_3.jpg | 90.50% | 98.17% |
| Glad_Page_4.jpg | 90.77% | 99.07% |
| Grace3_Page_1.jpg | 100.00% | 100.00% |
| Grace3_Page_2.jpg | 100.00% | 100.00% |
| Greensleeves.jpg | 100.00% | 100.00% |
| hall.jpg | 100.00% | 100.00% |
| Hallelujah_Page_1.jpg | 69.23% | 100.00% |
| Hallelujah_Page_2.jpg | 91.03% | 100.00% |
| johnny2.jpg | 100.00% | 100.00% |
| jolly2.jpg | 100.00% | 100.00% |
| mac.jpg | 100.00% | 100.00% |
| mary.jpg | 100.00% | 100.00% |
| matilda.jpg | 100.00% | 100.00% |
| minuet.jpg | 100.00% | 100.00% |
| odePiano.jpg | 100.00% | 100.00% |
| patrol.jpg | 98.72% | 100.00% |
| saints.jpg | 87.18% | 98.10% |
| silent.jpg | 100.00% | 100.00% |
| sole.jpg | 100.00% | 100.00% |
| sunshine.jpg | 100.00% | 100.00% |
| sweetheart.jpg | 96.15% | 100.00% |
| turk_Page_1.jpg | 100.00% | 100.00% |
| turk_Page_2.jpg | 100.00% | 100.00% |
| twinkle.jpg | 100.00% | 100.00% |
| yankee.jpg | 97.44% | 100.00% |
| Overall rate: | 97.24% | 99.80% |

***Table A-3:*** *Complete list of detection rates for all input images*

| Input image | Run 1 | Run 2 | Run 3 | Run 4 | Run 5 | Average |
|---|---|---|---|---|---|---|
| mary.jpg | 0.531907 | 0.528448 | 0.527252 | 0.538946 | 0.522531 | 0.529817 |
| frere.jpg | 0.616932 | 0.614073 | 0.614827 | 0.618363 | 0.610289 | 0.614897 |
| auld.jpg | 1.809298 | 1.803036 | 1.803914 | 1.800214 | 1.787964 | 1.800885 |
| twinkle.jpg | 1.824971 | 1.814949 | 1.827643 | 1.824815 | 1.829067 | 1.824289 |
| sunshine.jpg | 2.151170 | 2.153906 | 2.145202 | 2.151469 | 2.156075 | 2.151565 |
| battle.jpg | 2.241064 | 2.220265 | 2.248628 | 2.218379 | 2.236573 | 2.232982 |
| bonnie.jpg | 2.312345 | 2.303596 | 2.301758 | 2.705388 | 2.408103 | 2.406238 |
| Grace3_Page_2.jpg | 2.469291 | 2.591663 | 2.776092 | 2.754389 | 2.759001 | 2.670087 |
| camptown.jpg | 2.828821 | 2.854843 | 2.824367 | 2.857634 | 2.854117 | 2.843956 |
| yankee.jpg | 2.935620 | 2.898465 | 2.895721 | 2.896736 | 2.891560 | 2.903620 |
| Hallelujah_Page_2.jpg | 3.267130 | 3.291205 | 3.265008 | 3.260924 | 3.311701 | 3.279194 |
| jolly2.jpg | 3.328569 | 3.352300 | 3.340534 | 3.339370 | 3.348245 | 3.341804 |
| johnny2.jpg | 3.477297 | 3.512196 | 3.525991 | 3.507362 | 3.501521 | 3.504874 |
| sole.jpg | 3.628505 | 3.624549 | 3.609227 | 3.628878 | 3.647682 | 3.627768 |
| mac.jpg | 3.640948 | 3.684597 | 3.653984 | 3.635554 | 3.656254 | 3.654267 |
| burleske.jpg | 3.748803 | 3.889864 | 3.585998 | 3.549407 | 3.624310 | 3.679677 |
| minuet.jpg | 3.692919 | 3.700736 | 3.703757 | 3.688776 | 3.703710 | 3.697980 |
| matilda.jpg | 3.681597 | 3.692134 | 3.719346 | 3.748517 | 3.700038 | 3.708327 |
| saints.jpg | 3.698648 | 3.704156 | 3.709525 | 3.751408 | 3.721238 | 3.716995 |
| auldPiano.jpg | 3.731556 | 3.730164 | 3.871970 | 3.789682 | 3.753766 | 3.775428 |
| turk_Page_1.jpg | 3.878129 | 3.889375 | 3.872360 | 3.874375 | 3.862451 | 3.875338 |
| silent.jpg | 3.872300 | 3.873490 | 3.879376 | 3.865442 | 3.888667 | 3.875855 |
| turk_Page_2.jpg | 3.902532 | 3.900434 | 3.886696 | 3.895608 | 3.899964 | 3.897047 |
| Grace3_Page_1.jpg | 4.037300 | 4.050284 | 4.036039 | 4.029502 | 4.042934 | 4.039212 |
| furelise.jpg | 4.334066 | 4.097082 | 4.128086 | 4.140342 | 4.149874 | 4.169890 |
| odePiano.jpg | 4.189703 | 4.183576 | 4.190034 | 4.183441 | 4.179460 | 4.185243 |
| patrol.jpg | 4.301393 | 4.286913 | 4.335763 | 4.279070 | 4.285363 | 4.297700 |
| Greensleeves.jpg | 4.352824 | 4.413997 | 4.306200 | 4.288876 | 4.413761 | 4.355132 |
| hall.jpg | 4.554884 | 4.495708 | 4.449027 | 4.379882 | 3.910232 | 4.357947 |
| Glad_Page_1.jpg | 4.391674 | 4.373223 | 4.367449 | 4.391360 | 4.374216 | 4.379584 |
| America2.jpg | 4.398615 | 4.491463 | 4.391984 | 4.471453 | 4.382392 | 4.427181 |
| Glad_Page_4.jpg | 4.761526 | 4.894959 | 4.230452 | 4.459443 | 4.146141 | 4.498504 |
| Glad_Page_3.jpg | 4.557140 | 4.586240 | 4.710793 | 4.779128 | 4.436036 | 4.613867 |
| canon_d_Page_4.jpg | 4.687045 | 4.709880 | 4.583609 | 4.591710 | 4.556507 | 4.625750 |
| canon_d_Page_2.jpg | 4.665476 | 4.745748 | 4.732482 | 4.598110 | 4.608144 | 4.669992 |
| sweetheart.jpg | 4.806301 | 4.839003 | 4.875896 | 4.822576 | 4.836318 | 4.836019 |
| canon_d_Page_1.jpg | 4.896712 | 4.864520 | 4.887435 | 4.924217 | 4.889144 | 4.892406 |
| Glad_Page_2.jpg | 4.926783 | 4.942598 | 5.144169 | 5.047779 | 4.963424 | 5.004951 |
| canon_d_Page_3.jpg | 5.375318 | 5.341669 | 5.330366 | 5.339723 | 5.337199 | 5.344855 |
| Hallelujah_Page_1.jpg | 5.405879 | 5.404677 | 5.408524 | 5.387396 | 5.434184 | 5.408132 |

***Table A-4:*** *Average run times (in seconds) of the input images*

## Appendix B: Optical Music Recognition Source Code

### Preprocessing

```
%-------------------------------------------------------------------------
% Function name:    preprocess
% Input arg(s):     Color/binary image
% Outputs arg(s):   Cropped image, staff-space height, staff-line height,
% Description:      Returns a cropped, binarized version of the input
%                   image
%-------------------------------------------------------------------------
function [new_img, lineHeight, spaceHeight] = preprocess(img)

    % Checks if image is color: if true, convert to gray
    if (ndims(img) == 3)
        gray_img = rgb2gray(img);
    else
        gray_img = img;
    end

    % If resultant is grayscale image, it is binarized using Otsu's
    if (~islogical(gray_img))
        thresh = graythresh(gray_img);
        img1 = imbinarize(gray_img, thresh);
    else
        img1 = gray_img;
    end

    img_bw = not(img1);

    % Call getStaffParam function to obtain reference lengths
    [lineHeight, spaceHeight] = getStaffParam(img_bw);

    % Horizontal projection for staff detection
    % - In this case, only the topmost and bottommost staff lines
    %   are relevant to get upper/lower bounds
    project_H = sum(img_bw, 2);
    thresh_H = max(project_H) * 0.5;
    x = find(project_H >= thresh_H);
    top_bound = min(x)-(5*(lineHeight + spaceHeight));
    bottom_bound = max(x)+(5*(lineHeight + spaceHeight));

    % Vertical projection
    % - Left/right-most bar lines are detected for
    %   side bounds
    project_V = sum(img_bw, 1);
    [~, loc] = findpeaks(project_V);
    left_bound = min(loc) - 20;
    right_bound = max(loc) + 20;

    % Boundary validation for all four sides
    % - Checks if calculated bounds exceed the image size
    if (left_bound < 1)
        left_bound = 1;
    end

    if (top_bound < 1)
        top_bound = 1;
    end

    if (right_bound > size(img_bw,2))
```

```
        right_bound = size(img_bw,2);
    end

    if (bottom_bound > size(img_bw,1))
        bottom_bound = size(img_bw,1);
    end

    % Return a cropped image based on calculated bounds
    new_img = img_bw(top_bound:bottom_bound, ...
        left_bound:right_bound);

end
```

### Staff parameter generator

```
%-------------------------------------------------------------------------
% Function name:    getStaffParam
% Input arg(s):     Binarized image
% Outputs arg(s):   Staff-space height, staff-line height,
% Description:      Returns vital reference lengths to be used in
%                   subsequent functions
%-------------------------------------------------------------------------
function [lineHeight, spaceHeight]= getStaffParam(img)

    black_runs=[];
    white_runs=[];

    % Perform vertical run-length encoding on the image
    for i=1:size(img, 2)
        x=img(:,i)';
        len = encodeRL(x);
        black_runs = [black_runs len(1:2:end)];
        white_runs = [white_runs len(2:2:end)];
    end

    % Staffline height is determined by the black run with the most
    % occurrence
    lines = histcounts(white_runs, 1:max(white_runs));
    [~, lineHeight] = max(lines);

    % Staffspace height is determined by the white run with the most
    % occurrence
    spaces = histcounts(black_runs, 1:max(black_runs));
    [~, spaceHeight] = max(spaces);

end
```

### Run-length encoding

```
%-------------------------------------------------------------------------
% Function name:    encodeRL
% Input arg(s):     One column from an image
% Outputs arg(s):   Run lengths
% Description:      Calculates lengths of white and black sequences in
%                   the input column
%-------------------------------------------------------------------------
function [run] = encodeRL(col_in)
```

```
    % Return indices where unequal adjacent terms occur
    idx = [find(col_in(1:end-1) ~= col_in(2:end)), length(col_in)];
    % Return difference of adjacent indices
    run = diff([ 0 idx ]);

    % Since image is BW, if top pixel is white, let first value be 0
    if (col_in(1) == 1)
        run = [0 run];
    end
end
```

### Staff detection

```
%-------------------------------------------------------------------------
% Function name:    staffDetect
% Input arg(s):     Binarized image
% Outputs arg(s):   Staffline row locations and lengths
% Description:      Returns location of and thickness of stafflines
%                   through horizontal projection
%-------------------------------------------------------------------------
function [staffLines, staffHeights] = staffDetect(img_bw)

    % A staffline is detected if a row has black pixels which is
    % at least half the maximum number of black pixels per row
    project_H = sum(img_bw, 2);
    thresh_H = 0.67 * size(img_bw, 2);
    staffLines = find(project_H >= thresh_H);

    % Once staff lines are detected, run-length encoding will be applied
    % to one pixel column with only the stafflines to determine the
    % height of each line
    imgTemp = ones(size(img_bw,1),1);
    imgTemp(staffLines) = 0;
    len = encodeRL(imgTemp');
    staffHeights = len(1:2:end);
    staffHeights = staffHeights(staffHeights ~= 0);

end
```

### Staff division

```
%-------------------------------------------------------------------------
% Function name:    staffDivider
% Input arg(s):     sheet image, staffline locations, space/line heights
% Output arg(s):    staff dividers
% Description:      Returns the necessary staff dividers to properly
%                   segment sheet music images with correspondong staff
%                   types
%-------------------------------------------------------------------------
function [dividers] = staffDivider(img, staffLines, lineHeight, spaceHeight)

    % Assume staves are grand staff:
    % - insert a divider after two staff pairs,
    %   then record indices of staff dividers
    staffLines(diff(staffLines) == 1) = [];
    dividers(1) = max(staffLines(1)-(4*(spaceHeight+lineHeight)), 1);

    % if only one staff present, add divider at end
```

```
    if (length(staffLines) == 5)
        dividers(2) = max(staffLines(5)+(4*(spaceHeight+lineHeight)), 1);
    else
        for i=10:10:length(staffLines(:))-1
            dividers(end+1) = ceil(staffLines(i) ...
                + ((staffLines(i+1) - staffLines(i)) / 2));
        end
        % Add staff length to indices
        dividers(end+1) = length(img(:,1));

        % --------------------- Grand staff checker ---------------------
        % - if pair of staves is not a grand staff, add a divider
        staff_thresh = staffLines(10) - staffLines(1);
        for i = 1:(length(staffLines) / 10)
            img_tmp = img(dividers(i):(dividers(i+1)-1),:);
            project_V = sum(img_tmp, 1);
            if (staff_thresh > max(project_V))
                dividers(end+1) = ceil(staffLines((i*10)-5) ...
                    + ((staffLines((i*10)-4) - staffLines((i*10)-5)) / 2));
            end
        end
    end

    % Indices must be sorted for subsequent use in other functions
    dividers = sort(dividers);
end
```

## Staff removal

```
%-------------------------------------------------------------------------
% Function name:    staffRemove
% Input arg(s):     Binarized image
% Outputs arg(s):   Staffline row locations and lengths
% Description:      Returns location of and thickness of stafflines
%                   through horizontal projection
%-------------------------------------------------------------------------
function [imgNoStaff] = staffRemove(img_bw, staffLines, staffHeights, ...
    spaceHeight)

    % Invert image to have a white background, then eroding before next step
    img_new = ~img_bw;
    img_new = imerode(img_new, strel('disk',1));
    CC = bwconncomp(img_new, 4);
    stats = regionprops(CC, 'BoundingBox', 'Eccentricity', ...
        'Circularity', 'Area');
    idx = [];

    for i = 1:numel(stats)
        % bounding box parameters
        z = stats(i).BoundingBox;

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % whole/half notes inside staff heights will be temporarily filled
        % before staff removal to ensure noteheads will not be degraded
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        if  ... % hole length and width based on staff space height ...
            ((((z(3) >= (0.7*spaceHeight)) && z(3) <= (1.75*spaceHeight)) && ...
            (z(4) >= (0.7*spaceHeight) && z(4) <= spaceHeight)) && ...
            ... % half note hole is more eccentric and little less circular
            ((stats(i).Eccentricity >= 0.9 && stats(i).Eccentricity <= 1) && ...
```

```
            (stats(i).Circularity >= 0.6 && stats(i).Circularity < 0.9)) || ...
            ... % whole note hole is less eccentric and more circular
            ((stats(i).Eccentricity >= 0.6 && stats(i).Eccentricity <= 0.675) &&...
            (stats(i).Circularity >= 1.05 && stats(i).Circularity < 1.2)))

            idx = [idx; i];
        end
    end
    tempImg2 = imdilate(ismember(labelmatrix(CC), idx), strel('disk', 1));
    imgNoStaff = or(tempImg2,img_bw);

    % Staff lines differences are calculated in order to uniquify the lines
    % just before removal.
    staffDiff = [diff(staffLines); 0];
    newSL = staffLines(staffDiff ~= 1);

    % Staff removal part: remove white pixel column with no symbol segment
    % attached on either top or bottom of lines
    for i = 1:length(newSL)
        for j = 1:size(img_bw,2)
            if (unique(imgNoStaff(newSL(i)-staffHeights(i)+1:...
                    newSL(i),j)) == 1)
                imgNoStaff(newSL(i)-staffHeights(i)+1:newSL(i),j) ...
                = ~(isequal(imgNoStaff(newSL(i)-staffHeights(i)...
                ,j),0) && isequal(imgNoStaff(newSL(i)+1,j),0));
            end
        end
    end

    % Put the holes back in the no-staff image
    imgNoStaff = ~xor(imgNoStaff,tempImg2);

end
```

## Sheet music segmentation

```
%-------------------------------------------------------------------------
% Function name:    scoreToSections
% Input arg(s):     Binarized image
% Outputs arg(s):   (1 x n) cell of sheet music sections
% Description:      Divides original sheet music image into sections based
%                   on staff type (solo or grand)
%-------------------------------------------------------------------------
function [sections, newStaffLines] = scoreToSections(img_no_staff, ...
    staffLines, dividers)
    for a = 1:length(dividers)-1
        % Individual hypermeasures is assigned as a cell
        sections{a} = img_no_staff(dividers(a)+1:dividers(a+1)-1,:);
        staff_idx = staffLines(staffLines >= dividers(a) & ...
            staffLines <= dividers(a+1));

        % staff lines indices are recalculated based on each section
        newStaffLines{a} = staff_idx - dividers(a);
    end
end
```

## Importing dataset

```
%-------------------------------------------------------------------------
% Function name:    readDataset
% Input arg(s):     None
% Outputs arg(s):   Musical symbols and whole notes
% Description:      Returns a categorized set of notes and musical symbols
%                   to be used for template matching
%-------------------------------------------------------------------------
function [clefSyms, accidentalSyms, timeSigSyms, wholeNotes, restSyms, ...
    dotSyms, otherSyms, tieSlurs] = readDataset

    ext = '.bmp';
    %-----------------------------------------------------
    % Read clefs (15)
    trebleClef1 = imread('dataset\clefs\trebleClef1.bmp');
    trebleClef2 = imread('dataset\clefs\trebleClef2.bmp');
    trebleClef3 = imread('dataset\clefs\trebleClef3.bmp');
    trebleClef4 = imread('dataset\clefs\trebleClef4.bmp');
    trebleClef5 = imread('dataset\clefs\trebleClef5.bmp');
    bassClef1 = imread('dataset\clefs\bassClef1.bmp');
    bassClef2 = imread('dataset\clefs\bassClef2.bmp');
    bassClef3 = imread('dataset\clefs\bassClef3.bmp');
    bassClef4 = imread('dataset\clefs\bassClef4.bmp');
    bassClef5 = imread('dataset\clefs\bassClef5.bmp');
    bassClef6 = imread('dataset\clefs\bassClef6.bmp');
    altoClef1 = imread('dataset\clefs\altoClef1.bmp');
    altoClef2 = imread('dataset\clefs\altoClef2.bmp');
    altoClef3 = imread('dataset\clefs\altoClef3.bmp');
    altoClef4 = imread('dataset\clefs\altoClef4.bmp');
    clefs = {trebleClef1, trebleClef2, trebleClef3, trebleClef4, ...
        trebleClef5, bassClef1, bassClef2, bassClef3, bassClef4, ...
        bassClef5, bassClef6, altoClef1, altoClef2, altoClef3, altoClef4};
    category(:,1:length(clefs)) = {'clef'};
    type = {'treble', 'treble', 'treble', 'treble', 'treble', ...
        'bass', 'bass', 'bass', 'bass', 'bass', 'bass', ...
        'alto', 'alto', 'alto', 'alto'};
    clefSyms = [clefs', category', type' {}];
    %-----------------------------------------------------

    %-----------------------------------------------------
    % Read time signatures (26)
    category = {};
    ts22 = imread('dataset\timeSignature\ts22.bmp');
    ts24_1 = imread('dataset\timeSignature\ts24_1.bmp');
    ts24_2 = imread('dataset\timeSignature\ts24_2.bmp');
    ts24_3 = imread('dataset\timeSignature\ts24_3.bmp');
    ts34_1 = imread('dataset\timeSignature\ts34_1.bmp');
    ts34_2 = imread('dataset\timeSignature\ts34_2.bmp');
    ts34_3 = imread('dataset\timeSignature\ts34_3.bmp');
    ts34_4 = imread('dataset\timeSignature\ts34_4.bmp');
    ts34_5 = imread('dataset\timeSignature\ts34_5.bmp');
    ts34_6 = imread('dataset\timeSignature\ts34_6.bmp');
    ts34_7 = imread('dataset\timeSignature\ts34_7.bmp');
    ts44_1 = imread('dataset\timeSignature\ts44_1.bmp');
    ts44_2 = imread('dataset\timeSignature\ts44_2.bmp');
    ts44_3 = imread('dataset\timeSignature\ts44_3.bmp');
    ts44_4 = imread('dataset\timeSignature\ts44_4.bmp');
    ts44_5 = imread('dataset\timeSignature\ts44_5.bmp');
    ts54 = imread('dataset\timeSignature\ts54.bmp');
    ts38 = imread('dataset\timeSignature\ts38.bmp');
    ts68_1 = imread('dataset\timeSignature\ts68_1.bmp');
```

```
    ts68_2 = imread('dataset\timeSignature\ts68_2.bmp');
    ts98 = imread('dataset\timeSignature\ts98.bmp');
    ts128 = imread('dataset\timeSignature\ts128.bmp');
    common = imread('dataset\timeSignature\common.bmp');
    cut1 = imread('dataset\timeSignature\cut1.bmp');
    cut2 = imread('dataset\timeSignature\cut2.bmp');
    cut3 = imread('dataset\timeSignature\cut3.bmp');
    timeSig = {ts22, ts24_1, ts24_2, ts24_3, ts34_1, ts34_2, ts34_3, ts34_4,
ts34_5, ...
        ts34_6, ts34_7, ts44_1, ts44_2, ts44_3, ts44_4, ts44_5, ts54, ...
        ts38, ts68_1, ts68_2, ts98, ts128, common, cut1, cut2, cut3};
    category(:,1:length(timeSig)) = {'timeSignature'};
    type = {'2/2', '2/4', '2/4', '2/4', '3/4', '3/4', '3/4', '3/4', '3/4', '3/4',
...
        '3/4', '4/4', '4/4', '4/4', '4/4', '4/4', '5/4', '3/8', '6/8', ...
        '6/8', '9/8', '12/8', '4/4', '2/2', '2/2', '2/2'};
    beat = {2, 2, 2, 2, 3, 3, 3, 3, 3, 3, ...
        3, 4, 4, 4, 4, 4, 5, 3, 6, ...
        6, 9, 12, 4, 2, 2, 2};
    timeSigSyms = [timeSig' category' type' beat'];
    %-------------------------------------------------------

    %-------------------------------------------------------
    % Read whole notes (8)
    category = {};
    type = {};
    beat = {};
    loc = 'dataset\notes\whole';
    for i = 1:8
        img = imread(strcat(loc, num2str(i), ext));
        whole{i} = img;
        category(i) = {'note'};
        type(i) = {'whole'};
        beat(i) = {1};
    end
    wholeNotes = [whole' category' type' beat'];
    %-------------------------------------------------------

    %-------------------------------------------------------
    % Read rests (7)
    category = {};
    rest12 = imread('dataset\rests\rest12.bmp');
    rest4_1 = imread('dataset\rests\rest4_1.bmp');
    rest4_2 = imread('dataset\rests\rest4_2.bmp');
    rest8 = imread('dataset\rests\rest8.bmp');
    rest16 = imread('dataset\rests\rest16.bmp');
    rest32 = imread('dataset\rests\rest32.bmp');
    rest64 = imread('dataset\rests\rest64.bmp');
    rests = {rest12, rest4_1, rest4_2, rest8, rest16, rest32, rest64};
    category(:,1:length(rests)) = {'rest'};
    type = {'wholehalf', 'quarter', 'quarter', '8th', '16th', '32nd', '64th'};
    beat = {1, 0.25, 0.25, 0.125, 0.0625, 0.03125, 0.015625};
    restSyms = [rests', category', type', beat'];
    %-------------------------------------------------------

    %-------------------------------------------------------
    % Read dots (6)
    category = {};
    type = {};
    beat = {};
    loc = 'dataset\dots\dot';
    for i = 1:6
        img = imread(strcat(loc, num2str(i), ext));
```

```
        dots{i} = img;
        category(i) = {'dot'};
        type(i) = {'augmentation'};
        value(i) = {0};
    end
    dotSyms = [dots' category' type' value'];
    %-----------------------------------------------------

    %-----------------------------------------------------
    % Read accidentals (5)
    category = {};
    flat1 = imread('dataset\accidental\flat1.bmp');
    flat2 = imread('dataset\accidental\flat2.bmp');
    sharp1 = imread('dataset\accidental\sharp1.bmp');
    sharp2 = imread('dataset\accidental\sharp2.bmp');
    natural = imread('dataset\accidental\natural.bmp');
    accidentals = {flat1, flat2, sharp1, sharp2, natural};
    category(:,1:length(accidentals)) = {'accidental'};
    type = {'flat', 'flat', 'sharp', 'sharp', 'natural'};
    pitch = {-1, -1, 1, 1, 0};
    accidentalSyms = [accidentals', category', type', pitch'];
    %-----------------------------------------------------

    %-----------------------------------------------------
    % Read fermatas (4)
    category = {};
    type = {};
    beat = {};
    loc = 'dataset\other\fermata';
    for i = 1:4
        img = imread(strcat(loc, num2str(i), ext));
        other{i} = img;
        category(i) = {''};
        type(i) = {'fermata'};
        beat(i) = {2};
    end
    otherSyms = [other' category' type' beat'];
    %-----------------------------------------------------

    %-----------------------------------------------------
    % Read slurs (6)
    type = {};
    beat = {};
    loc = 'dataset\other\slur';
    for i = 1:6
        img = imread(strcat(loc, num2str(i), ext));
        other{i} = img;
        category(i) = {''};
        type(i) = {'slur'};
        beat(i) = {0};
    end
    slurs = [other' category' type' beat'];
    tieSlurs = slurs;
    %-----------------------------------------------------

    %-----------------------------------------------------
    % Read ties (15)
    category = {};
    type = {};
    beat = {};
    loc = 'dataset\other\tie';
    for i = 1:17
        img = imread(strcat(loc, num2str(i), ext));
```

```
        tie{i} = img;
        category(i) = {''};
        type(i) = {'tie'};
        beat(i) = {0};
    end
    ties = [tie' category' type' beat'];
    tieSlurs = [tieSlurs; ties];
    %-------------------------------------------------

end
```

## Clef detection

```
%-------------------------------------------------------------------------
% Function name:    detectClefs
% Input arg(s):     section, staffspace height, staff lines, clef symbols
% Outputs arg(s):   clef(s) definition, key signature ROI LH boundary
% Description:      Detects clef(s) in the lefthand side of sections, and
%                   returns the starting column value for key signature
%                   detection
%-------------------------------------------------------------------------
function [clefs, bound] = detectClefs(section, spaceHeight, ...
    staffLines, clefSyms, debug)

    % ROI is from the leftmost edge to 30 times the staff spaceheight
    tempImg = ones(size(section));
    boundary = 1:30*spaceHeight;
    tempImg(:,boundary) = section(:,boundary);

    % In the case of piano sheets, the brace is removed before template
    % matching
    if ((max(staffLines) - min(staffLines)) > 10*spaceHeight)
        project_V = sum(~tempImg,1);
        braceLine = (project_V >= 0.9 * (max(staffLines) - min(...
            staffLines)));
        tempImg(:,braceLine) = 1;
    end

    % Apply closing to ROI to connect dots from bass clef
    z = imclose(~tempImg, strel('rectangle', [5, 6])); % 6 8
    stats = regionprops(z, 'BoundingBox');

    symbol = {};
    loc = {};
    category = {};
    type = {};
    value = {};
    line = {};
    edge = [];
    symCount = 1;
    for j = 1:numel(stats)
        % Extract component boundaries
        z = stats(j).BoundingBox;
        h = ceil(z(2))+ceil(z(4));
        w = ceil(z(1))+ceil(z(3));
        if (h < 1)
            h = 1;
        elseif (w < 1)
            w = 1;
        elseif (h >= size(section,1))
```

```
            h = size(section,1)-1;
        elseif (w >= size(section,2))
            w = size(section,2)-1;
        end
        boxHeight = ceil(z(2)):h;
        boxWidth = ceil(z(1)):w;
        % Potential clef symbols are chosen based on symbol height
        % and width
        if ((z(3) >= 2*spaceHeight && z(3) <= 4.5*spaceHeight) ...
            && (z(4) >= 3*spaceHeight && z(4) <= 9.5*spaceHeight))
            % Copy symbol in a temporary image holder
            symTmp = tempImg(boxHeight, boxWidth);
            % Perform template matching with clef dataset
            corr_val = [];
            for jj = 1:length(clefSyms)
                clefTmp = imresize(clefSyms{jj},[length(boxHeight) ...
                        length(boxWidth)]);
                % calculate 2-D correlation between potential clef and
                % dataset
                corr_val = [corr_val corr2(symTmp, clefTmp)];
            end
            % If the maximum correlation coefficient is >= 0.5, clef
            % parameters are saved
            [corrPct, idx] = max(corr_val);
            if (corrPct >= 0.5)
                symbol{symCount} = symTmp;
                loc(symCount) = {[ceil(z(2)) h ceil(z(1)) w]};
                category(symCount) = clefSyms(idx,2);
                type(symCount) = clefSyms(idx,3);
                edge(symCount) = w;
                % New parameters generated based on clef type to be used
                % in phase 3 (reconstruction)
                if (strcmp(cell2mat(clefSyms(idx,3)),'treble'))
                    value(symCount) = {'G4'};
                    line(symCount) = {4};
                elseif (strcmp(cell2mat(clefSyms(idx,3)),'bass'))
                    value(symCount) = {'F3'};
                    line(symCount) = {2};
                elseif (strcmp(cell2mat(clefSyms(idx,3)),'alto'))
                    value(symCount) = {'C4'};
                    line(symCount) = {3};
                end
                symCount = symCount + 1;
                if (debug == 1)
                    rectangle('Position',stats(j).BoundingBox, ...
                    'EdgeColor','m', 'LineWidth',2);
                end
            end
        end
    end
    % Bound is the starting point of the key signature detection
    bound = max(edge);
    % Combine parameters for complete clef definitions
    clefs = [symbol', loc', category', type', value', line'];
end
```

## Key signature detection

```
%-------------------------------------------------------------------------
% Function name:    detectKeySig
% Input arg(s):     section, staffspace height, staff lines, key symbols,
%                   section count, ROI boundaries
% Outputs arg(s):   Key(s), key signature, next part boundary
% Description:      Detects the sheet music's key signature, and returns
%                   the keys for reconstruction
%-------------------------------------------------------------------------
function [keySig, keys, newTermBound] = detectKeySig(section, ...
    spaceHeight, sectionCount, newStaffLines, initBound, termBound, ...
    keySyms, debug)

    % Typical ROI is in between the clef(s) and time signature(s)
    tempImg = ones(size(section));
    boundary = initBound:termBound;
    tempImg(:,boundary) = section(:,boundary);

    % Close any symbols that might have been broken during staff removal
    z = imclose(~tempImg, strel('line', 3, 90));
    cc = bwconncomp(z);
    stats = regionprops(cc, 'BoundingBox');

    symbol = {};
    loc = {};
    category = {};
    type = {};
    value = {};
    symIdx = {};
    edge = [];
    leftBounds = [];
    symCount = 1;


    for j = 1:numel(stats)
        % Extract component boundaries
        z = stats(j).BoundingBox;
        h = ceil(z(2))+z(4);
        w = ceil(z(1))+z(3);
        boxHeight = ceil(z(2)):h;
        boxWidth = ceil(z(1)):w;

        % Checks for potential flats or sharps present based on height and
        % width
        if ((z(3) >= 0.75*spaceHeight && z(3) <= 2.25*spaceHeight) && ...
            (z(4) >= 2*spaceHeight && z(4) <= 3.75*spaceHeight))
            % Each potential key is isolated and copied in a temporary
            % image holder
            tempImg = ismember(labelmatrix(cc),j);
            symTmp = ~tempImg(boxHeight, boxWidth);
            % 2-D correlation template matching with key signature dataset
            corr_val = [];
            for jj = 1:length(keySyms)
                keyTmp = imresize(keySyms{jj},[length(boxHeight), ...
                    length(boxWidth)]); %z(3)+1]);
                corr_val = [corr_val corr2(symTmp, keyTmp)];
            end
            % If the maximum correlation coefficient is >= 0.5, key
            % parameters are saved
            [corrPct, idx] = max(corr_val);
            if (corrPct >= 0.5)
```

```
                % Every right-hand edge of the detected key is saved
                edgeW = w;
                % In the case of flat symbol, the hole is extracted to be
                % used in phase 3
                if (strcmp(cell2mat(keySyms(idx,3)), 'flat'))
                    flatImg = ones(size(tempImg));
                    flatImg(boxHeight, boxWidth) = section(boxHeight, ...
                        boxWidth);
                    flatImg = ~xor(imfill(~flatImg, 'holes'), flatImg);
                    flatStats = regionprops(flatImg, 'BoundingBox');
                    z = flatStats(1).BoundingBox;
                    stats(j).BoundingBox = z;
                    h = ceil(z(2))+z(4);
                    w = ceil(z(1))+z(3);
                    boxHeight = ceil(z(2)):h;
                    boxWidth = ceil(z(1)):w;
                    symTmp = ~flatImg(boxHeight, boxWidth);
                end
                symbol{symCount} = symTmp;
                loc(symCount) = {[ceil(z(2)) h ceil(z(1)) w]};
                category(symCount) = keySyms(idx,2);
                type(symCount) = keySyms(idx,3);
                value(symCount) = keySyms(idx,4);
                symIdx(symCount) = {ceil(z(1))};
                edge(symCount) = edgeW+3;
                symCount = symCount + 1;
                if (debug == 1)
                    rectangle('Position',stats(j).BoundingBox, ...
                        'EdgeColor','r', 'LineWidth',2);
                end
            end
        end

        % Checks for time signature or notes only for boudary sections
        % of subsequent sections after 1
        if (sectionCount == 1)
            if ((z(3) >= 1.5*spaceHeight && z(3) <= 3*spaceHeight) && ...
                (z(4) >= 1.75*spaceHeight && z(4) <= 5.5*spaceHeight))
                leftBounds = [leftBounds ceil(z(1))];
            end
        end
    end

    % Next part boundary definition based on section status
    if (sectionCount == 1 && ~isempty(leftBounds))
        newTermBound = min(leftBounds)-2;
    elseif (sectionCount == 1 && isempty(symbol))
        newTermBound = initBound;
    elseif (~(isempty(symbol) || ~isempty(leftBounds)))
        newTermBound = max(edge)-2;
    else
        newTermBound = initBound;
    end

    % Combine parameters for complete key definitions
    if(isempty(symbol))
        keys = [];
    else
        keys = [symbol', loc', category' type', value', symIdx'];
    end

    % Sets the number of total keys based on staff type (solo/grand)
    if (length(newStaffLines) >= 10 && max(newStaffLines) ...
```

```
        - min(newStaffLines) > 5*spaceHeight)
        totalKeys = (symCount-1)/2;
    else
        totalKeys = symCount-1;
    end

    % Key signature definition based on total key count
    if (totalKeys == 0)
        keySig = 'C/Am';
    else
        % Key signature with sharp(s)
        if (strcmpi(type{1},'sharp'))
            if (totalKeys == 1)
                keySig = 'G/Em';
            elseif (totalKeys == 2)
                keySig = 'D/Bm';
            elseif (totalKeys == 3)
                keySig = 'A/F#m';
            elseif (totalKeys == 4)
                keySig = 'E/C#m';
            elseif (totalKeys == 5)
                keySig = 'B/G#m';
            elseif (totalKeys == 6)
                keySig = 'F#/D#m';
            elseif (totalKeys == 7)
                keySig = 'C#/A#m';
            end
        % Key signature with flat(s)
        else
            if (totalKeys == 1)
                keySig = 'F/Dm';
            elseif (totalKeys == 2)
                keySig = 'Bb/Gm';
            elseif (totalKeys == 3)
                keySig = 'Eb/Cm';
            elseif (totalKeys == 4)
                keySig = 'Ab/Fm';
            elseif (totalKeys == 5)
                keySig = 'Db/Bbm';
            elseif (totalKeys == 6)
                keySig = 'Gb/Ebm';
            elseif (totalKeys == 7)
                keySig = 'Cb/Abm';
            end
        end
    end

end
```

### Time signature detection

```
%-------------------------------------------------------------------------
% Function name:    detectTimeSig
% Input arg(s):     section, staffspace height, ROI boundary, TS dataset
% Outputs arg(s):   time signature definition, next part boundary
% Description:      Detects time signature in the lefthand side of sec-
%                   tions, and returns the starting column value for
%                   next step detection
%-------------------------------------------------------------------------
function [timeSignature, bound] = detectTimeSig(section, spaceHeight, ...
```

```
    initBound, timeSigSyms, debug)

    % ROI is between righthand edge of keys and four times the spaceheight
    % distance after it
    tempImg = ones(size(section));
    boundary = initBound:(initBound+4*spaceHeight);
    tempImg(:,boundary) = section(:,boundary);

    % Applying closing to combine the two numbers
    z = imclose(~tempImg, strel('rectangle', [6 8]));
    stats = regionprops(z, 'BoundingBox');

    symbol = {};
    loc = {};
    category = {};
    type = {};
    beat = {};
    edge = [];
    symCount = 1;
    for j = 1:numel(stats)
        % Extract component boundaries
        z = stats(j).BoundingBox;
        h = ceil(z(2))+z(4);
        w = ceil(z(1))+z(3);
        boxHeight = ceil(z(2)):h;
        boxWidth = ceil(z(1)):w;

        % Potential time signature symbols are chosen based on symbol
        % height and width
        if ((z(3) >= 1.375*spaceHeight && z(3) <= 4*spaceHeight) && ...
            (z(4) >= 1.75*spaceHeight && z(4) <= 6*spaceHeight))
            % Copy symbol in a temporary image holder
            symTmp = tempImg(boxHeight, boxWidth);
            % Perform template matching with time signature dataset
            corr_val = [];
            for jj = 1:length(timeSigSyms)
                timeTmp = imresize(timeSigSyms{jj},[length(boxHeight),...
                    length(boxWidth)]); %z(3)]);
                % calculate 2-D correlation between potential TS and
                % dataset
                corr_val = [corr_val corr2(symTmp, timeTmp)];
            end
            % If the maximum correlation coefficient is >= 0.6, clef
            % parameters are saved
            [corrPct, idx] = max(corr_val);
            if (corrPct >= 0.6)
                symbol{symCount} = symTmp;
                loc(symCount) = {[ceil(z(2)) h ceil(z(1)) w]};
                category(symCount) = timeSigSyms(idx,2);
                type(symCount) = timeSigSyms(idx,3);
                beat(symCount) = timeSigSyms(idx,4);
                edge(symCount) = w;
                symCount = symCount + 1;
                if (debug == 1)
                    rectangle('Position',stats(j).BoundingBox, ...
                        'EdgeColor','#D95319', 'LineWidth',2);
                end
            end
        end
    end
    % Bound is the starting point of the note sorter part
    bound = max(edge);
    % Combine parameters for complete time signature definitions
```

```
    timeSignature = [symbol', loc', category', type', beat'];
end
```

## Note sorter

```
%--------------------------------------------------------------------------
% Function name:    noteSorter
% Input arg(s):     section, ROI boundary, staff space/line heights
% Outputs arg(s):   unbeamed, beamed, and other symbol sections
% Description:      Sorts the components based on type (unbeamed notes,
%                   beamed notes, or non-note symbols)
%--------------------------------------------------------------------------
function [unbeamedSec, beamedSec, otherSymsSec] = noteSorter(section, ...
    bound, spaceHeight, lineHeight)

    % The ROI is the area just after the time signature's rightmost boundary
    tempImg = ones(size(section));
    boundary = bound:size(section,2);
    tempImg(:,boundary) = section(:,boundary);

    %----------------------------------------------------------------------
    % First pass: Connected components that fall in the stemmed note/chord
    % and whole note/chord category are sorted
    %----------------------------------------------------------------------
    cc = bwconncomp(~tempImg);
    unbeamedStats = regionprops(cc, 'BoundingBox', 'Circularity');
    unbeamedSec = ones(size(section));
    for i = 1:numel(unbeamedStats)
        z = unbeamedStats(i).BoundingBox;
        % Symbols that fall under unbeamed category:
        % - half-64th note/chords
        % - whole notes/chords
        if ((z(3) >= 1.375*spaceHeight && z(3) <= 3.5*spaceHeight) && ...
            (z(4) >= 2*spaceHeight && z(4) <= 15*spaceHeight))
            % Perform stem check on current symbol
            temp = ismember(labelmatrix(cc), i);
            tempOpen = imopen(temp, strel('line', (1.75*spaceHeight), 90));
            stemStats = regionprops(tempOpen);

            % If stem(s) exist, it will be included in the unbeamed section
            if(numel(stemStats) ~= 0)
                unbeamedSec = xor(ismember(labelmatrix(cc),i), ...
                    unbeamedSec);
            end
        % Potential single whole notes; circularity is added in the
        % condition
        elseif ((z(3) >= 1.375*spaceHeight && z(3) <= 3.25*spaceHeight) ...
            && (z(4) >= spaceHeight && z(4) <= (1.5*spaceHeight)) && ...
            (unbeamedStats(i).Circularity > 0.2))
            unbeamedSec = xor(ismember(labelmatrix(cc),i), unbeamedSec);
        end
    end
    % Remove potential unbeamed symbols from original section
    beamedSec = xor(tempImg, unbeamedSec);

    %----------------------------------------------------------------------
    % Second pass: Connected components that fall in the other symbol
    % category are sorted
    %----------------------------------------------------------------------
    cc2 = bwconncomp(beamedSec);
```

```
    otherStats = regionprops(cc2, 'BoundingBox');
    otherSymsSec = ones(size(section));
    for j = 1:numel(otherStats)
        % Bounding box parameters for second pass
        z2 = otherStats(j).BoundingBox;
        if (((z2(3) < 20*spaceHeight)  && ...
            (z2(4) < 3*spaceHeight)) || ...
            ((z2(3) < 3*spaceHeight)  && ...
            (z2(4) < 6*spaceHeight)) || ...
            z2(3) <= spaceHeight)
            otherSymsSec = xor(ismember(labelmatrix(cc2),j), otherSymsSec);
        end
    end
    % Remove potential other symbols from original section
    beamedSec = xor(beamedSec, ~otherSymsSec);

    %--------------------------------------------------------------------
    % Third pass: Connected components that was initially counted as
    % beamed notes will be sorted
    %--------------------------------------------------------------------
    cc3 = bwconncomp(beamedSec);
    beamedStats = regionprops(cc3, 'BoundingBox');
    for k = 1:numel(beamedStats)
        % Perform opening on remaining symbols to detect stems
        tempImg3 = ismember(labelmatrix(cc3),k);
        tempOpen = imopen(tempImg3, strel('line', (2*spaceHeight + ...
            3*lineHeight), 90));
        stemStats2 = regionprops(tempOpen);

        % If two stems are detected, symbol is sent to beamed section,
        % otherwise, sent to other symbol section
        if (numel(stemStats2) < 2)
            otherSymsSec = xor(otherSymsSec, tempImg3);
            beamedSec = xor(beamedSec, tempImg3);
        end
    end
    % Invert final result as beamed section
    beamedSec = ~beamedSec;
end
```

## Unbeamed note detection

```
%--------------------------------------------------------------------------
% Function name:    detectUnbeamedNotes
% Input arg(s):     unbeamed section, staff space/line heights, whole
%                   note dataset
% Outputs arg(s):   unbeamed notes and ledger line locations
% Description:      Detects single and chord unbeamed notes from whole
%                   to 64th
%--------------------------------------------------------------------------
function [unbeamedNotes, ledgerLineLocs] = detectUnbeamedNotes(...
    section, spaceHeight, lineHeight, wholeNotes, debug)

    % Extract ledger lines for subsequent pitch assignment
    ledgerLines = imclose(section, strel('rectangle', ...
        [2, floor(spaceHeight * 2)]));
    cc = bwconncomp(~ledgerLines);
    stats = regionprops(cc, 'Area');
    % Ensures that only ledger lines are being removed and not segments
    % of adjacent notes
```

```
    idx = find([stats.Area] <= 12*spaceHeight);
    ledgerLines = ismember(labelmatrix(cc), idx);
    % Extends the ledger lines for location calculation
    lineExt = imdilate(ledgerLines, strel('line', ...
        2*size(section,2), 0));
    ledgerLineLocs = find(sum(lineExt, 2) ~= 0);

    % If ledger lines exist in the section, they are to be removed before
    % detection
    if(~isempty(ledgerLineLocs))
        section = ledgerRemove(section, ledgerLineLocs);
    end

    symbol = {};
    loc = {};
    category = {};
    type = {};
    value = {};
    symIdx = {};
    symCount = 1;

    % Perform a small opening to ensure that note edges are connected
    section = imopen(section, strel('line', 3, 0));
    noteStats = regionprops(~section, 'BoundingBox');
    % Examine individual component
    for i = 1:numel(noteStats)
        % Extract component boundaries
        tempImg = ones(size(section));
        unbeamedNoteHeads = ones(size(section));
        z = noteStats(i).BoundingBox;
        h = ceil(z(2))+z(4);
        w = ceil(z(1))+z(3);
        boxHeight = ceil(z(2)):h-1;
        boxWidth = ceil(z(1)):w-1;
        % Copy note in temporary holder
        tempImg(boxHeight, boxWidth) = section(boxHeight, boxWidth);
        %-----------------------------------------------------------
        % Stem detection: Determines if the symbol is either:
        % - (half-64th) note/chord         - whole note chords
        % - single whole notes
        %-----------------------------------------------------------
        % Checks for (half-64th) notes/chords:
        stemChecker = imerode(~tempImg, strel('line', ...
            (3*spaceHeight + 2*lineHeight), 90));
        stemChecker = imdilate(stemChecker, strel('line', ...
            (3*spaceHeight + 2*lineHeight), 90));
        stemStats = regionprops(stemChecker);
        % Checks for (half-64th) chords that were missed by first check:
        if (numel(stemStats) > 1)
            stemChecker = imopen(stemChecker, strel('line', ...
                4.25*spaceHeight, 90));
            stemStats = regionprops(stemChecker);
        % Checks for whole notes/chords:
        elseif (numel(stemStats) == 0)
            stemChecker = imopen(~tempImg, strel('line', ...
                1.75*spaceHeight, 90));
            stemStats = regionprops(stemChecker);
        end

        %-----------------------------------------------------------
        % Notehead detection: Section step to determine note type
        % - One or more: (quarter-64th) notes
        % - Zero: whole/half notes
```

```
        %-------------------------------------------------------
        % Because of the notehead's geometry, the structuring element
        % used is a line of length (spaceHeight), angled at 30 degrees
        noteHead = imerode(~tempImg, strel('line', ...
            spaceHeight, 30));
        % Detected noteheads are enhanced in order to be counted properly
        % *Second condition is added as a failsafe for smaller sheets
        if (spaceHeight > 10)
            noteHead = imopen(noteHead, strel('disk', 3));
        else
            noteHead = imopen(noteHead, strel('disk', 2));
        end
        cc = bwconncomp(noteHead);
        noteHeadStats = regionprops(noteHead, 'BoundingBox', 'Area');
        % If side-adjacent notes are present, they have to be broken
        % further for correct detection
        idx = find([noteHeadStats.Area] > 8*spaceHeight);
        % Isolate the partially detected notes
        tempImg2 = ismember(labelmatrix(cc), idx);
        % Remove isolated component from complete notehead image
        noteHead = xor(noteHead, ismember(labelmatrix(cc), idx));

        % Adjacent note segmentation
        if (~isequal(tempImg2, zeros(size(noteHead))))
            % The stem is extended horizontally and vertically
            stemExt = imdilate(stemChecker, strel('line', 2*lineHeight,...
                90));
            stemExt = imdilate(stemExt, strel('line', floor(0.5*...
                spaceHeight), 0));
            % By extending the stems, it ensures complete separation
            % between connected noteheads
            tempImg2 = or(~tempImg2, stemExt);
            noteHead = or(noteHead, ~tempImg2);
            noteHeadStats = regionprops(noteHead, 'BoundingBox');
        end

        %-------------------------------------------------------
        % Non-whole single note detection
        %-------------------------------------------------------
        if ((z(3) > spaceHeight && z(4) >= 2*spaceHeight))
            %---------------------------------------------------
            % Closed note detection: quarter-64th notes
            %---------------------------------------------------
            if (numel(stemStats) == 1)
                % If noteheads are detected, potential note falls on
                % (quarter-64)
                if (numel(noteHeadStats) ~= 0)
                    % Isolate the detected noteheads for counting
                    unbeamedNoteHeads(boxHeight, boxWidth) = ~noteHead(...
                        boxHeight, boxWidth);
                    % Increase the size of noteheads
                    noteHead2 = imdilate(noteHead, strel('disk', floor(...
                        spaceHeight*0.3)));
                    %-----------------------------------------------
                    % Flag detection: Because of its geometry, once the
                    %     noteheads are removed, flags are detected w/
                    %     a structuring element with length 1.25x of
                    %     staff spaceheight, leaning at an angle of -45
                    %     degrees
                    %-----------------------------------------------
                    filled1 = imopen(xor(~noteHead2, tempImg), strel(...
                        'line', floor(spaceHeight*1.25), -45));
                    cc1 = bwconncomp(filled1);
```

```
                % Returns the total number of flags detected
                flagCount = numel(regionprops(cc1));
                % If there are no flags detected initially, it is
                % possible that the note is upside down
                if(flagCount == 0)
                    % With an upside down note, the flags are angled
                    % at 45 degrees
                    filled2 = imopen(xor(~noteHead2, tempImg), ...
                        strel('line', floor(spaceHeight*1.25), 45));
                    cc2 = bwconncomp(filled2);
                    % Returns the total number of flags detected
                    flagCount = numel(regionprops(cc2));
                end
                % Determine the note type based on flag count
                if (flagCount >= 0 && flagCount <= 4)
                    if (flagCount == 0)
                        noteType = 'quarter';
                        noteVal = 0.25;
                    elseif (flagCount == 1)
                        noteType = '8th';
                        noteVal = 0.125;
                    elseif (flagCount == 2)
                        noteType = '16th';
                        noteVal = 0.0625;
                    elseif (flagCount == 3)
                        noteType = '32nd';
                        noteVal = 0.03125;
                    elseif (flagCount == 4)
                        noteType = '64th';
                        noteVal = 0.015625;
                    end
                    % Count the number of noteheads detected in
                    % the isolated note/chord
                    cc2 = bwconncomp(~unbeamedNoteHeads);
                    singleNote = regionprops(cc2, 'BoundingBox');
                    for ii = 1:numel(singleNote)
                        % Every notehead's boundaries are extracted
                        z2 = noteHeadStats(ii).BoundingBox;
                        h2 = ceil(z2(2))+z2(4);
                        w2 = ceil(z2(1))+z2(3);
                        boxHeight2 = ceil(z2(2)):h2-1;
                        boxWidth2 = ceil(z2(1)):w2-1;

                        % Each notehead's parameters are saved
                        chordNote = ismember(labelmatrix(cc2),ii);
                        symbol{symCount} = ~chordNote(boxHeight2, ...
                            boxWidth2);
                        loc(symCount) = {[ceil(z2(2)) h2-1 ceil(...
                            z2(1)) w2-1]};
                        category(symCount) = {'note'};
                        type(symCount) = {noteType};
                        value(symCount) = {noteVal};
                        symIdx(symCount) = {ceil(z2(1))};
                        symCount = symCount + 1;
                        if (debug == 1)
                            rectangle('Position',noteHeadStats(...
                            ii).BoundingBox, 'EdgeColor',...
                            '#EDB120', 'LineWidth',2);
                        end
                    end
                end
            %-------------------------------------------------------
            % Half note detection: Detection based on holes
```

```
            %-------------------------------------------------------
            else
                % Returns the number of holes detected in the note or
                % chord
                [~, cc, holeStats] = detectOpenNoteheads(tempImg,...
                    spaceHeight);

                for ii = 1:numel(holeStats)
                    % Every note hole's boundaries are extracted
                    z2 = holeStats(ii).BoundingBox;
                    h2 = ceil(z2(2))+z2(4);
                    w2 = ceil(z2(1))+z2(3);
                    boxHeight2 = ceil(z2(2)):h2;
                    boxWidth2 = ceil(z2(1)):w2;

                    % Each note hole's parameters are saved
                    singleHole = ismember(labelmatrix(cc),ii);
                    symbol{symCount} = ~singleHole(boxHeight2, ...
                        boxWidth2);
                    loc(symCount) = {[ceil(z2(2)) h2 ceil(z2(1)) ...
                        w2]};
                    category(symCount) = {'note'};
                    type(symCount) = {'half'};
                    value(symCount) = {0.5};
                    symIdx(symCount) = {ceil(z2(1))};
                    symCount = symCount + 1;
                    if (debug == 1)
                        rectangle('Position',holeStats(ii). ...
                            BoundingBox, 'EdgeColor','#EDB120', ...
                            'LineWidth',2);
                    end
                end
            end
        %-----------------------------------------------------------
        % Whole note chord detection: The detection process is
        %     similar to half note detection
        %-----------------------------------------------------------
        else
            % Returns the number of holes detected in the chord
            [~, cc, holeStats] = detectOpenNoteheads(tempImg, ...
                spaceHeight);
            for ii = 1:numel(holeStats)
                % Every note hole's boundaries are extracted
                z2 = holeStats(ii).BoundingBox;
                h2 = ceil(z2(2))+z2(4);
                w2 = ceil(z2(1))+z2(3);
                boxHeight2 = ceil(z2(2)):h2;
                boxWidth2 = ceil(z2(1)):w2;

                % Each note hole's parameters are saved
                singleHole = ismember(labelmatrix(cc),ii);
                symbol{symCount} = ~singleHole(boxHeight2, boxWidth2);
                loc(symCount) = {[ceil(z2(2)) h2 ceil(z2(1)) w2]};
                category(symCount) = {'note'};
                type(symCount) = {'whole'};
                value(symCount) = {1};
                    symIdx(symCount) = {ceil(z2(1))};
                symCount = symCount + 1;
                if (debug == 1)
                    rectangle('Position',holeStats(ii).BoundingBox, ...
                        'EdgeColor','#EDB120', 'LineWidth',2);
                end
            end
```

```
            end
        %-------------------------------------------------------
        % Single whole note detection: The only note type that
        %     undergoes template matching to distinguish it from
        %     non-note symbols that might have been included in
        %     the section
        %-------------------------------------------------------
        else
            % Detects the number of holes, which in this case, just 1
            [wholeNoteHole, ~, holeStats] = detectOpenNoteheads(...
                tempImg, spaceHeight);
            % Isolate whole note
            symTmp = section(boxHeight, boxWidth);
            corr_val = [];
            % 2-D correlation template matching with whole note dataset
            for jj = 1:length(wholeNotes)
                whole = imresize(wholeNotes{jj},[length(boxHeight), ...
                    length(boxWidth)]);
                corr_val = [corr_val corr2(symTmp, whole)];
            end
            % If the maximum correlation coefficient is >= 0.5, note
            % parameters are saved
            [corrPct, idx] = max(corr_val);
            if (corrPct >= 0.5)
                % For phase 3 assignment, the note hole bounds will be
                % extracted
                z2 = holeStats(1).BoundingBox;
                h2 = ceil(z2(2))+z2(4);
                w2 = ceil(z2(1))+z2(3);
                boxHeight2 = ceil(z2(2)):h2;
                boxWidth2 = ceil(z2(1)):w2;

                % Each note hole's parameters are saved
                symbol{symCount} = ~wholeNoteHole(boxHeight2, boxWidth2);
                loc(symCount) = {[ceil(z2(2)) h2 ceil(z2(1)) w2]};
                category(symCount) = wholeNotes(idx,2);
                type(symCount) = wholeNotes(idx,3);
                value(symCount) = wholeNotes(idx,4);
                symIdx(symCount) = {ceil(z2(1))};
                symCount = symCount + 1;
                if (debug == 1)
                    rectangle('Position',holeStats(1).BoundingBox, ...
                        'EdgeColor','#EDB120', 'LineWidth',2);
                end
            end
        end
    end

    % Combine parameters for complete unbeamed note definitions
    unbeamedNotes = [symbol', loc', category', type', value', symIdx'];
end
```

## Ledger line removal

```
%-------------------------------------------------------------------------
% Function name:    ledgerRemove
% Input arg(s):     unbeamed/beamed section, ledger line locations
% Outputs arg(s):   section without ledger lines
% Description:      Removes the lines that protrude through the noteheads
%                   of closed notes
```

```
%-------------------------------------------------------------------------
function [imgNoLedger] = ledgerRemove(section, ledgerLineLocs)

    % Calculate the heights of the ledger lines
    imgTemp = ones(size(section,1),1);
    imgTemp(ledgerLineLocs) = 0;
    len = encodeRL(imgTemp');
    ledgerHeights = len(1:2:end);
    ledgerHeights = ledgerHeights(ledgerHeights ~= 0);

    % Calculate reference values
    ledgerDiff = [diff(ledgerLineLocs); 0];
    newLL = ledgerLineLocs(ledgerDiff ~= 1);

    % Holes in image are filled to keep the integrity of the note once
    % the ledger lines are removed
    imgNoLedger = ~section;
    imgFilled = imfill(imgNoLedger, 'holes');
    holes = xor(imgNoLedger, imgFilled);

    % Ledger removal part: remove white pixel column with no symbol
    % segment attached on either top or bottom of lines
    for i = 1:length(newLL)
        for j = 1:size(section,2)
            if (unique(imgFilled(newLL(i)-ledgerHeights(i)+1:newLL(i),j)) == 1)
                imgFilled(newLL(i)-ledgerHeights(i)+1:newLL(i),j) ...
                = ~(isequal(imgFilled(newLL(i)-ledgerHeights(i),j),0) && ...
                isequal(imgFilled(newLL(i)+1,j),0));
            end
        end
    end

    % Put the holes back in the no-ledger image
    imgNoLedger = ~xor(holes,imgFilled);
end
```

## Open notehead detection

```
%-------------------------------------------------------------------------
% Function name:    detectOpenNoteHeads
% Input arg(s):     half/whole note image, spaceHeight
% Outputs arg(s):   note hole(s), connected components, note hole stats
% Description:      Counts the number of note holes that exist in whole
%                   and half notes
%-------------------------------------------------------------------------
function [noteHole, cc, holeStats] = detectOpenNoteheads(img, spaceHeight)

    % After buffering the note, it will be filled, and after performing
    % an XOR with the original note, only the hole(s) remain
    tempImg = img;
    % Ensures that vertical edges are completely connected
    tempImg = imopen(tempImg, strel('line', 2, 90));
    filledHoles = imfill(~tempImg, 'holes');
    noteHoles = ~xor(filledHoles, tempImg);

    % If no hole is detected, the note is not completely connected, thus,
    % an opening is applied to close any disconnect in the edges
    if (isequal(noteHoles, zeros(size(tempImg))))
        tempImg = imopen(img, strel('line', 3, -45));
        filledHoles = imfill(~tempImg, 'holes');
```

```
        noteHoles = ~xor(filledHoles, tempImg);
    end

    %---------------------------------------------------------------
    % First pass: Checks if the holes are partial(holes from notes
    %    with ledger lines) or complete
    %---------------------------------------------------------------
    cc1 = bwconncomp(noteHoles);
    stats = regionprops(cc1, 'Area');
    % Failsafe: If spaceheight is greater than the specified value, the
    % minimum/maximum area constraints are increased
    %    idx1: whole holes
    %    idx1_1: partial holes
    if (spaceHeight > 15)
        idx1 = find([stats.Area] >= 6*spaceHeight);
        idx1_1 = find([stats.Area] < 6*spaceHeight);
    else
        idx1 = find([stats.Area] >= 4.5*spaceHeight);
        idx1_1 = find([stats.Area] < 4.5*spaceHeight);
    end
    % Temporary holder that contains whole holes
    noteHoles = ismember(labelmatrix(cc1),idx1);
    % Temporary holder that contains partial holes
    tempNoteHole1 = ismember(labelmatrix(cc1),idx1_1);
    statsTmp = regionprops(tempNoteHole1);

    % Variation of the ledger line removal: Some part of the line is kept
    % to connect partial holes to form a 'complete' hole
    ledgerLines = imopen(~tempImg, strel('line', ...
        floor(spaceHeight * 1.375), 0));
    ledgerLines2 = imerode(ledgerLines, strel('line', ...
        floor(spaceHeight * 0.9), 0));

    %---------------------------------------------------------------
    % Second pass: Once the holes are fixed, this pass removes un-
    %    necessary lines from the ledger line detection part
    %---------------------------------------------------------------
    tempNoteHole2 = xor(~ledgerLines2, ~tempNoteHole1);
    cc2 = bwconncomp(tempNoteHole2);
    stats2 = regionprops(cc2,'Area');
    % Failsafe: For smaller sheets, the area constraint is reduced
    if (spaceHeight > 10)
        idx2 = find([stats2.Area] >= 5*spaceHeight);
    else
        idx2 = find([stats2.Area] >= 2*spaceHeight);
    end
    % If the constraint is met, the hole is passed through
    tempNoteHole2 = ismember(labelmatrix(cc2),idx2);
    noteHoles = xor(noteHoles, tempNoteHole2);


    % If only one partial hole is detected in a half note with ledger
    % line, it is possible that the edges are open, thus, it is fixed
    % in this part
    if (numel(statsTmp) == 1)
        img = or(img, noteHoles);
        tempImg = imopen(img, strel('line', 3, -45));
        filledHoles = imfill(~tempImg, 'holes');
        noteHoles = ~xor(filledHoles, tempImg);
    end

    %---------------------------------------------------------------
    % Third pass: Ensures that only holes are detected and passed
```

```
    %-------------------------------------------------------------------
    cc = bwconncomp(noteHoles);
    stats = regionprops(cc,'Area');
    idx = find([stats.Area] > 4*spaceHeight);
    noteHole = ismember(labelmatrix(cc),idx);

    % Parameters that will be used in the unbeamed note detection
    cc = bwconncomp(noteHole);
    holeStats = regionprops(cc, 'BoundingBox');

end
```

### Beamed note detection

```
%-------------------------------------------------------------------------
% Function name:    detectBeamedNotes
% Input arg(s):     beamed section, staff spaceheights
% Outputs arg(s):   beamed notes and ledger line locations
% Description:      Detects single and chord beamed notes from 8th to
%                   64th
%-------------------------------------------------------------------------
function [beamedNotes, ledgerLineLocs] = detectBeamedNotes(section, ...
    spaceHeight, debug)

    %-------------------------------------------------------------------
    % Preprocessing: Ledger removal and beam/stem isolation
    %-------------------------------------------------------------------
    % Using a diamond structuring element, note heads are temporarily
    % detected, then removed.
    tempOpen = imopen(~section, strel('diamond', floor(0.5*spaceHeight)));
    tempOpen = imdilate(tempOpen, strel('square', 1));
    tempOpen = xor(section, ~tempOpen);

    % Perform an opening to isolate only the beams with the stems attached
    % and small leftover components from the first part
    beamBuf = imopen(tempOpen, strel('line', floor(0.5*spaceHeight), 90));
    cc = bwconncomp(beamBuf);
       stats = regionprops(beamBuf, 'BoundingBox', 'Area');
    % Remove the small components from the image
       idx = find([stats.Area] > 2*spaceHeight);
    % Only the beam/stems are used for next calculation
    beamStems = ismember(labelmatrix(cc), idx);

    % Partial notehead detection (removing beam/stems)
    noteHeads = imopen(xor(section, beamStems), strel('line', 3, 0));
    % Ledger line detection similar to unbeamed note function
    ledgerLines = imclose(noteHeads, strel('rectangle', ...
        [2, floor(spaceHeight * 2)]));
    cc2 = bwconncomp(~ledgerLines);
    stats = regionprops(cc2, 'Area');
    % Ensures that only ledger lines are being removed and not segments
    % of adjacent notes
    idx2 = find([stats.Area] <= 12*spaceHeight);
    ledgerLines = ismember(labelmatrix(cc2), idx2);
    % Extends the ledger lines for location calculation
    lineExt = imdilate(ledgerLines, strel('line', ...
        2*size(section,2), 0));
    ledgerLineLocs = find(sum(lineExt, 2) ~= 0);

    % If ledger lines are detected, they will be removed to keep only the
```

```
% noteheads, otherwise, invert the notehead image
if(~isempty(ledgerLineLocs))
    newNoteHeads = ~ledgerRemove(noteHeads, ledgerLineLocs);
else
    newNoteHeads = ~noteHeads;
end

%-------------------------------------------------------------------
% Beam segmentation: spaces will be added in between notes in the
%     same beam so that each note(s) can be processed separately
%-------------------------------------------------------------------
cutBeamStems = zeros(size(section));
for i=1:length(idx)
    % Isolate individual beamstem, then extract the stems for distance
    % calculation
    beamStems2 = ismember(labelmatrix(cc), idx(i));
    tempImg2 = imopen(beamStems2, strel('line', spaceHeight, 90));
    % Get the centroid of each stem and only use the x value
    beamRefPts = regionprops(tempImg2, 'Centroid');
    beamCenters = cat(1,beamRefPts.Centroid);
    beamCenters = beamCenters(:,1);
    % Calculate the midpoints between existing stems
    beamDiff = floor(beamCenters(1:end-1) + (diff(beamCenters/2)));

    % Set three pixels to the left and right of the midpoints to 0
    % to segment notes
    for k = 1:length(beamDiff)
        beamStems2(:,beamDiff(k)-3:beamDiff(k)+3) = 0;
    end
    cutBeamStems = xor(cutBeamStems, beamStems2);
end
% The updated beams are reattached to the noteheads for next step
% detection
newSection = or(~noteHeads, cutBeamStems);
cc = bwconncomp(newSection);
sectionStats = regionprops(cc, 'BoundingBox', 'Area');

symbol = {};
loc = {};
category = {};
type = {};
value = {};
symIdx = {};
symCount = 1;

for j = 1:numel(sectionStats)
    % Isolate individual note/chord
    tempImg = ismember(labelmatrix(cc), j);

    %-------------------------------------------------------------
    % Beam counting: determines the type of note (8th-64th)
    %-------------------------------------------------------------
    % Extract the beam/stem part of the note
    beamTemp = and(cutBeamStems, tempImg);
    % Remove the stem, leaving only flags
    stem = imopen(beamTemp, strel('line', spaceHeight, 90));
    % Slightly erode the flags to ensure disconnect among flags
    beamTemp = imerode(xor(stem, beamTemp), strel('disk', 1));
    % Close distance between flags that were cut during stem removal
    beamTemp = imclose(beamTemp, strel('line', floor(spaceHeight...
        * 0.5), 0));
    beamStats = regionprops(beamTemp, 'BoundingBox');
    % Return the number of existing flags in the current note
```

```
beamStatsCount = numel(beamStats);

if (debug == 1)
    for jj = 1:beamStatsCount
        rectangle('Position',beamStats(jj).BoundingBox, ...
            'EdgeColor', 'b', 'LineWidth',2)
    end
end

%-------------------------------------------------------------
% Notehead counting: Similar to the unbeamed note function
%-------------------------------------------------------------
% Isolate noteheads from segmented note image
noteHeadTemp = and(newNoteHeads, tempImg);
% Detect individual notehead using a line structuring element
% with length of spaceHeight, angled at 30 degrees
noteHead = imerode(noteHeadTemp, strel('line', spaceHeight, 30));
% Detected noteheads are enhanced in order to be counted properly
% *Second condition is added as a failsafe for smaller sheets
if (spaceHeight > 10)
    noteHead = imopen(noteHead, strel('disk', 3));
else
    noteHead = imopen(noteHead, strel('disk', 2));
end
cc3 = bwconncomp(noteHead);
noteHeadStats = regionprops(cc3, 'BoundingBox', 'Area');
% If side-adjacent notes are present, they have to be broken
% further for correct detection
idx = find([noteHeadStats.Area] > 10*spaceHeight);
% Isolate the partially detected notes
tempImg2 = ismember(labelmatrix(cc3), idx);
% Remove isolated component from complete notehead image
noteHead = xor(noteHead, ismember(labelmatrix(cc3), idx));

% Adjacent note segmentation
if (~isequal(tempImg2, zeros(size(noteHead))))
    % The stem is extended horizontally and vertically
    stemExt = imdilate(stem, strel('line', 2*spaceHeight, 90));
    stemExt = imdilate(stemExt, strel('line', floor(0.5*...
        spaceHeight), 0));
    % By extending the stems, it ensures complete separation
    % between connected noteheads
    tempImg2 = or(~tempImg2, stemExt);
    noteHead = or(noteHead, ~tempImg2);
    noteHeadStats = regionprops(noteHead, 'BoundingBox');
end

% Determine the note type based on flag count
if (beamStatsCount >= 1 && beamStatsCount <= 4)
    if (numel(beamStats) == 1)
        noteType = '8th';
        noteVal = 0.125;
    elseif (numel(beamStats) == 2)
        noteType = '16th';
        noteVal = 0.0625;
    elseif (numel(beamStats) == 3)
        noteType = '32nd';
        noteVal = 0.03125;
    elseif (numel(beamStats) == 4)
        noteType = '64th';
        noteVal = 0.015625;
    end
```

```
            % Count the number of noteheads detected in the isolated
            % note/chord
            for ii = 1:numel(noteHeadStats)
                % Every notehead's boundaries are extracted
                z2 = noteHeadStats(ii).BoundingBox;
                h2 = ceil(z2(2))+z2(4);
                w2 = ceil(z2(1))+z2(3);
                boxHeight2 = ceil(z2(2)):h2;
                boxWidth2 = ceil(z2(1)):w2;

                % Each notehead's parameters are saved
                chordNote = ismember(labelmatrix(cc3),ii);
                symbol{symCount} = chordNote(boxHeight2, boxWidth2);
                loc(symCount) = {[ceil(z2(2)) h2 ceil(z2(1)) w2]};
                category(symCount) = {'note'};
                type(symCount) = {noteType};
                value(symCount) = {noteVal};
                symIdx(symCount) = {ceil(z2(1))};
                symCount = symCount + 1;

                if (debug == 1)
                    rectangle('Position',noteHeadStats(ii).BoundingBox, ...
                        'EdgeColor', '#4DBEEE', 'LineWidth',2);
                end
            end
        end
    end

    % Combine parameters for complete beamed note definitions
    beamedNotes = [symbol', loc', category', type', value', symIdx'];
end
```

## Other symbol detection

```
%-------------------------------------------------------------------------
% Function name:    detectOtherSymbols
% Input arg(s):     non-note section, staff space/line heights, rest/acci-
%                   dental/dot/tie/slur dataset
% Outputs arg(s):   other symbols and ledger line locations
% Description:      Detects relevant non-note symbols such as rests, aug-
%                   mentation dots, among others
%-------------------------------------------------------------------------
function [otherSyms, ledgerLineLocs] = detectOtherSymbols(section, ...
    spaceHeight, lineHeight, staffLines, combinedSyms, dotSyms, ...
    tieSlurSyms, debug)

    % Calculate the height of the entire staff
    staffHeight = max(staffLines) - min(staffLines);
    % Preliminary barline detection: Removal of lines that are almost the
    % same height as the staff
    tempErode = imerode(~section,strel('rectangle', [staffHeight-5, 2]));
    tempDilate = imdilate(tempErode,strel('rectangle', [staffHeight-5,...
        2]));
    noBarLines = xor(section,tempDilate);
    barSection = ~tempDilate;

    %-----------------------------------------------------------------
    % First pass: Detecting symbols after vertical connection
    % - Before template matching, potential symbols are connected
    %   vertically to ensure that symbols with multiple components
```

```
%   (such as fermata) will be detected properly
% - All but the tie/slur dataset will be used
%--------------------------------------------------------------
% Connect symbols with a line of lenght spaceHeight
tempClose = imclose(~noBarLines, strel('line', spaceHeight, 90));
stats = regionprops(tempClose, 'BoundingBox', 'Area');
% Call vertical symbols detection function
[otherSyms1, tempImg1] = otherSymsVert(section, stats, spaceHeight, ...
    combinedSyms, dotSyms, debug);

% Remove symbols that have been already detected from the non-barline
% image
imgNoSyms1 = ~xor(noBarLines,tempImg1);

% Perform run-length encoding on the image with only barlines to
% get maximum barline width; if there are no barlines, the default
% width is three times the lineHeight
len = encodeRL(barSection');
barWidth = len(1:2:end);
barWidth = max(barWidth(barWidth ~= 0));
if(isempty(barWidth))
    barWidth = 3*lineHeight;
end

%--------------------------------------------------------------
% Second pass: Detection of ties/slurs and other symbols missed
%     by the first pass
%--------------------------------------------------------------
% Temporarily connect the symbols horizontally for reference
tempClose2 = imclose(~imgNoSyms1, strel('line', barWidth+2, 0));
% Get and add the gaps that were produced by the barline removal
barGap = and(~barSection, tempClose2);
% Connect the symbols using the generated gaps
tempClose2 = or(~imgNoSyms1, barGap);
stats2 = regionprops(tempClose2, 'BoundingBox', 'Area');
% Call horizontal symbols detection function
[otherSyms2, tempImg2] = otherSymsHorz(section, stats2, ...
    spaceHeight, combinedSyms, dotSyms, tieSlurSyms, debug);

% Remove symbols that have already been detected from the non-barline
% image, leaving potential ledger lines
ledgerBarline = xor(imgNoSyms1,tempImg2);
cc = bwconncomp(ledgerBarline);
ledgerStats = regionprops(cc, 'Area');
% Remove bigger components that might be treated as ledger lines
% such as dynamic markings
idx = find([ledgerStats.Area] <= (20*spaceHeight));
ledgerBarline2 = ~ismember(labelmatrix(cc), idx);

% Extract ledger line locations similar to the note detections
ledgerLines = imclose(ledgerBarline2, strel('line', ...
    floor(spaceHeight * 2), 0));
lineExt = imdilate(~ledgerLines, strel('line', ...
    2*size(section,2), 0));
ledgerLineLocs = find(sum(lineExt, 2) ~= 0);


%--------------------------------------------------------------
% Third pass: Detection of barlines (single/double)
%--------------------------------------------------------------
% Temporarily connect lines horizontally for bounding box calculation
barClose = imclose(~barSection, strel('line', spaceHeight, 0));
barStats = regionprops(barClose, 'BoundingBox');
```

```
    symbol = {};
    loc = {};
    category = {};
    type = {};
    value = {};
    symIdx = {};
    symCount = 1;
    for jjj = 1:numel(barStats)
        % Extract bounding box parameters
        tempImg3 = ones(size(section));
        z3 = barStats(jjj).BoundingBox;
        h3 = ceil(z3(2))+z3(4);
        w3 = ceil(z3(1))+z3(3);
        boxHeight3 = ceil(z3(2)):h3;
        boxWidth3 = ceil(z3(1)):w3;

        % Copy section based on bounding box parametters
        tempImg3(boxHeight3, boxWidth3) = section(boxHeight3, boxWidth3);
        barStats2 = regionprops(~tempImg3);
        barlineCount = numel(barStats2);

        % Determine barline type based on the number of vertical lines
        if (barlineCount == 1 || barlineCount == 2)
            if (barlineCount == 1)
                barLineType = 'single';
            else
                barLineType = 'double';
            end

            % Each barline's parameters are saved
            symbol{symCount} = tempImg3(boxHeight3, boxWidth3);
            loc(symCount) = {[ceil(z3(2)) h3 ceil(z3(1)) w3]};
            category(symCount) = {'barline'};
            type(symCount) = {barLineType};
            value(symCount) = {0};
            symIdx(symCount) = {ceil(z3(1))};
            symCount = symCount + 1;

        end
        if (debug == 1)
            rectangle('Position',barStats(jjj).BoundingBox, ...
                'EdgeColor','g', 'LineWidth',2);
        end
    end

    % Combine parameters for complete non-note symbol definitions
    if (~isempty(symbol))
        otherSymsTmp = [symbol', loc', category', type', value', symIdx'];
        % Sort symbols based on x-axis locations
        otherSyms = sortrows([otherSyms1; otherSyms2; otherSymsTmp],6);
    end
end
```

## Vertical symbol detection

```
%-------------------------------------------------------------------------
% Function name:    otherSymsVert
% Input arg(s):     non-note section, staff space/line heights, non-note
%                   dataset except tie/slur
```

```
% Outputs arg(s):   other symbols and section with detected symbols
% Description:      Detects non-note symbols after vertical connection
%                   of components
%-------------------------------------------------------------------------
function [otherSymsArr, newSection] = otherSymsVert(section, stats, ...
    spaceHeight, combinedSyms, dotSyms, debug)

    newSection = ones(size(section));
    symbol = {};
    loc = {};
    category = {};
    type = {};
    value = {};
    symIdx = {};
    symCount = 1;

    % Goes through the image based on individual bounding boxes
    for j = 1:numel(stats)
        % Extract component boundaries
        z = stats(j).BoundingBox;
        h = ceil(z(2))+ceil(z(4));
        w = ceil(z(1))+ceil(z(3));
        % Limits the boundaries when it reaches upper/lower bounds
        if (h < 1)
            h = 1;
        elseif (w < 1)
            w = 1;
        elseif (h >= size(section,1))
            h = size(section,1)-1;
        elseif (w >= size(section,2))
            w = size(section,2)-1;
        end
        boxHeight1 = ceil(z(2)):h;
        boxWidth1 = ceil(z(1)):w;

        % Isolate component before template matching
        symTmp = section(boxHeight1, boxWidth1);

        % Check if component meets the height and width criteria
        if ((z(3) >= 3 && z(3) <= 3.5*spaceHeight) && ...
            (z(4) >= 3 && z(4) <= 6*spaceHeight))
            % Potential dots are smaller than spaceheight, and the
            % absolute difference between the length and width is 1
            if(((z(3)-z(4) == 1) || (z(3)-z(4) == -1) || ...
                (z(3)-z(4) == 0)) && (z(3) < floor(spaceHeight * 0.9) ...
                && z(4) < floor(spaceHeight * 0.9)))
                corr_val = [];
                % 2-D correlation template matching with dot dataset
                for jj = 1:length(dotSyms)
                    temp = imresize(dotSyms{jj},[length(boxHeight1),...
                        length(boxWidth1)]);
                    corr_val = [corr_val corr2(symTmp, temp)];
                end
                % If the maximum correlation coefficient is >= 0.5, dot
                % parameters are saved
                [corrPct, idx] = max(corr_val);
                if (corrPct >= 0.5)
                    symbol{symCount} = symTmp;
                    loc(symCount) = {[ceil(z(2)) h ceil(z(1)) w]};
                    category(symCount) = dotSyms(idx,2);
                    type(symCount) = dotSyms(idx,3);
                    value(symCount) = dotSyms(idx,4);
                    symIdx(symCount) = {ceil(z(1))};
```

```
                    symCount = symCount + 1;
                    if (debug == 1)
                        rectangle('Position',stats(j).BoundingBox, ...
                            'EdgeColor','#77AC30', 'LineWidth',2);
                    end
                    newSection(boxHeight1, boxWidth1) = symTmp;
                end
            % If component's length or width is bigger than spaceheight,
            % use the rest of the dataset
            else
                corr_val = [];
                % 2-D correlation template matching with combined dataset
                % (rest/accidental/fermata)
                for jj = 1:length(combinedSyms)
                    temp = imresize(combinedSyms{jj},[length(...
                        boxHeight1), length(boxWidth1)]);
                    corr_val = [corr_val corr2(symTmp, temp)];
                end
                % If the maximum correlation coefficient is >= 0.6, symbol
                % parameters are saved
                [corrPct, idx] = max(corr_val);
                if (corrPct >= 0.6)
                    newSection(boxHeight1, boxWidth1) = section(...
                        boxHeight1, boxWidth1);
                    % In the case of flat symbol, the hole is extracted
                    if (strcmp(cell2mat(combinedSyms(idx,3)), 'flat'))
                        flatImg = ones(size(section));
                        flatImg(boxHeight1, boxWidth1) = section(...
                            boxHeight1, boxWidth1);
                        flatImg = ~xor(imfill(~flatImg, 'holes'), flatImg);
                        flatStats = regionprops(flatImg, 'BoundingBox');
                        z = flatStats(1).BoundingBox;
                        stats(j).BoundingBox = z;
                        h = ceil(z(2))+z(4);
                        w = ceil(z(1))+z(3);
                        boxHeight1 = ceil(z(2)):h;
                        boxWidth1 = ceil(z(1)):w;
                        symTmp = ~flatImg(boxHeight1, boxWidth1);
                    end
                    symbol{symCount} = symTmp;
                    loc(symCount) = {[ceil(z(2)) h ceil(z(1)) w]};
                    category(symCount) = combinedSyms(idx,2);
                    type(symCount) = combinedSyms(idx,3);
                    value(symCount) = combinedSyms(idx,4);
                    symIdx(symCount) = {ceil(z(1))};
                    symCount = symCount + 1;
                    if (debug == 1)
                        rectangle('Position',stats(j).BoundingBox, ...
                            'EdgeColor','#77AC30', 'LineWidth',2);
                    end
                end
            end
        end
    end
    % If they exist, combine parameters for non-note definitions
    if (~isempty(symbol))
        otherSymsTmp = [symbol', loc', category', type', value', symIdx'];
        otherSymsArr = sortrows(otherSymsTmp,6);
    else
        otherSymsArr = [];
    end
end
```

## Horizontal symbol detection

```
%-------------------------------------------------------------------------
% Function name:    otherSymsHorz
% Input arg(s):     non-note section, staff space/line heights, rest/acci-
%                   dental/dot/tie/slur dataset
% Outputs arg(s):   other symbols and section with detected symbols
% Description:      Detects non-note symbols missed by the vertical symbol
%                   detection
%-------------------------------------------------------------------------
function [otherSymsArr, newSection] = otherSymsHorz(section, stats, ...
    spaceHeight, combinedSyms, dotSyms, tieSlurSyms, debug)

    newSection = ones(size(section));
    symbol = {};
    loc = {};
    category = {};
    type = {};
    value = {};
    symIdx = {};
    symCount = 1;

    % Goes through the image based on individual bounding boxes
    for j = 1:numel(stats)
        % Extract component boundaries
        z = stats(j).BoundingBox;
        h = ceil(z(2))+ceil(z(4));
        w = ceil(z(1))+ceil(z(3));
        % Limits the boundaries when it reaches upper/lower bounds
        if (h < 1)
            h = 1;
        elseif (w < 1)
            w = 1;
        elseif (h >= size(section,1))
            h = size(section,1)-1;
        elseif (w >= size(section,2))
            w = size(section,2)-1;
        end
        boxHeight1 = ceil(z(2)):h;
        boxWidth1 = ceil(z(1)):w;

        % Isolate component before template matching
        symTmp = section(boxHeight1, boxWidth1);
        % Checks for potential tie/slur first
        if (z(3) > 2*spaceHeight)
            corr_val = [];
            % 2-D correlation template matching with tie/slur dataset
            for jj = 1:length(tieSlurSyms)
                temp = imresize(tieSlurSyms{jj},[length(boxHeight1), ...
                    length(boxWidth1)]);
                corr_val = [corr_val corr2(symTmp, temp)];
            end
            % If the maximum correlation coefficient is >= 0.5, tie/slur
            % parameters are saved
            [corrPct, idx] = max(corr_val);
            if (corrPct >= 0.5)
                symbol{symCount} = symTmp;
                loc(symCount) = {[ceil(z(2)) h ceil(z(1)) w]};
                category(symCount) = tieSlurSyms(idx,2);
                type(symCount) = tieSlurSyms(idx,3);
                value(symCount) = tieSlurSyms(idx,4);
                symIdx(symCount) = {ceil(z(1))};
```

```
                symCount = symCount + 1;
                if (debug == 1)
                    rectangle('Position',stats(j).BoundingBox, ...
                        'EdgeColor','#77AC30', 'LineWidth',2);
                end
                newSection(boxHeight1, boxWidth1) = symTmp;
            end
        % If not detected as a tie/slur, perform detection similar to
        % vertical symbol detection
        elseif ((z(3) >= 3 && z(3) <= 3.5*spaceHeight) && ...
            (z(4) >= 3 && z(4) <= 6*spaceHeight))
            % Potential dots
            if(((z(3)-z(4) == 1) || (z(3)-z(4) == -1) || ...
                (z(3)-z(4) == 0)) && ...
                (z(3) < floor(spaceHeight * 0.9) && z(4) < ...
                    floor(spaceHeight * 0.9)))
                corr_val = [];
                % 2-D correlation template matching with dot dataset
                for jj = 1:length(dotSyms)
                    temp = imresize(dotSyms{jj},[length(boxHeight1), ...
                        length(boxWidth1)]); %z(4)+1, z(3)+1]);
                    corr_val = [corr_val corr2(symTmp, temp)];
                end
                % If the maximum correlation coefficient is >= 0.5, dot
                % parameters are saved
                [corrPct, idx] = max(corr_val);
                if (corrPct >= 0.5)
                    symbol{symCount} = symTmp;
                    loc(symCount) = {[ceil(z(2)) h ceil(z(1)) w]};
                    category(symCount) = dotSyms(idx,2);
                    type(symCount) = dotSyms(idx,3);
                    value(symCount) = dotSyms(idx,4);
                    symIdx(symCount) = {ceil(z(1))};
                    symCount = symCount + 1;
                    if (debug == 1)
                        rectangle('Position',stats(j).BoundingBox, ...
                            'EdgeColor','#77AC30', 'LineWidth',2);
                    end
                    newSection(boxHeight1, boxWidth1) = symTmp;
                end
            % Potential rest/accidental/fermata
            else
                corr_val = [];
                % 2-D correlation template matching with combined dataset
                % (rest/accidental/fermata)
                for jj = 1:length(combinedSyms)
                    temp = imresize(combinedSyms{jj},[length(...
                        boxHeight1), length(boxWidth1)]);
                    corr_val = [corr_val corr2(symTmp, temp)];
                end
                % If the maximum correlation coefficient is >= 0.6, symbol
                % parameters are saved
                [corrPct, idx] = max(corr_val);
                if (corrPct >= 0.6)
                    newSection(boxHeight1, boxWidth1) = section(...
                        boxHeight1, boxWidth1);
                    % In the case of flat symbol, the hole is extracted
                    % to be used in phase 3
                    if (strcmp(cell2mat(combinedSyms(idx,3)), 'flat'))
                        flatImg = ones(size(section));
                        flatImg(boxHeight1, boxWidth1) = section(...
                            boxHeight1, boxWidth1);
                        flatImg = ~xor(imfill(~flatImg, 'holes'), flatImg);
```

```
                        flatStats = regionprops(flatImg, 'BoundingBox');
                        z = flatStats(1).BoundingBox;
                        stats(j).BoundingBox = z;
                        h = ceil(z(2))+z(4);
                        w = ceil(z(1))+z(3);
                        boxHeight1 = ceil(z(2)):h;
                        boxWidth1 = ceil(z(1)):w;
                        symTmp = ~flatImg(boxHeight1, boxWidth1);
                    end
                    symbol{symCount} = symTmp;
                    loc(symCount) = {[ceil(z(2)) h ceil(z(1)) w]};
                    category(symCount) = combinedSyms(idx,2);
                    type(symCount) = combinedSyms(idx,3);
                    value(symCount) = combinedSyms(idx,4);
                    symIdx(symCount) = {ceil(z(1))};
                    symCount = symCount + 1;
                    if (debug == 1)
                        rectangle('Position',stats(j).BoundingBox, ...
                            'EdgeColor','#77AC30', 'LineWidth',2);
                    end
                end
            end
        end
    end

    % If they exist, combine parameters for non-note definitions
    if (~isempty(symbol))
        otherSymsTmp = [symbol', loc', category', type', value', symIdx'];
        otherSymsArr = sortrows(otherSymsTmp,6);
    else
        otherSymsArr = [];
    end
end
```

## Pitch translator

```
%--------------------------------------------------------------------------
% Function name:    pitchTranslator
% Input arg(s):     Pitch number
% Outputs arg(s):   String equivalent of pitch number
% Description:      Returns the pitch equivalent of a number based on the
%                   piano's 88 keys
%--------------------------------------------------------------------------
function pitch = pitchTranslator(pitchNumber)

    % If pitch number is 1, 2, 3, or 88, the pitch is specified
    if (pitchNumber == 1)
        pitch = 'A0';
    elseif (pitchNumber == 2)
        pitch = 'Bb0';
    elseif (pitchNumber == 3)
        pitch = 'B0';
    elseif (pitchNumber == 88)
        pitch = 'C8';
    % If pitch is between 4 and 87, the pitch assignment is generalized
    elseif ((pitchNumber > 3) && (pitchNumber < 88))
        % Returns the octave where the pitch belongs to
        octave = (fix((pitchNumber - 4) / 12) + 1);
        % Returns the pitch based on modulo-12
        temp = mod((pitchNumber - 4), 12);
```

```
        if (temp == 0)
            note = 'C';
        elseif (temp == 1)
            note = 'Cs';
        elseif (temp == 2)
            note = 'D';
        elseif (temp == 3)
            note = 'Eb';
        elseif (temp == 4)
            note = 'E';
        elseif (temp == 5)
            note = 'F';
        elseif (temp == 6)
            note = 'Fs';
        elseif (temp == 7)
            note = 'G';
        elseif (temp == 8)
            note = 'Gs';
        elseif (temp == 9)
            note = 'A';
        elseif (temp == 10)
            note = 'Bb';
        elseif (temp == 11)
            note = 'B';
        end
        % Concatenate initial pitch and octave to produce the
        % complete pitch
        pitch = strcat(note, num2str(octave));
    % If number is out of bounds, set pitch to 0
    else
        pitch = '0';
    end
end
```

## Initial staff pitch assignment

```
%--------------------------------------------------------------------------
% Function name:    staffPitchAssignment
% Input arg(s):     staff and ledger centers, clef parameters, note
%                   reference array
% Outputs arg(s):   Reference values for final pitch assignment
% Description:      Assigns the initial staff pitch/note based on clef
%                   attached to the given staff
%--------------------------------------------------------------------------
function [ledgerStaffSpace, pitch, note] = staffPitchAssignment(centers,...
    staffCenters, clefs, clefIdx, noteRefArr)

    pitch = [];
    note = [];
    % Calculate staff and ledger line midpoints
    tempCentroidDiff = centers(1:end-1) + (diff(centers/2));
    % In addition, spaces on top and bottom of staff are assigned values
    centroidDiff = [centers(1) - (diff(tempCentroidDiff(1:2))/2);...
        tempCentroidDiff; centers(end) + ...
        (diff(tempCentroidDiff(end-1:end))/2)];
    % Complete reference points are the combination of the above points
    ledgerStaffSpace = sort([centers; centroidDiff]);

    % Length of reference points
    ledgerStaffSpaceSize = length(ledgerStaffSpace);
```

```
    % Returns the reference note value based on clef
    staffVal = cell2mat(clefs(clefIdx,5));
    % Staff line number based on clef
    staffLineVal = cell2mat(clefs(clefIdx,6));
    % Returns the note reference index where the note is equal the
    % staff line note value
    refPitchIdx = find(strcmp(noteRefArr(:,2), staffVal));
    % Returns the index where ledger/staff/space line is equal the
    % staffline specified in the clef parameters
    refLineIdx = find(ledgerStaffSpace == staffCenters(staffLineVal));
    % Returns the starting index for pitch/note assignment
    offset = refPitchIdx - refLineIdx + 1;
    for ii = 1:ledgerStaffSpaceSize
        pitch = [pitch; str2double(noteRefArr(offset, 1))];
        note = [note; noteRefArr(offset, 2)];
        offset = offset+1;
    end
end
```

### Final staff pitch assignment

```
%-------------------------------------------------------------------------
% Function name:    finalStaffPitchAssignment
% Input arg(s):     Sheet music section, staff/ledger line locations,
%                   clef and key signature parameters
% Outputs arg(s):   Staff/ledger/space locations, current/pre-key sig-
%                   nature pitches and notes
% Description:      Assigns the final pitches and notes of a given staff
%                   based on key signature
%-------------------------------------------------------------------------
function [ledgerStaffSpace, pitch, note, pitchPreKeySig, staffCenters, ...
    keyType, keySigIdx, staffDivider] = finalStaffPitchAssignment(...
    section, staffLines, ledgerLineLocs, clefs, keys)

    %-----------------------------------------------------------------
    % Part 1: Reference centroid calculations (staff/ledger lines)
    %-----------------------------------------------------------------
    % Stafflines are placed in a blank image, then centroids are
    % calculated
    staffCopy = ones(size(section));
    staffCopy(staffLines,10:end-10) = 0;
    staffRefPts = regionprops(~staffCopy, 'Centroid');
    staffCenters = cat(1,staffRefPts.Centroid);
    staffCenters = staffCenters(:,2);
    % Ledger lines are placed in a blank image, then centroids are
    % calculated
    ledgerCopy = ones(size(section));
    ledgerCopy(ledgerLineLocs,10:end-10) = 0;
    ledgerRefPts = regionprops(~ledgerCopy, 'Centroid');
    ledgerCenters = cat(1,ledgerRefPts.Centroid);
    % If ledger lines exist, take the y-component of the centroids
    if (~isempty(ledgerCenters))
        ledgerCenters = ledgerCenters(:,2);
    end
    % Sort the concatenated staff and ledger centroids
    centers = sort([staffCenters; ledgerCenters]);

    %-----------------------------------------------------------------
    % Part 2: Generate reference note array without sharps or flats
    %-----------------------------------------------------------------
```

```
    noteRefArr = [];
    % Higher pitches are set on top, therefore, the reference array
    % is in reverse order
    for ii = 88:-1:1
        [note] = pitchTranslator(ii);
        % Notes with sharps or flats have a length of 3
        if (length(note) ~= 2)
            continue;
        else
            noteRefArr = [noteRefArr; [ii, string(note)]];
        end
    end

    %-----------------------------------------------------------------
    % Part 3: Assigning overall pitches based on staff type
    %-----------------------------------------------------------------
    % If staffCenters == 5, the staff type is solo
    if (length(staffCenters) == 5)
        staffDivider = size(section,1);
        symbolLocs = cell2mat(clefs(:,2));
        % Checks for the clef present in the staff
        clefIdx = find(symbolLocs(:,2) < staffDivider);
        % Assigns preliminary pitch based on clef only
        [ledgerStaffSpace, pitch, note] = staffPitchAssignment(centers, ...
            staffCenters, clefs, clefIdx, noteRefArr);
        pitchPreKeySig = pitch;

        % Assigns final key based on key signature
        if(~isempty(keys))
            % Check the location and type of keys
            keyType = string(cell2mat(keys(1,4)));
            keySigLocs = cell2mat(keys(:,2));
            visited = [];
            for j = 1:size(keys, 1)
                % Key centroid calculation
                h1 = keySigLocs(j,1);
                h2 = keySigLocs(j,2);
                cent_y = mean([h1, h2]);
                % Returns the staff/space location closest to the key
                % centroid
                [~, idx] = min(abs(ledgerStaffSpace - cent_y));
                noteTemp = char(note);
                % Ensures that all instances of the note is adjusted
                % basd on key
                tempNote = find((noteTemp(:,1) == noteTemp(idx,1)));
                tempNote = tempNote(~ismember(tempNote, visited));
                visited = sort([visited; tempNote]);
                for k = 1:length(tempNote)
                    pitch(tempNote(k)) = pitch(tempNote(k))...
                        + cell2mat(keys(j,5));
                    note(tempNote(k)) = pitchTranslator(pitch(tempNote(k)));
                end
            end
            % Ensures that after a pitch was modified once, it will not be
            % modified further
            keySigIdx = visited;
        else
            keyType = [];
            keySigIdx = [];
        end
        staffDivider = [];
    % If staffCenters ~= 5, the staff type is grand
    else
```

```
        % Calculate center divider between two staves
        staffDivider = (staffCenters(5)+staffCenters(6))/2;

        % Top staff preliminary pitch assignment clef
        symbolLocs = cell2mat(clefs(:,2));
        clefIdx1 = find(symbolLocs(:,2) < staffDivider);
        grandStaffCenters1 = centers(centers < staffDivider);
        staffCenters1 = staffCenters(1:5);
        [ledgerStaffSpace1, pitch1, note1] = staffPitchAssignment(...
            grandStaffCenters1, staffCenters1, clefs, clefIdx1, noteRefArr);

        % Bottom staff preliminary pitch assignment based on clef
        clefIdx2 = find(symbolLocs(:,2) > staffDivider);
        grandStaffCenters2 = centers(centers > staffDivider);
        staffCenters2 = staffCenters(6:10);
        [ledgerStaffSpace2, pitch2, note2] = staffPitchAssignment(...
            grandStaffCenters2, staffCenters2, clefs, clefIdx2, noteRefArr);

        % Initial pitches and ledger/staff lines and spaces of both
        % staves are combined
        pitchPreKeySig = [pitch1 ; pitch2];
        ledgerStaffSpace = [ledgerStaffSpace1; ledgerStaffSpace2];

        % Assigns final key based on key signature0
        if(~isempty(keys))
            % Check the location and type of keys
            keyType = string(cell2mat(keys(1,4)));
            keySigLocs = cell2mat(keys(:,2));
            visited1 = [];
            visited2 = [];
            for j = 1:size(keys, 1)
                % Key centroid calculation
                h1 = keySigLocs(j,1);
                h2 = keySigLocs(j,2);
                cent_y = mean([h1, h2]);

                % Final pitch assignment for top staff
                if (cent_y < staffDivider)
                    % Returns the staff/space location closest to the key
                    % centroid
                    [~, idx] = min(abs(ledgerStaffSpace1 - cent_y));
                    noteTemp1 = char(note1);
                    % Ensures that all instances of the note is adjusted
                    % basd on key
                    tempNote1 = find((noteTemp1(:,1) == noteTemp1(idx,1)));
                    tempNote1 = tempNote1(~ismember(tempNote1, visited1));
                    visited1 = sort([visited1; tempNote1]);
                    for k = 1:length(tempNote1)
                        pitch1(tempNote1(k)) = pitch1(tempNote1(k))...
                            + cell2mat(keys(j,5));
                        note1(tempNote1(k)) = pitchTranslator(pitch1(...
                            tempNote1(k)));
                    end
                % Final pitch assignment for bottom staff
                else
                    % Returns the staff/space location closest to the key
                    % centroid
                    [~, idx] = min(abs(ledgerStaffSpace2 - cent_y));
                    noteTemp2 = char(note2);
                    % Ensures that all instances of the note is adjusted
                    % basd on key
                    tempNote2 = find((noteTemp2(:,1) == noteTemp2(idx,1)));
                    tempNote2 = tempNote2(~ismember(tempNote2, visited2));
```

```
                    visited2 = sort([visited2; tempNote2]);
                    for k = 1:length(tempNote2)
                        pitch2(tempNote2(k)) = pitch2(tempNote2(k))...
                            + cell2mat(keys(j,5));
                        note2(tempNote2(k)) = pitchTranslator(pitch2(...
                            tempNote2(k)));
                    end
                end
            end
            % Ensures that after a pitch was modified once, it will not be
            % modified further
            keySigIdx = [visited1; visited2];
        else
            keyType = [];
            keySigIdx = [];
        end
        % Combine the pitch and note from both staves to generate complete
        % pitch/note arrays
        pitch = [pitch1; pitch2];
        note = [note1; note2];
    end
end
```

## Solo sheet music generator

```
%-------------------------------------------------------------------------
% Function name:    SoloSheetMusicGenerator
% Input arg(s):     ledger/staff/space locations, pre/post-key pitches and
%                   notes, total symbol list, space/line heights, time and
%                   key signature array, note/beat file IDs
% Description:      Produces a note and beat array based on the recon-
%                   struction of the musical symbols on a solo section
%-------------------------------------------------------------------------
function SoloSheetMusicGenerator(totalSyms, totalTimeSignature, ...
    ledgerStaffSpace, pitch, note, pitchPreKS, staffCenters, keyType, ...
    keySigIdx, spaceHeight, lineHeight, fid1, fid2)

    % Generate reference indices for subsequent symbol locations
    totalSymsIdx = 1:length(totalSyms);
    totalSymType = string(totalSyms(:,3));
    % Get indices of note/rest/barline symbols
    noteRestBarIdx = find(strcmp(totalSymType,"note") | ...
         strcmp(totalSymType,"rest") | ...
         strcmp(totalSymType,"barline"));
    % Get indices of symbols that are not note/rest/barline
    nonNoteIdx = totalSymsIdx(~ismember(totalSymsIdx, noteRestBarIdx));
    nonNoteSyms = totalSyms(nonNoteIdx,:);

    % Calculate reference centroids
    symbolLocs = cell2mat(totalSyms(:,2));
    h1 = symbolLocs(:,1);
    h2 = symbolLocs(:,2);
    w1 = symbolLocs(:,3);
    w2 = symbolLocs(:,4);
    cent_x = (w1+w2)./2;
    cent_y = (h1+h2)./2;

    % Modifiable buffers for pitch and note
    measurePitch = pitch;
    measureNote = note;
```

```
% Array initializers
tiePresent = [];
visited = [];
totalNotes = [];

% Process individual symbol detected by phase 2
for j = 1:size(totalSyms, 1)
    % Performs a type check if the symbol is either a note, rest,
    % or barline
    isNote = strcmp(cell2mat(totalSyms(j,3)),'note');
    isRest = strcmp(cell2mat(totalSyms(j,3)),'rest');
    isBar = strcmp(cell2mat(totalSyms(j,3)),'barline');

    % If current symbol has already been processed, skip processing
    if(ismember(j, sort(visited)))
        continue;
    else
        % In the case of solo sheets, a maximum of three notes can
        % be played simultaneously
        noteBuf = string(zeros(1,3));
        % Add current symbol index to visited array
        visited = [visited; j];
        %-------------------------------------------------------
        % Note/rest check: Assigns pitch(es) and beat based on
        % adjacencies (other notes/dot/accidental/fermata)
        %-------------------------------------------------------
        if (isNote || isRest)
            % Calculate location of staff/ledger/space closest to
            % symbol centroid
            [~, idx] = min(abs(ledgerStaffSpace - cent_y(j)));
            if (isNote)
                % If symbol is a note, temporary pitch/note is based
                % on buffer values, while beat is based on default
                tempPitch = measurePitch(idx);
                tempNote = measureNote(idx);
                tempBeat = cell2mat(totalSyms(j,5));
            elseif (isRest)
                % If symbol is a rest, pitch is set to 0,
                tempPitch = 0;
                tempNote = string(pitchTranslator(tempPitch));
                % In the case of whole/half rest, beat value is
                % assigned based on location
                if (strcmp(string(totalSyms(j,4)), "wholehalf"))
                    [~, idx2] = min(abs(staffCenters - cent_y(j)));
                    % If the location is the third staffline, the rest
                    % is a half rest
                    if (idx2 == 3)
                        tempBeat = 0.5;
                    % On the other hand, if location is second staff-
                    % line, the rest if a whole rest
                    elseif (idx2 == 2)
                        temp = strsplit(string(totalTimeSignature(1,4)),'/');
                        tempBeat = str2double(temp(1))/str2double(temp(2));
                    end
                % For the rest of the staff lines, use the default
                % value
                else
                    tempBeat = cell2mat(totalSyms(j,5));
                end
            end

            %-------------------------------------------------------
```

```
                % Accidental locator and pitch re-assignment
                % - Accidentals are typically located on the left-hand
                %   side of notes
                % - Modifies the note's pitch based on conditions
                %-------------------------------------------------------
                % Find possible accidental on the note's LHS
                accidentalIdx = find((...
                    abs(cent_x(nonNoteIdx) - cent_x(j)) < 4*spaceHeight) & ...
                    (abs(cent_y(nonNoteIdx) - cent_y(j)) < 7*lineHeight) & ...
                    (cent_x(nonNoteIdx) < cent_x(j)));
                % Checks if the symbol located is indeed an accidental
                if (strcmp(string(nonNoteSyms(accidentalIdx,3)), ...
                        "accidental"))
                    % Returns the index of the staff/space/ledger
                    % centroid closest to the accidental
                    [~, tempIdx] = min(abs(ledgerStaffSpace - cent_y(...
                        nonNoteIdx(accidentalIdx))));
                    % Natural accidental case: its value depends on current
                    % line pitch
                    if (strcmp(string(nonNoteSyms(accidentalIdx, 4)), ...
                        "natural"))
                        % if natural is found in a line with a sharp key,
                        % or buffered pitch is higher than original pitch,
                        % current pitch will be decreased by 1
                        if ((ismember(tempIdx, keySigIdx) && strcmp(...
                            keyType, "sharp")) || (measurePitch(...
                            tempIdx) > pitchPreKS(tempIdx)))
                            tempPitch = tempPitch - 1;
                        % if natural is found in a line with a flat key,
                        % or buffered pitch is lower than original pitch,
                        % current pitch will be increased by 1
                        elseif ((ismember(tempIdx, keySigIdx) && strcmp(...
                            keyType, "flat")) || (measurePitch(...
                            tempIdx) < pitchPreKS(tempIdx)))
                            tempPitch = tempPitch + 1;
                        end
                    % For flat/sharp, use default value
                    else
                        tempPitch = tempPitch + cell2mat(nonNoteSyms(...
                            accidentalIdx,5));
                    end
                    % Pitch/note buffers will be modified based on changes
                    % made by accidentals
                    measurePitch(tempIdx) = tempPitch;
                    tempNote = string(pitchTranslator(tempPitch));
                    measureNote(tempIdx) = tempNote;
                end
                %-------------------------------------------------------
                % Accent dot locator and beat re-assignment
                % - These are dots that are located on the right-hand
                %   side of notes/rests
                % - If found, increases the beat value by a factor of
                %   1.5
                %-------------------------------------------------------
                % First pass: Checks for symbols meeting the location
                %     requirements
                accentDotIdx = find(...
                    (abs(cent_x(nonNoteIdx) - cent_x(j)) <= 2.5*spaceHeight) & ...
                    (cent_y(j) - cent_y(nonNoteIdx) <= 0.75*spaceHeight) & ...
                    (cent_y(j) - cent_y(nonNoteIdx) >= -3*lineHeight) & ...
                    (cent_x(nonNoteIdx) > cent_x(j)));
                % Second pass: Checks if the located symbol is a dot
                accentDotIdx = find(strcmp(string(nonNoteSyms(accentDotIdx,3)),...
```

```
    "dot"));

%-------------------------------------------------------
% Vertical symbol locator:
% - Locates symbols that are on top or bottom of the
%   current symbol, such as other notes, staccato
%   dots, fermatas, or edges of ties/slurs
%-------------------------------------------------------
% First pass: Check for any vertical symbols regardless
%     of type
verticalSyms = find((cent_y(totalSymsIdx) ~= cent_y(j)) & ...
    (abs(cent_x(j) - cent_x(totalSymsIdx)) <= 3*lineHeight));

% Fermata case: Locate symbols matching the type
fermataIdx = find(strcmp(string(totalSyms(verticalSyms,4)),...
    "fermata"));

% Second pass: Check for any vertical symbols that are
%     close to the x-axis of the current note
verticalSyms2 = find(...
    (abs(cent_y(nonNoteIdx) - cent_y(j)) <= 3*spaceHeight) & ...
    (abs(cent_x(j) - cent_x(nonNoteIdx)) <= 3*lineHeight));
% Staccato dot case: Locate symbols matching description
staccatoIdx = find(strcmp(string(nonNoteSyms(verticalSyms2,4)),...
    "augmentation"));


%-------------------------------------------------------
% Tie/slur locator and 'note break' assignment
% - Note break is a parameter which determines if
%   a note/s should be played without breaks
%-------------------------------------------------------
% First pass: checks for left or right edges which are
%     found above or below a note
tieSlurStart = find(abs(w1(nonNoteIdx) - cent_x(j)) < ...
    0.5*spaceHeight);
tieSlurEnd = find(abs(w2(nonNoteIdx) - cent_x(j)) < ...
    0.5*spaceHeight);
% Second pass: check if symbol type is a tie or a slur
tieSlurStartIdx = find(strcmp(string(nonNoteSyms(...
    tieSlurStart,4)), "tie") | strcmp(string(totalSyms(...
    tieSlurStart,4)), "slur"));
tieSlurEndIdx = isempty(find(strcmp(string(nonNoteSyms(...
    tieSlurEnd,4)), "tie") | strcmp(string(totalSyms(...
    tieSlurEnd,4)), "slur"),1));

% If a tie/slur starting edge is detected, note break is
% set to 0
if (~isempty(tieSlurStartIdx))
    tiePresent = [tiePresent; tieSlurStartIdx];
    noteBreak = 0;
% If end edge is detected, add a note break (set to 1)
elseif(~isempty(tieSlurEndIdx) && ~isempty(tiePresent))
    tiePresent = [];
    noteBreak = 1;
% If note is in between tie edges, note break is also
% set to 0
elseif (~isempty(tiePresent) && (cent_x(j) > tiePresent(1)))
    noteBreak = 0;
% If above conditions are not met, set note break to 1
else
    noteBreak = 1;
end
```

```
% Third pass: Check for any vertical symbols that are notes
%     (maximum of 2)
doubleStops = verticalSyms(strcmp(string(totalSyms(...
    verticalSyms,3)), "note"));
% If they are detected, add indices to the visited array
visited = [visited; doubleStops];

% Process each double/triple stop note similar to previous
% note
if (~isempty(doubleStops))
    % Add previous note to total notes array
    totalNotes = [totalNotes; tempNote];

    for jj = 1:length(doubleStops)
        % Calculate location of staff/ledger/space closest to
        % symbol centroid
        [~, idx] = min(abs(ledgerStaffSpace - cent_y(...
            doubleStops(jj))));

        % Pitch/note assignment based on buffers
        tempPitch = measurePitch(idx);
        tempNote = measureNote(idx);

        % Accidental locator
        accidentalIdx = find((abs(cent_x(nonNoteIdx) - ...
            cent_x(doubleStops(jj))) < 4*spaceHeight) & ...
            (abs(cent_y(nonNoteIdx) - cent_y(doubleStops(...
            jj))) < 7*lineHeight) & (cent_x(nonNoteIdx) <...
            cent_x(doubleStops(jj))));
        if (strcmp(string(nonNoteSyms(accidentalIdx,3)), ...
                "accidental"))
            [~, tempIdx] = min(abs(ledgerStaffSpace - ...
                cent_y(nonNoteIdx(accidentalIdx))));
            if (strcmp(string(nonNoteSyms(accidentalIdx,...
                4)), "natural"))
                if ((ismember(tempIdx, keySigIdx) && ...
                    strcmp(keyType, "sharp")) || ...
                    (measurePitch(tempIdx) > pitchPreKS(...
                    tempIdx)))
                    tempPitch = tempPitch - 1;
                elseif ((ismember(tempIdx, keySigIdx) &&...
                    strcmp(keyType, "flat")) || ...
                    (measurePitch(tempIdx) < pitchPreKS(...
                    tempIdx)))
                    tempPitch = tempPitch + 1;
                end
            else
                tempPitch = tempPitch + cell2mat(...
                    nonNoteSyms(accidentalIdx,5));
            end
            measurePitch(tempIdx) = tempPitch;
            tempNote = string(pitchTranslator(tempPitch));
            measureNote(tempIdx) = tempNote;
        end
        totalNotes = [totalNotes; tempNote];

        % Vertical symbol locator
        verticalSyms2 = find((abs(cent_y(nonNoteIdx) - ...
            cent_y(doubleStops(jj))) <= 3*spaceHeight) & ...
            (abs(cent_x(j) - cent_x(nonNoteIdx)) <= ...
            3*lineHeight));
        % Staccato dot locator
```

```
                    staccatoIdx2 = find(strcmp(string(nonNoteSyms(...
                        verticalSyms2,4)), "augmentation"));
                    % Fermata locator
                    fermataIdx = find(strcmp(string(totalSyms(...
                        verticalSyms,4)), "fermata"));
                end

                %-----------------------------------------
                % Final beat assignment based on exisiting
                %     symbols
                %-----------------------------------------
                % If accent dot exists, original beat is increase by
                % a factor of 1.5
                if (~isempty(accentDotIdx))
                    tempBeat = tempBeat * 1.5;
                % If accent dot doesn't exist and staccato dot exist,
                % original beat is deacreased in half
                elseif (~isempty(staccatoIdx) || ~isempty(staccatoIdx2))
                    tempBeat = tempBeat * 0.5;
                % If fermata exists, add 4 beats in the original beat
                elseif (~isempty(fermataIdx))
                    tempBeat = tempBeat * cell2mat(nonNoteSyms(...
                        fermataIdx,5));
                end

                %-----------------------------------------------------
                % File writing procedure: THe format is as
                % follows:
                % - {note1, note2, note3, 0, 0, 0, 0, 0, note break}
                % - {..., beat, ...}
                %-----------------------------------------------------
                noteBuf(1:length(totalNotes)) = totalNotes;
                fprintf(fid1,'\t{%s, %s, %s, 0, 0, 0, 0, 0, %d},\n',...
                    noteBuf, noteBreak);
                fprintf(fid2,'%.3f, ', tempBeat);
                % If fermata is detected, silence is added afterwards,
                % with the beat equiavalent to previous beat
                if (~isempty(staccatoIdx))
                    fprintf(fid1,'\t{0, 0, 0, 0, 0, 0, 0, 0, %d},\n', ...
                        noteBreak);
                    fprintf(fid2,'%.3f, ', tempBeat);
                end
                totalNotes = [];
            else
                %-----------------------------------------
                % Final beat assignment based on exisiting
                %     symbols
                %-----------------------------------------
                % If staccato exists, cut the original beat in half
                if (~isempty(staccatoIdx))
                    tempBeat = tempBeat * 0.5;
                % If accent dot exists, beat increases by a factor
                % of 1.5
                elseif (~isempty(accentDotIdx))
                    tempBeat = tempBeat * 1.5;
                % IF fermata exists, add 4 beats in the original beat
                elseif (~isempty(fermataIdx))
                    tempBeat = tempBeat * cell2mat(nonNoteSyms(...
                        fermataIdx,5));
                end

                %-----------------------------------------------------
                % File writing procedure: THe format is as
```

```
                    % follows:
                    % - {note, 0, 0, 0, 0, 0, 0, 0, note break}
                    % - {..., beat, ...}
                    %-------------------------------------------------------
                    fprintf(fid1,'\t{%s, 0, 0, 0, 0, 0, 0, 0, %d},\n',...
                        tempNote, noteBreak);
                    fprintf(fid2,'%.3f, ', tempBeat);
                    % If fermata is detected, silence is added afterwards,
                    % with the beat equiavalent to previous beat
                    if (~isempty(staccatoIdx))
                        fprintf(fid1,'\t{0, 0, 0, 0, 0, 0, 0, 0, %d},\n',...
                            noteBreak);
                        fprintf(fid2,'%.3f, ', tempBeat);
                    end
                end
            %-------------------------------------------
            % Barline symbol check
            %-------------------------------------------
            elseif(isBar)
                % If barline is found, reset the pitch and note buffers
                measurePitch = pitch;
                measureNote = note;
                % Add new lines to both note and beat input files
                fprintf(fid1, '\n');
                fprintf(fid2, '\n\t');
            end
        end
    end
end
```

### Piano sheet music generator

```
%-------------------------------------------------------------------------
% Function name:    PianoSheetMusicGenerator
% Input arg(s):     ledger/staff/space locations, pre/post-key pitches and
%                   notes, total symbol list, space/line heights, time and
%                   key signature array, note/beat file IDs
% Description:      Produces a note and beat array based on the recon-
%                   struction of the musical symbols on a piano section
%-------------------------------------------------------------------------
function PianoSheetMusicGenerator(totalSyms, totalTimeSignature, ...
    ledgerStaffSpace, pitch, note, pitchPreKS, staffCenters, keyType, ...
    keySigIdx, spaceHeight, lineHeight, fid1, fid2)


    % Generate reference indices for subsequent symbol locations
    totalSymsIdx = 1:length(totalSyms);
    symbolLocs = cell2mat(totalSyms(:,2));
    symbolCat = string(totalSyms(:,3));
    % Get indices of note/rest/barline symbols
    noteRestBarIdx = find(strcmp(symbolCat,"note") | ...
        strcmp(symbolCat,"rest") | ...
        strcmp(symbolCat,"barline"));
    noteRestBars = totalSyms(noteRestBarIdx,:);
    % Get indices of symbols that are not note/rest/barline
    nonNoteIdx = totalSymsIdx(~ismember(totalSymsIdx, noteRestBarIdx));
    nonNoteSyms = totalSyms(nonNoteIdx,:);

    % Calculate reference centroids
    h1 = symbolLocs(:,1);
```

```
    h2 = symbolLocs(:,2);
    w1 = symbolLocs(:,3);
    w2 = symbolLocs(:,4);
    cent_x = (w1+w2)./2;
    cent_y = (h1+h2)./2;

    %--------------------------------------------------------------------
    % Preprocessing: single whole rest in a measure is removed from
    % note/rest assignment; if two whole rests in one measure, keep the
    % first one and remove the second one
    %--------------------------------------------------------------------
    wholeCount = 0;
    wholeArr = [];
    visited = [];
    for ii = 1:length(noteRestBarIdx)
        if (strcmp(cell2mat(noteRestBars(ii,4)),'wholehalf'))
            [~, wholeHalfIdx] = min(abs(staffCenters - cent_y(...
                noteRestBarIdx(ii))));
            if (wholeHalfIdx == 2 || wholeHalfIdx == 7)
                if (isempty(totalTimeSignature))
                    totalSyms(noteRestBarIdx(ii),5) = {1};
                else
                    temp = strsplit(string(totalTimeSignature(1,4)),'/');
                    temp2 = str2double(temp(1))/str2double(temp(2));
                    totalSyms(noteRestBarIdx(ii),5) = {temp2};
                end
                wholeCount = wholeCount + 1;
                wholeArr = [wholeArr; noteRestBarIdx(ii)];
            elseif (wholeHalfIdx == 3 || wholeHalfIdx == 8)
                totalSyms(noteRestBarIdx(ii),5) = {0.5};
            end
        elseif (strcmp(cell2mat(noteRestBars(ii,3)),'barline'))
            if (wholeCount == 1)
                visited = [visited; wholeArr(1)];
            elseif (wholeCount == 2)
                visited = [visited; wholeArr(2)];
            end
            wholeCount = 0;
            wholeArr = [];
        end
    end

    % Modifiable buffers for pitch and note
    measurePitch = pitch;
    measureNote = note;

    % Array initializers
    tiePresent = [];
    totalNotes = [];
    totalBeat = [];

    % Process individual symbol detected by phase 2
    for j = 1:size(totalSyms, 1)
        % Performs a type check if the symbol is either a note, rest,
        % or barline
        isNote = strcmp(cell2mat(totalSyms(j,3)),'note');
        isRest = strcmp(cell2mat(totalSyms(j,3)),'rest');
        isBar = strcmp(cell2mat(totalSyms(j,3)),'barline');

        % If current symbol has already been processed, skip processing
        if(ismember(j, sort(visited)))
            continue;
        else
```

```
            % In the case of piano sheets, a maximum of eight notes can
            % be played simultaneously
            noteBuf = string(zeros(1,8));
            % Add current symbol index to visited array
            visited = [visited; j];
            if (isNote || isRest)
                % Calculate location of staff/ledger/space closest to
                % symbol centroid
                [~, idx] = min(abs(ledgerStaffSpace - cent_y(j)));
                if (isNote)
                    % If symbol is a note, temporary pitch/note is based
                    % on buffer values, while beat is based on default
                    tempPitch = measurePitch(idx);
                    tempNote = measureNote(idx);
                elseif (isRest)
                    % If symbol is a rest, pitch is set to 0,
                    tempPitch = 0;
                    tempNote = string(pitchTranslator(tempPitch));
                end
                % Set beat to default/assigned value
                tempBeat = cell2mat(totalSyms(j,5));


                %-------------------------------------------------------
                % Accidental locator and pitch re-assignment
                % - Accidentals are typically located on the left-hand
                %   side of notes
                % - Modifies the note's pitch based on conditions
                %-------------------------------------------------------
                % Find possible accidental on the note's LHS
                accidentalIdx = find((...
                    abs(cent_x(nonNoteIdx) - cent_x(j)) < 5*spaceHeight) & ...
                    (abs(cent_y(nonNoteIdx) - cent_y(j)) < 7*lineHeight) & ...
                    (cent_x(nonNoteIdx) < cent_x(j)));
                % Ensures that the located symbol is indeed an
                % accidental
                accidentalIdx = accidentalIdx(find(strcmp(string(...
                    nonNoteSyms(accidentalIdx,3)), "accidental")));
                if (~isempty(accidentalIdx))
                    % Returns the index of the staff/space/ledger
                    % centroid closest to the accidental
                    [~, tempIdx] = min(abs(ledgerStaffSpace - cent_y(...
                        nonNoteIdx(accidentalIdx))));
                    % Natural accidental case: its value depends on current
                    % line pitch
                    if (strcmp(string(nonNoteSyms(accidentalIdx, 4)),...
                            "natural"))
                        % if natural is found in a line with a sharp key,
                        % or buffered pitch is higher than original pitch,
                        % current pitch will be decreased by 1
                        if ((ismember(tempIdx, keySigIdx) && strcmp(...
                            keyType, "sharp")) || (measurePitch(...
                            tempIdx) > pitchPreKS(tempIdx)))
                            tempPitch = tempPitch - 1;
                        % if natural is found in a line with a flat key,
                        % or buffered pitch is lower than original pitch,
                        % current pitch will be increased by 1
                        elseif ((ismember(tempIdx, keySigIdx) && strcmp(...
                            keyType, "flat")) || (measurePitch(...
                            tempIdx) < pitchPreKS(tempIdx)))
                            tempPitch = tempPitch + 1;
                        end
                    % For flat/sharp, use default value
```

```
        else
            tempPitch = tempPitch + cell2mat(nonNoteSyms(...
                accidentalIdx,5));
        end
        % Pitch/note buffers will be modified based on changes
        % made by accidentals
        measurePitch(tempIdx) = tempPitch;
        tempNote = string(pitchTranslator(tempPitch));
        measureNote(tempIdx) = tempNote;
    end
    %-------------------------------------------------------
    % Accent dot locator and beat re-assignment
    % - These are dots that are located on the right-hand
    %   side of notes/rests
    % - If found, increases the beat value by a factor of
    %   1.5
    %-------------------------------------------------------
    % First pass: Checks for symbols meeting the location
    %     requirements
    accentDotIdx = find((...
        abs(cent_x(nonNoteIdx) - cent_x(j)) <= 2.5*spaceHeight) & ...
        (cent_y(j) - cent_y(nonNoteIdx) <= 0.75*spaceHeight) & ...
        (cent_y(j) - cent_y(nonNoteIdx) >= -0.25*spaceHeight) & ...
        (cent_x(nonNoteIdx) > cent_x(j)));
    % Second pass: Checks if the located symbol is a dot
    accentDotIdx = find(strcmp(string(nonNoteSyms(accentDotIdx,3)),...
        "dot"));
    % If accent dot exists, increase current note/rest's
    % beat value by 1.5
    if (~isempty(accentDotIdx))
        tempBeat = tempBeat * 1.5;
    end


    %-------------------------------------------------------
    % Vertical symbol locator:
    % - Compared to the solo sheets, only other notes and
    %   rests are considered as vertical symbols
    %-------------------------------------------------------
    verticalSyms = find((cent_y(totalSymsIdx) ~= cent_y(j)) & ...
        (abs(cent_x(j) - cent_x(totalSymsIdx)) <= 2*spaceHeight));
    chordNoteIdx = verticalSyms(strcmp(string(totalSyms(...
        verticalSyms,3)), "note") | strcmp(string(totalSyms(...
        verticalSyms,3)), "rest"));
    % If the vertical note/chord has been already visited,
    % the index is cleared
    chordNoteIdx(ismember(chordNoteIdx, visited)) = [];

    %-------------------------------------------------------
    % Tie/slur locator and 'note break' assignment
    % - Note break is a parameter which determines if
    %   a note/s should be played without breaks
    %-------------------------------------------------------
    % First pass: checks for left or right edges which are
    %     found above or below a note
    tieSlurStart = find(abs(w1(nonNoteIdx) - cent_x(j)) < ...
        0.5*spaceHeight);
    tieSlurEnd = find(abs(w2(nonNoteIdx) - cent_x(j)) < ...
        0.5*spaceHeight);
    % Second pass: check if symbol type is a tie or a slur
    tieSlurStartIdx = find(strcmp(string(nonNoteSyms(...
        tieSlurStart,4)), "tie") | strcmp(string(totalSyms(...
        tieSlurStart,4)), "slur"));
```

```
                tieSlurEndIdx = isempty(find(strcmp(string(nonNoteSyms(...
                    tieSlurEnd,4)), "tie") | strcmp(string(totalSyms(...
                    tieSlurEnd,4)), "slur"),1));

                % If it exists, process each note/rest similar to current
                % symbol
                if (~isempty(chordNoteIdx))
                    % Add previous note to total notes array
                    totalNotes = [totalNotes; tempNote];
                    totalBeat = [totalBeat; tempBeat];

                    for jj = 1:length(chordNoteIdx)
                        % Add current note to visited array
                        visited = [visited; chordNoteIdx(jj)];
                        % Type check (note/rest)
                        isNote2 = strcmp(string(totalSyms(chordNoteIdx(...
                            jj),3)),'note');
                        isRest2 = strcmp(string(totalSyms(chordNoteIdx(...
                            jj),3)),'rest');
                        % Calculate location of staff/ledger/space closest to
                        % symbol centroid
                        [~, idx] = min(abs(ledgerStaffSpace - cent_y(...
                            chordNoteIdx(jj))));
                        % If symbol is note, buffered pitch/note and beat
                        % values are used
                        if (isNote2)
                            tempPitch = measurePitch(idx);
                            tempNote = measureNote(idx);
                            tempBeat = cell2mat(totalSyms(chordNoteIdx(...
                                jj),5));
                        % For the case of rest, pitch is set to 0, and
                        % beat is set to default
                        elseif (isRest2)
                            tempPitch = 0;
                            tempNote = string(pitchTranslator(tempPitch));
                            tempBeat = cell2mat(totalSyms(chordNoteIdx(...
                                jj),5));
                        end

                        % Accidental locator
                        accidentalIdx = find((...
                            abs(cent_x(nonNoteIdx) - cent_x(chordNoteIdx(...
                                jj))) < 5*spaceHeight) & ...
                            (abs(cent_y(nonNoteIdx) - cent_y(chordNoteIdx(...
                                jj))) < 7*lineHeight) & ...
                            (cent_x(nonNoteIdx) < cent_x(chordNoteIdx(jj))));
                        accidentalIdx = accidentalIdx(find(strcmp(string(...
                            nonNoteSyms(accidentalIdx,3)), "accidental")));
                        if (~isempty(accidentalIdx))
                            [~, tempIdx] = min(abs(ledgerStaffSpace - ...
                                cent_y(nonNoteIdx(accidentalIdx))));
                            if (strcmp(string(nonNoteSyms(accidentalIdx,...
                                    4)), "natural"))
                                if ((ismember(tempIdx, keySigIdx) && ...
                                    strcmp(keyType, "sharp")) || ...
                                    (measurePitch(tempIdx) > pitchPreKS(...
                                    tempIdx)))
                                    tempPitch = tempPitch - 1;
                                elseif ((ismember(tempIdx, keySigIdx) && ...
                                    strcmp(keyType, "flat")) || ...
                                    (measurePitch(tempIdx) < pitchPreKS(...
                                    tempIdx)))
                                    tempPitch = tempPitch + 1;
```

```
                        end
                    else
                        tempPitch = tempPitch + cell2mat(...
                            nonNoteSyms(accidentalIdx,5));
                    end
                    measurePitch(tempIdx) = tempPitch;
                    tempNote = string(pitchTranslator(tempPitch));
                    measureNote(tempIdx) = tempNote;
                end
                % Accent dot locator
                accentDotIdx = find((...
                    abs(cent_x(nonNoteIdx) - cent_x(chordNoteIdx(...
                        jj))) <= 2.5*spaceHeight) & ...
                    (cent_y(chordNoteIdx(jj)) - cent_y(nonNoteIdx)...
                        <= 0.75*spaceHeight) & ...
                    (cent_y(chordNoteIdx(jj)) - cent_y(nonNoteIdx)...
                        >= -0.5*spaceHeight) & ...
                    (cent_x(nonNoteIdx) > cent_x(chordNoteIdx(jj))));
                accentDotIdx = find(strcmp(string(nonNoteSyms(...
                    accentDotIdx,3)), "dot"));
                % If accent dot exists, increase current beat by
                % a factor of 1.5
                if (~isempty(accentDotIdx))
                    tempBeat = tempBeat * 1.5;
                end

                % Add current note/beat values in note and beat
                % arrays
                totalNotes = [totalNotes; tempNote];
                totalBeat = [totalBeat; tempBeat];
            end
            % If total beat values are the same, there is no
            % need for further processing
            if (numel(unique(totalBeat)) == 1)
                % Write beat value to file
                fprintf(fid2,'%.3f, ', min(totalBeat));
                % Copy total notes in buffer
                noteBuf(1:length(totalBeat)) = totalNotes;

                % If a tie/slur starting edge is detected, note
                % break is set to 0
                if (~isempty(tieSlurStartIdx))
                    tiePresent = [tiePresent; tieSlurStartIdx];
                    noteBreak = 0;
                % If end edge is detected, add a note break
                % (set to 1)
                elseif(~isempty(tieSlurEndIdx) && ~isempty(...
                        tiePresent))
                    tiePresent = [];
                    noteBreak = 1;
                % If note is in between tie edges, note break is
                % also set to 0
                elseif (~isempty(tiePresent) && (cent_x(j) > ...
                        tiePresent(1)))
                    noteBreak = 0;
                % If above conditions are not met, set note
                % break to 1
                else
                    noteBreak = 1;
                end

                % Clear total notes and total beats, and write
                % current notes and note break value to file
```

```
                    totalNotes = [];
                    totalBeat = [];
                    fprintf(fid1,'\t{%s,%s,%s,%s, %s, %s, %s, %s, %d},\n', ...
                        noteBuf, noteBreak);
                else
                    % If multiple beats are present, the minimum value
                    % will be taken as current beat
                    [offset, ~] = min(totalBeat);
                    minIdx = find(totalBeat == min(totalBeat));
                    fprintf(fid2,'%.3f, ', min(totalBeat));
                    % Once the value is written to file, the rest of
                    % the beat values will be decreased by the value of
                    % of the minimum beat (offset)
                    for ii = 1:length(totalBeat)
                        if (ii ~= minIdx)
                            totalBeat(ii) = totalBeat(ii) - offset;
                        end
                    end
                    % Copy total notes to buffer and write to file
                    noteBuf(1:length(totalBeat)) = totalNotes;
                    fprintf(fid1,'\t{%s, %s, %s, %s, %s,%s,%s, %s, 0},\n', ...
                        noteBuf);
                    % Remove minimum beat, note from total note/beat
                    % arrays
                    totalNotes(minIdx) = [];
                    totalBeat(minIdx) = [];
                end
            else
                % Add current note/beat values in note and beat
                % arrays
                totalNotes = [totalNotes; tempNote];
                totalBeat = [totalBeat; tempBeat];
                % If total beat values are the same, there is no
                % need for further processing
                if (numel(unique(totalBeat)) == 1)
                    % If a tie/slur starting edge is detected, note
                    % break is set to 0
                    if (~isempty(tieSlurStartIdx))
                        tiePresent = [tiePresent; tieSlurStartIdx];
                        noteBreak = 0;
                    % If end edge is detected, add a note break
                    % (set to 1)
                    elseif(~isempty(tieSlurEndIdx) && ~isempty(...
                            tiePresent))
                        tiePresent = [];
                        noteBreak = 1;
                    % If note is in between tie edges, note break is
                    % also set to 0
                    elseif (~isempty(tiePresent) && (cent_x(j) > ...
                            tiePresent(1)))
                        noteBreak = 0;
                    % If above conditions are not met, set note
                    % break to 1
                    else
                        noteBreak = 1;
                    end

                    % Write beat value to file
                    fprintf(fid2,'%.3f, ', min(totalBeat));
                    % Copy total notes in buffer
                    noteBuf(1:length(totalBeat)) = totalNotes;
                    % Clear total notes and total beats, and write
                    % current notes and note break value to file
```

```
                        fprintf(fid1,'\t{%s, %s, %s, %s, %s,%s,%s,%s, %d},\n',...
                            noteBuf, noteBreak);
                        totalNotes = [];
                        totalBeat = [];
                    else
                        % If multiple beats are present, the minimum value
                        % will be taken as current beat
                        [offset, ~] = min(totalBeat);
                        minIdx = find(totalBeat == min(totalBeat));
                        fprintf(fid2,'%.3f, ', min(totalBeat));
                        % Once the value is written to file, the rest of
                        % the beat values will be decreased by the value of
                        % of the minimum beat (offset)
                        for ii = 1:length(totalBeat)
                            if (ii ~= minIdx)
                                totalBeat(ii) = totalBeat(ii) - offset;
                            end
                        end
                        % Copy total notes to buffer and write to file
                        noteBuf(1:length(totalBeat)) = totalNotes;
                        fprintf(fid1,'\t{%s, %s, %s, %s, %s, %s, %s, %s, 0},\n',...
                            noteBuf);
                        % Remove minimum beat, note from total note/beat
                        % arrays
                        totalNotes(minIdx) = [];
                        totalBeat(minIdx) = [];
                    end
                end
            %-------------------------------------------
            % Barline symbol check
            %-------------------------------------------
            elseif(isBar)
                % If barline is found, reset the pitch and note buffers
                measurePitch = pitch;
                measureNote = note;
                % Add new lines to both note and beat input files
                fprintf(fid1, '\n');
                fprintf(fid2, '\n\t');
            end
        end
    end
end
```

## HOMeR app

```
classdef HOMeR < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        HOMeRUIFigure           matlab.ui.Figure
        FileMenu                matlab.ui.container.Menu
        ExitMenu                matlab.ui.container.Menu
        HelpMenu                matlab.ui.container.Menu
        InstructionsMenu        matlab.ui.container.Menu
        AboutMenu               matlab.ui.container.Menu
        Status                  matlab.ui.control.TextArea
        FirstButton             matlab.ui.control.Button
        LastButton              matlab.ui.control.Button
        PrevButton              matlab.ui.control.Button
        NextButton              matlab.ui.control.Button
```

```
        SectionnavigationLabel  matlab.ui.control.Label
        Panel                   matlab.ui.container.Panel
        ReadScoreButton         matlab.ui.control.Button
        ResetButton             matlab.ui.control.Button
        SelectimagesLabel       matlab.ui.control.Label
        BrowseButton            matlab.ui.control.Button
        ExportDataButton        matlab.ui.control.Button
        ScoreoptionsLabel       matlab.ui.control.Label
        PicCount                matlab.ui.control.Label
        SectionImg              matlab.ui.control.UIAxes
    end

    properties (Access = public)
        Property % Description
        file
        path
        inputFile
        fid1
        fid2
        debug = 0;
        stop = 0;
        totalSections = 0;
        sectionBuf = {};
        tempCount;
    end

    methods (Access = public)
        function ReadScore(app, inputImg, fid1, fid2)
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%         PHASE ONE: PREPROCESSING         %%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            [img_bw, lineHeight, spaceHeight] = preprocess(inputImg);


            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%   PHASE TWO: MUSIC SYMBOLS RECOGNITION   %%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % Part 2.1: Staff detection, removal, and division
            [staffLines, staffHeights] = staffDetect(img_bw);
            dividers = staffDivider(img_bw, staffLines, lineHeight, ...
                spaceHeight);
            img_no_staff = staffRemove(img_bw, staffLines, staffHeights, ...
                spaceHeight);
            [sections, newStaffLines] = scoreToSections(img_no_staff, ...
                staffLines, dividers);

            % Part 2.2: Importing dataset for template matching
            [clef, key, timeSig, wholeNotes, rest, dot, other, ...
                tieslur] = readDataset;
            combined = [key; rest; other];

            totalTimeSignature = [];
            for i = 1:length(sections)
                app.totalSections = app.totalSections + 1;
                if (app.stop == 1)
                    break;
                end
                app.Status.Value = [app.Status.Value;
                    sprintf("%s\nSECTION %d:", pad('-', 100, 'left', '-'), ...
                    app.totalSections)];

                app.PicCount.Text = string(app.totalSections);
                app.sectionBuf{app.totalSections} = sections{i};
```

```
                I = imshow(app.sectionBuf{app.totalSections}, ...
                    'Parent', app.SectionImg, ...
                    'XData', [1 app.SectionImg.Position(3)], ...
                    'YData', [1 app.SectionImg.Position(4)]);
                app.SectionImg.XLim = [0 I.XData(2)];
                app.SectionImg.YLim = [0 I.YData(2)];

                % Part 2.3: Clef detection
                [clefs, clefBound]  = detectClefs(sections{i}, spaceHeight, ...
                    newStaffLines{i}, clef, app.debug);
                temp = [];
                for jjj = 1:size(clefs,1)
                    if (app.stop ~= 1)
                        temp = [temp, sprintf('%s %s, ',...
                        string(clefs(jjj,4)), string(clefs(jjj,3)))];
                   end
                end
                app.Status.Value = [app.Status.Value; strcat("Clefs: ", temp)];

                % Part 2.4: Key signature detection
                if (i == 1)
                    [keySignature, keys, keyTemp] = detectKeySig(sections{i}, ...
                        spaceHeight, i, newStaffLines{i}, clefBound, ...
                        clefBound+10*spaceHeight, key, app.debug);
                    keySigBound = keyTemp;
                    app.Status.Value = [app.Status.Value; sprintf(...
                        'Key signature: %s,', keySignature)];
                else
                    [keySignature, keys, keyTemp] = detectKeySig(...
                        sections{i}, spaceHeight, i, newStaffLines{i}, ...
                        clefBound, keySigBound, key, app.debug);
                    keySigBound = keyTemp;
                    app.Status.Value = [app.Status.Value; sprintf(...
                        'Key signature: %s,', keySignature)];
                end

                % Part 2.5: Time signature detection
                [timeSignature, timeSigBound] = detectTimeSig(sections{i}, ...
                    spaceHeight, keySigBound, timeSig, app.debug);

                totalTimeSignature =  [totalTimeSignature; timeSignature];

                if(isempty(timeSignature))
                    nextBound = keySigBound;
                else
                    nextBound = timeSigBound;
                    app.Status.Value = [app.Status.Value; sprintf(...
                        'Time signature: %s,', string(timeSignature(1,4)))];
                end

                % Part 2.6: Section sorting to unbeamed/beamed notes, and
                % other symbols
                [unbeamedSec, beamedSec, otherSymsSec] = noteSorter(...
                    sections{i}, nextBound, spaceHeight, lineHeight);

                % Part 2.7: Unbeamed note detection
                if(~isequal(zeros(size(sections{i})), unbeamedSec))
                    [unbeamedNotes, ledgerLineLocs1] = detectUnbeamedNotes(...
                        unbeamedSec, spaceHeight, lineHeight, wholeNotes, ...
                        app.debug);
                end
                % Part 2.8: Beamed note detection
                if(~isequal(zeros(size(sections{i})), beamedSec))
```

```
                    [beamedNotes, ledgerLineLocs2] = detectBeamedNotes(...
                        beamedSec, spaceHeight, app.debug);
                end

                % Part 2.9: Other symbol detection (rests, dots, ties, slurs,
                % barlines)
                [otherSymbols, ledgerLineLocs3] = detectOtherSymbols(...
                    otherSymsSec, spaceHeight, lineHeight, ...
                    newStaffLines{i}, combined, dot, tieslur, app.debug);

                % Part 2.10: Combining symbols sorted by horizontal location for
                % next phase
                totalSyms = [unbeamedNotes; beamedNotes; otherSymbols];
                totalSyms = sortrows(totalSyms, 6);
                app.Status.Value = [app.Status.Value; sprintf('Symbols:')];
                for jjj = 1:length(totalSyms)
                    if (app.stop ~= 1)
                        app.Status.Value = [app.Status.Value; sprintf('\t%s %s',...
                            string(totalSyms(jjj,4)), string(totalSyms(jjj,3)))];
                   end
                end
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                %%%%%%%   PHASES 3/4: MUSIC NOTATION RECONSTRUCTION   %%%%%%
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                % Part 3.1 Calculating ledger/staff lines and staff space
                % locations, and assigning appropriate pitches
                ledgerLineLocs = sort([ledgerLineLocs1; ledgerLineLocs2; ...
                    ledgerLineLocs3]);
                [ledgerStaffSpace, pitch, note, pitchPreKS, staffCenters, keyType,...
                    keySigIdx, grandStaffDivider] = finalStaffPitchAssignment(...
                    sections{i}, newStaffLines{i}, ledgerLineLocs, clefs, keys);

                % Part 3.2
                if (isempty(grandStaffDivider))
                    SoloSheetMusicGenerator(totalSyms, totalTimeSignature, ...
                        ledgerStaffSpace,  pitch, note, pitchPreKS, ...
                        staffCenters, keyType, keySigIdx, spaceHeight, ...
                        lineHeight, fid1, fid2);
                else
                    PianoSheetMusicGenerator(totalSyms, totalTimeSignature, ...
                        ledgerStaffSpace, pitch, note, pitchPreKS, ...
                        staffCenters, keyType, keySigIdx, spaceHeight, ...
                        lineHeight, fid1, fid2);
                end

                scroll(app.Status, 'bottom');
                pause(1);
            end
        scroll(app.Status, 'bottom');
        end
    end


    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app)
            movegui(app.HOMeRUIFigure,"center");
            disableDefaultInteractivity(app.SectionImg);
            app.SectionImg.Visible = false;
            app.ExportDataButton.Enable = false;
            app.ReadScoreButton.Enable = false;
```

```
        end

        % Button pushed function: BrowseButton
        function BrowseButtonPushed(app, event)
            app.FirstButton.Enable = false;
            app.PrevButton.Enable = false;
            app.NextButton.Enable = false;
            app.LastButton.Enable = false;
            [app.file,app.path] = uigetfile({'*.jpg';'*.png'}, ...
                'multiselect', 'on');
            cla(app.SectionImg);
            app.totalSections = 0;
            if isequal(app.file, 0)
                app.Status.Value = '';
                app.Status.Value = 'User selection canceled.';
            else
                app.Status.Value = '';
                app.inputFile = string(fullfile(app.path, app.file));
                if (length(app.inputFile) == 1)
                    readStatus = strcat('User selected', " ", fullfile(...
                        app.path, app.file));
                    app.Status.Value = [app.Status.Value; readStatus];
                else
                    for i = 1:length(app.inputFile)
                        readStatus = strcat('User selected', " ", string(...
                            app.inputFile(i)));
                        app.Status.Value = [app.Status.Value; readStatus];
                    end
                end
                app.ReadScoreButton.Enable = true;
                app.ExportDataButton.Enable = false;

                dir = 'OrganNotes.sdk\Synthesizer\src\';
                outFile1 = 'sheetNotes.h';
                fullFile = strcat(dir, outFile1);
                app.fid1 = fopen(fullFile, 'w');

                outFile2 = 'sheetBeat.h';
                fullFile = strcat(dir, outFile2);
                app.fid2 = fopen(fullFile, 'w');

                fprintf(app.fid1,'#include "notes.h"\n\n');
                fprintf(app.fid1,'int noteArray[][9] = \n{\n');
                fprintf(app.fid2,'double beatArray[] = \n{\n\t');

                app.stop = 0;
            end
            app.PicCount.Text = "";
        end

        % Button pushed function: ReadScoreButton
        function ReadScoreButtonPushed(app, event)
            app.totalSections = 0;
            app.ReadScoreButton.Enable = false;
            app.Status.Value = [app.Status.Value;
                sprintf("%s", pad('-', 100, 'left', '-'))];
           if (length(app.inputFile) == 1)
                scanStatus = strcat('Scanning', " ", app.file, '...');
                tic
                app.Status.Value = [app.Status.Value; scanStatus];
                inputImg = imread(app.inputFile);
                ReadScore(app, inputImg, app.fid1, app.fid2);
```

```
                elapsed = toc;
                app.Status.Value = [app.Status.Value;
                    sprintf("%s", pad('-', 100, 'left', '-'))];
            else
                tic
                for i = 1:length(app.inputFile)
                    scanStatus = strcat('Scanning', " ", string(...
                        app.file(i)), " ", '...');
                    app.Status.Value = [app.Status.Value; scanStatus];
                    inputImg = imread(cell2mat(app.inputFile(i)));
                    ReadScore(app, inputImg, app.fid1, app.fid2);
                    app.Status.Value = [app.Status.Value;
                        sprintf("%s", pad('-', 100, 'left', '-'))];
                end
                elapsed = toc;
            end

            if (app.stop ~= 1)
                app.Status.Value = [app.Status.Value;
                    sprintf("Elapsed time: %.6f seconds", elapsed); ...
                    sprintf("%s", pad('-', 100, 'left', '-')); sprintf(...
                    "Please open Xilinx SDK to listen to the tune.")];
                scroll(app.Status, 'bottom');

                fprintf(app.fid1,'};');
                fprintf(app.fid2,'\n};');
                pause(1);

                scroll(app.Status, 'bottom');

                app.FirstButton.Enable = true;
                app.PrevButton.Enable = true;
                app.ExportDataButton.Enable = true;
                app.tempCount = app.totalSections;
            end

        end

        % Button pushed function: ResetButton
        function ResetButtonPushed(app, event)
            cla(app.SectionImg);
            app.ExportDataButton.Enable = false;
            app.ReadScoreButton.Enable = false;
            app.FirstButton.Enable = false;
            app.PrevButton.Enable = false;
            app.NextButton.Enable = false;
            app.LastButton.Enable = false;
            app.Status.Value = "";
            app.stop = 1;
            app.totalSections = 0;
            app.PicCount.Text = "";
            close all
        end

        % Button pushed function: FirstButton
        function FirstButtonPushed(app, event)
            app.tempCount = 1;
            I = imshow(app.sectionBuf{app.tempCount}, ...
                'Parent', app.SectionImg, ...
                'XData', [1 app.SectionImg.Position(3)], ...
                'YData', [1 app.SectionImg.Position(4)]);
            % Set limits of axes
            app.SectionImg.XLim = [0 I.XData(2)];
```

```
            app.SectionImg.YLim = [0 I.YData(2)];
            app.PicCount.Text = string(app.tempCount);
            app.FirstButton.Enable = false;
            app.PrevButton.Enable = false;
            app.NextButton.Enable = true;
            app.LastButton.Enable = true;
        end

        % Button pushed function: LastButton
        function LastButtonPushed(app, event)
            app.tempCount = app.totalSections;
            I = imshow(app.sectionBuf{app.tempCount}, ...
                'Parent', app.SectionImg, ...
                'XData', [1 app.SectionImg.Position(3)], ...
                'YData', [1 app.SectionImg.Position(4)]);
            % Set limits of axes
            app.SectionImg.XLim = [0 I.XData(2)];
            app.SectionImg.YLim = [0 I.YData(2)];
            app.PicCount.Text = string(app.tempCount);
            app.FirstButton.Enable = true;
            app.NextButton.Enable = false;
            app.PrevButton.Enable = true;
            app.LastButton.Enable = false;
        end

        % Button pushed function: NextButton
        function NextButtonPushed(app, event)
            app.tempCount = app.tempCount + 1;
            I = imshow(app.sectionBuf{app.tempCount}, ...
                'Parent', app.SectionImg, ...
                'XData', [1 app.SectionImg.Position(3)], ...
                'YData', [1 app.SectionImg.Position(4)]);
            % Set limits of axes
            app.SectionImg.XLim = [0 I.XData(2)];
            app.SectionImg.YLim = [0 I.YData(2)];
            app.PicCount.Text = string(app.tempCount);
            if (app.tempCount < app.totalSections)
                app.NextButton.Enable = true;
                app.LastButton.Enable = true;
            else
                app.NextButton.Enable = false;
                app.LastButton.Enable = false;
            end
            app.FirstButton.Enable = true;
            app.PrevButton.Enable = true;

        end

        % Button pushed function: PrevButton
        function PrevButtonPushed(app, event)
            app.tempCount = app.tempCount - 1;
            I = imshow(app.sectionBuf{app.tempCount}, ...
                'Parent', app.SectionImg, ...
                'XData', [1 app.SectionImg.Position(3)], ...
                'YData', [1 app.SectionImg.Position(4)]);
            % Set limits of axes
            app.SectionImg.XLim = [0 I.XData(2)];
            app.SectionImg.YLim = [0 I.YData(2)];
            app.PicCount.Text = string(app.tempCount);
            if (app.tempCount > 1 )
                app.FirstButton.Enable = true;
                app.PrevButton.Enable = true;
            else
```

```
                app.FirstButton.Enable = false;
                app.PrevButton.Enable = false;
            end
            app.NextButton.Enable = true;
            app.LastButton.Enable = true;
        end

        % Menu selected function: ExitMenu
        function ExitMenuSelected(app, event)
            app.delete;
        end

        % Button pushed function: ExportDataButton
        function ExportDataButtonPushed(app, event)
            scroll(app.Status, 'bottom');
            value = app.Status.Value;  % Value entered in textArea
            fileName = strcat('SheetData.txt');
            f=fopen(fileName,'w');
            formatSpec = "%s\n";
            for i =1:length(value)
                fprintf(f,formatSpec,value{i});
            end
            fclose(f);
            app.Status.Value = [app.Status.Value; sprintf(...
                "%s", pad('-', 100, 'left', '-')); sprintf(...
                "Sheet music data exported.")];
            app.ExportDataButton.Enable = false;
            pause(1);
            scroll(app.Status, 'bottom');


        end

        % Menu selected function: AboutMenu
        function AboutMenuSelected(app, event)
            About;
        end

        % Menu selected function: InstructionsMenu
        function InstructionsMenuSelected(app, event)
            Instructions;
        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create HOMeRUIFigure and hide until all components are created
            app.HOMeRUIFigure = uifigure('Visible', 'off');
            app.HOMeRUIFigure.Position = [100 100 910 610];
            app.HOMeRUIFigure.Name = 'HOMeR';
            app.HOMeRUIFigure.Resize = 'off';

            % Create FileMenu
            app.FileMenu = uimenu(app.HOMeRUIFigure);
            app.FileMenu.Text = 'File';

            % Create ExitMenu
            app.ExitMenu = uimenu(app.FileMenu);
            app.ExitMenu.MenuSelectedFcn = createCallbackFcn(app,
```

```
            @ExitMenuSelected, true);
        app.ExitMenu.Text = 'Exit';

        % Create HelpMenu
        app.HelpMenu = uimenu(app.HOMeRUIFigure);
        app.HelpMenu.Text = 'Help';

        % Create InstructionsMenu
        app.InstructionsMenu = uimenu(app.HelpMenu);
        app.InstructionsMenu.MenuSelectedFcn = createCallbackFcn(app,
            @InstructionsMenuSelected, true);
        app.InstructionsMenu.Text = 'Instructions';

        % Create AboutMenu
        app.AboutMenu = uimenu(app.HelpMenu);
        app.AboutMenu.MenuSelectedFcn = createCallbackFcn(app,
            @AboutMenuSelected, true);
        app.AboutMenu.Text = 'About';

        % Create Status
        app.Status = uitextarea(app.HOMeRUIFigure);
        app.Status.Editable = 'off';
        app.Status.FontName = 'Cambria';
        app.Status.FontSize = 14;
        app.Status.Position = [361 313 535 270];

        % Create FirstButton
        app.FirstButton = uibutton(app.HOMeRUIFigure, 'push');
        app.FirstButton.ButtonPushedFcn = createCallbackFcn(app,
            @FirstButtonPushed, true);
        app.FirstButton.Enable = 'off';
        app.FirstButton.Position = [730 16 39 22];
        app.FirstButton.Text = 'First';

        % Create LastButton
        app.LastButton = uibutton(app.HOMeRUIFigure, 'push');
        app.LastButton.ButtonPushedFcn = createCallbackFcn(app,
            @LastButtonPushed, true);
        app.LastButton.Enable = 'off';
        app.LastButton.Position = [857 16 39 22];
        app.LastButton.Text = 'Last';

        % Create PrevButton
        app.PrevButton = uibutton(app.HOMeRUIFigure, 'push');
        app.PrevButton.ButtonPushedFcn = createCallbackFcn(app,
            @PrevButtonPushed, true);
        app.PrevButton.Enable = 'off';
        app.PrevButton.Position = [774 16 25 22];
        app.PrevButton.Text = '<';

        % Create NextButton
        app.NextButton = uibutton(app.HOMeRUIFigure, 'push');
        app.NextButton.ButtonPushedFcn = createCallbackFcn(app,
            @NextButtonPushed, true);
        app.NextButton.Enable = 'off';
        app.NextButton.Position = [826 16 26 22];
        app.NextButton.Text = '>';

        % Create SectionnavigationLabel
        app.SectionnavigationLabel = uilabel(app.HOMeRUIFigure);
        app.SectionnavigationLabel.HorizontalAlignment = 'right';
        app.SectionnavigationLabel.Position = [615 16 107 22];
        app.SectionnavigationLabel.Text = 'Section navigation:';
```

```
            % Create Panel
            app.Panel = uipanel(app.HOMeRUIFigure);
            app.Panel.Position = [19 313 322 270];

            % Create ReadScoreButton
            app.ReadScoreButton = uibutton(app.Panel, 'push');
            app.ReadScoreButton.ButtonPushedFcn = createCallbackFcn(app,
               @ReadScoreButtonPushed, true);
            app.ReadScoreButton.Enable = 'off';
            app.ReadScoreButton.Position = [153 134 100 22];
            app.ReadScoreButton.Text = 'Read Score';

            % Create ResetButton
            app.ResetButton = uibutton(app.Panel, 'push');
            app.ResetButton.ButtonPushedFcn = createCallbackFcn(app,
               @ResetButtonPushed, true);
            app.ResetButton.Position = [153 58 100 22];
            app.ResetButton.Text = 'Reset';

            % Create SelectimagesLabel
            app.SelectimagesLabel = uilabel(app.Panel);
            app.SelectimagesLabel.Position = [47 194 107 22];
            app.SelectimagesLabel.Text = 'Select image(s): ';

            % Create BrowseButton
            app.BrowseButton = uibutton(app.Panel, 'push');
            app.BrowseButton.ButtonPushedFcn = createCallbackFcn(app,
               @BrowseButtonPushed, true);
            app.BrowseButton.Position = [153 194 100 22];
            app.BrowseButton.Text = 'Browse';

            % Create ExportDataButton
            app.ExportDataButton = uibutton(app.Panel, 'push');
            app.ExportDataButton.ButtonPushedFcn = createCallbackFcn(app,
               @ExportDataButtonPushed, true);
            app.ExportDataButton.Enable = 'off';
            app.ExportDataButton.Position = [153 97 100 22];
            app.ExportDataButton.Text = 'Export Data';

            % Create ScoreoptionsLabel
            app.ScoreoptionsLabel = uilabel(app.Panel);
            app.ScoreoptionsLabel.HorizontalAlignment = 'right';
            app.ScoreoptionsLabel.Position = [48 134 83 22];
            app.ScoreoptionsLabel.Text = 'Score options:';

            % Create PicCount
            app.PicCount = uilabel(app.HOMeRUIFigure);
            app.PicCount.HorizontalAlignment = 'center';
            app.PicCount.Position = [798 15 29 25];
            app.PicCount.Text = '';

            % Create SectionImg
            app.SectionImg = uiaxes(app.HOMeRUIFigure);
            app.SectionImg.Toolbar.Visible = 'off';
            app.SectionImg.XTick = [];
            app.SectionImg.YTick = [];
            app.SectionImg.Clipping = 'off';
            app.SectionImg.Visible = 'off';
            app.SectionImg.HandleVisibility = 'off';
            app.SectionImg.Interruptible = 'off';
            app.SectionImg.HitTest = 'off';
            app.SectionImg.Position = [1 40 910 265];
```

```
            % Show the figure after all components are created
            app.HOMeRUIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = OMRApp_exported

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.HOMeRUIFigure)

            % Execute the startup function
            runStartupFcn(app, @startupFcn)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.HOMeRUIFigure)
        end
    end
end
```

## Appendix C: Programmable Logic Source Code

### Input note controller

```
/**********************************************************************
 * Module name:     fcw_ctrl
 * Input arg(s):    Note value (legal: 1-88)
 * Output arg(s):   Note equivalents of the fundamental frequency and
 *                  overtones
 * Description:     Checks the value of the input note and produces the
 *                  note values of the corresponding sub-third and sub-
 *                  fundamental frequencies, and the input note itself
 **********************************************************************/
`timescale 1ns / 1ps
module fcw_ctrl(
    // port declarations
    input [6:0] note,
    output reg [6:0] fcw_sub_fund,
    output reg [6:0] fcw_sub_third,
    output reg [6:0] fcw_fund
    );

    // Sub-fundamental note offset: 12 semitones below the input note
    parameter SUB_FUND = 12;
    // Sub-third note offset: 7 semitones above the input note
    parameter SUB = 7;

    always @ (note) begin
        // Since the synthesizer is based on the grand piano, the legal
        // inputs are between 1 and 88
        if (note >= 1 && note <= 88) begin
            // for all legal inputs, the fundamental note is the
            // same as the input note
            fcw_fund = note;
            // To avoid hearing unnecessary noises, adjustments are made
            // for certain cases
            if (note < 13) begin
                // for notes lower than 13, the sub-fundamental harmonic
                // is removed
                fcw_sub_fund = 127;
                fcw_sub_third = note + SUB;
            end
            else if (note > 81) begin
                // for notes higher than 81, the sub-third harmonic
                // is removed
                fcw_sub_fund = note - SUB_FUND;
                fcw_sub_third = 127;
            end
            else begin
                // for all other cases, both overtone cases exist
                fcw_sub_fund = note - SUB_FUND;
                fcw_sub_third = note + SUB;
            end
        end
        // For all inputs that are out of range, the notes are set to
        // max value (silence)
        else begin
```

```
            fcw_sub_fund = 127;
            fcw_sub_third = 127;
            fcw_fund = 127;
        end
    end
endmodule
```

### Frequency control lookup table

```
/*************************************************************************
 * Module name:         fcw_table
 * Input arg(s):        FCW address
 * Output arg(s):       Equivalent FCW
 * Description:         Lookup table for the frequency control word based
 *                      on the input note
 *************************************************************************/
`timescale 1ns / 1ps
module fcw_table(
    // port declarations
    input [6:0] fcw_addr,
    output reg [23:0] fcw
    );

    always @ (fcw_addr) begin
        case (fcw_addr)
            7'h01: fcw = 24'h00258b;
            7'h02: fcw = 24'h0027c7;
            7'h03: fcw = 24'h002a25;
            7'h04: fcw = 24'h002ca6;
            7'h05: fcw = 24'h002f4e;
            7'h06: fcw = 24'h00321e;
            7'h07: fcw = 24'h003519;
            7'h08: fcw = 24'h003841;
            7'h09: fcw = 24'h003b9a;
            7'h0a: fcw = 24'h003f25;
            7'h0b: fcw = 24'h0042e6;
            7'h0c: fcw = 24'h0046e0;
            7'h0d: fcw = 24'h004b17;
            7'h0e: fcw = 24'h004f8f;
            7'h0f: fcw = 24'h00544a;
            7'h10: fcw = 24'h00594d;
            7'h11: fcw = 24'h005e9c;
            7'h12: fcw = 24'h00643c;
            7'h13: fcw = 24'h006a32;
            7'h14: fcw = 24'h007083;
            7'h15: fcw = 24'h007734;
            7'h16: fcw = 24'h007e4a;
            7'h17: fcw = 24'h0085cd;
            7'h18: fcw = 24'h008dc1;
            7'h19: fcw = 24'h00962f;
            7'h1a: fcw = 24'h009f1e;
            7'h1b: fcw = 24'h00a894;
            7'h1c: fcw = 24'h00b29a;
            7'h1d: fcw = 24'h00bd39;
            7'h1e: fcw = 24'h00c879;
            7'h1f: fcw = 24'h00d465;
            7'h20: fcw = 24'h00e106;
            7'h21: fcw = 24'h00ee68;
            7'h22: fcw = 24'h00fc95;
            7'h23: fcw = 24'h010b9a;
```

```
            7'h24: fcw = 24'h011b83;
            7'h25: fcw = 24'h012c5f;
            7'h26: fcw = 24'h013e3c;
            7'h27: fcw = 24'h015128;
            7'h28: fcw = 24'h016534;
            7'h29: fcw = 24'h017a72;
            7'h2a: fcw = 24'h0190f3;
            7'h2b: fcw = 24'h01a8ca;
            7'h2c: fcw = 24'h01c20d;
            7'h2d: fcw = 24'h01dcd0;
            7'h2e: fcw = 24'h01f92a;
            7'h2f: fcw = 24'h021734;
            7'h30: fcw = 24'h023707;
            7'h31: fcw = 24'h0258bf;
            7'h32: fcw = 24'h027c78;
            7'h33: fcw = 24'h02a250;
            7'h34: fcw = 24'h02ca69;
            7'h35: fcw = 24'h02f4e4;
            7'h36: fcw = 24'h0321e6;
            7'h37: fcw = 24'h035195;
            7'h38: fcw = 24'h03841a;
            7'h39: fcw = 24'h03b9a0;
            7'h3a: fcw = 24'h03f254;
            7'h3b: fcw = 24'h042e68;
            7'h3c: fcw = 24'h046e0e;
            7'h3d: fcw = 24'h04b17e;
            7'h3e: fcw = 24'h04f8f0;
            7'h3f: fcw = 24'h0544a1;
            7'h40: fcw = 24'h0594d2;
            7'h41: fcw = 24'h05e9c9;
            7'h42: fcw = 24'h0643cd;
            7'h43: fcw = 24'h06a32b;
            7'h44: fcw = 24'h070834;
            7'h45: fcw = 24'h07733f;
            7'h46: fcw = 24'h07e4aa;
            7'h47: fcw = 24'h085cd0;
            7'h48: fcw = 24'h08dc1e;
            7'h49: fcw = 24'h0962fc;
            7'h4a: fcw = 24'h09f1e1;
            7'h4b: fcw = 24'h0a8941;
            7'h4c: fcw = 24'h0b29a4;
            7'h4d: fcw = 24'h0bd392;
            7'h4e: fcw = 24'h0c879a;
            7'h4f: fcw = 24'h0d4657;
            7'h50: fcw = 24'h0e1069;
            7'h51: fcw = 24'h0ee682;
            7'h52: fcw = 24'h0fc955;
            7'h53: fcw = 24'h10b9a1;
            7'h54: fcw = 24'h11b83c;
            7'h55: fcw = 24'h12c5f9;
            7'h56: fcw = 24'h13e3c0;
            7'h57: fcw = 24'h151287;
            7'h58: fcw = 24'h16534c;
            default: fcw = 24'h000000;
        endcase
    end
endmodule
```

## Numerically-controlled oscillator

```
/*************************************************************************
 * Module name:         nco
 * Input arg(s):        clock, reset, frequency control word
 * Output arg(s):       ROM address
 * Description:         Numerically-controlled oscillator that generates the
 *                       address values for the triangle waveform lookup table
 *************************************************************************/
`timescale 1ns / 1ps
module nco(
    input clk, rst,
    input [23:0] fcw,
    output reg [11:0] addr
    );

    // Scaling factor to produce a 48kHz to match audio codec sampling
    // frequency
    parameter DIV = 28'd1024;

    // internal signal declarations
    reg fdiv_clk;
    reg [23:0] accum;
    reg [28:0] fdiv_cnt;

    // Sampling frequency divider process (48kHz clock generator)
    always @ (posedge clk or negedge rst) begin
        if(!rst)
            fdiv_cnt <= 0;
        else if (fdiv_cnt < (DIV-1))
            fdiv_cnt <= fdiv_cnt +1;
        else
            fdiv_cnt <= 0;
        end
    always @ (fdiv_cnt) fdiv_clk = (fdiv_cnt < (DIV/2)) ? 1'b0 : 1'b1;

    // Phase accummulator process
    always @ (posedge fdiv_clk or negedge rst)
    begin
        if (!rst)
            accum <= 0;
        else
            accum <= accum + fcw;
        end

    // To address the ROM, the 12 MSB's of the phase accummulator
    // will be used
    always @ (accum or fcw) begin
        if (fcw == 0)
            addr = 12'h000;
        else
            addr = accum[23:12];
    end

endmodule
```

### Direct digital synthesizer

```
/*************************************************************************
 * Module name:         music_dds
 * Input arg(s):        Clock, reset, input note
 * Output arg(s):       Single triangle waveform output
 * Description:         Top-level module for a digital direct synthesizer
 *************************************************************************/
`timescale 1ns / 1ps
module music_dds(
    // port declarations
    input clk, rst,
    input [6:0] fcw_addr,
    output [15:0] tri_out
    );

    // internal signal declarations
    wire [23:0] fcw;
    wire [11:0] addr;

    // frequency control word ROM instantiation
    fcw_table fcw_lut(.fcw_addr(fcw_addr), .fcw(fcw));

    // numerically-controlled oscillator instantiation
    nco nco(.clk(clk), .rst(rst), .fcw(fcw), .addr(addr));

    // triangle values values ROM instantiation
    tri_table tri_lut(
        .tri_addr(addr), .tri_word(tri_out));

endmodule
```

### Single tone generator

```
/*************************************************************************
 * Module name:         tone_gen
 * Input arg(s):        fundamental, sub-fundamental, and sub-third waveform
 *                      samples
 * Output arg(s):       sum of samples
 * Description:         Function that produces an organ-like waveform sample
 *************************************************************************/
`timescale 1ns / 1ps
module tone_gen(
    // port declarations
    input [15:0] sub_fund,
    input [15:0] sub_third,
    input [15:0] fund,
    output reg [17:0] tone
    );

    always @ (*)
        // Adds the three frequency samples to add timbre to the tone
        // - the overtones are shifted right so that they won't
        // - overpower the fundamental sound
        tone = (sub_fund >> 4) + (sub_third >> 4) + (fund << 1);

endmodule
```

## Organ tone generator

```
/*************************************************************************
 * Module name:         organ_note
 * Input arg(s):        Clock, reset, note
 * Output arg(s):       Synthesized wave output
 * Description:         Tfop-level module for a single-note synthesizer
 *************************************************************************/
 `timescale 1ns / 1ps
module organ_note(
    // port declarations
    input clk, rst,
    input [6:0] note,
    output [17:0] wave_out
    );

    // internal signal declarations
    wire [6:0] fcw_sub_fund;
    wire [6:0] fcw_sub_third;
    wire [6:0] fcw_fund;
    wire [15:0] sub_fund;
    wire [15:0] sub_third;
    wire [15:0] fund;

    // frequency control word controller instantiation
    fcw_ctrl ctrl(.note(note),
        .fcw_sub_fund(fcw_sub_fund),
        .fcw_sub_third(fcw_sub_third),
        .fcw_fund(fcw_fund)
        );

    // sub-fundamental DDS instantiation
    music_dds dds_sub_fund(.clk(clk), .rst(rst),
        .fcw_addr(fcw_sub_fund),
        .tri_out(sub_fund)
        );

    // sub-third DDS instantiation
    music_dds dds_sub_third(.clk(clk), .rst(rst),
        .fcw_addr(fcw_sub_third),
        .tri_out(sub_third)
        );

    // fundamental DDS instantiation
    music_dds dds_fund(.clk(clk), .rst(rst),
        .fcw_addr(fcw_fund),
        .tri_out(fund)
        );

    // tone generator instantiation
    tone_gen tone_gen(
        .sub_fund(sub_fund),
        .sub_third(sub_third),
        .fund(fund),
        .tone(wave_out)
        );

endmodule
```

## Multi-tone blender

```
/*************************************************************************
 * Module name:         blender
 * Input arg(s):        Clock, reset, 8 input notes
 * Output arg(s):       Synthesized waveform
 * Description:         Performs additive synthesis on the individual organ
 *                      notes to produce a synthesized waveform
 *************************************************************************/
`timescale 1ns / 1ps
module blender(
    // port declarations
    input clk, rst,
    input [17:0] note7, note6, note5, note4,
    input [17:0] note3,note2, note1, note0,
    output reg [23:0] wave_out
    );

    reg [23:0] wave_tmp;

    always @ (*)
        // Additive synthesis is simply adding waveforms to generate timbre
        wave_tmp = note7 + note6 + note5 + note4 + note3 + note2 + note1 + note0;

    always @ (posedge clk or negedge rst)
    begin
        // outputs a zero at reset, otherwise, a shifted resultant waveform
        // is produced
        if (!rst) wave_out <= 0;
        else wave_out <= (wave_tmp << 3);
    end

endmodule
```

## Organ synthesizer

```
/*************************************************************************
 * Module name:         harmonizer
 * Input arg(s):        Clock, reset, 8 input notes
 * Output arg(s):       Synthesized digital waveform
 * Description:         Produces a 24-bit digital waveform from the sum of
 *                      8 individual waveforms
 *************************************************************************/
`timescale 1ns / 1ps
module harmonizer(
    // port declarations
    input clk, rst,
    input [6:0] note_in7, note_in6, note_in5, note_in4,
    input [6:0] note_in3, note_in2, note_in1, note_in0,
    output [23:0] wave_out
    );

    // internal signal declarations
    wire [17:0] note7, note6, note5, note4;
    wire [17:0] note3, note2, note1, note0;

    // organ synthesizer instantiation for note 7
    organ_note N7(.clk(clk), .rst(rst),
       .note(note_in7), .wave_out(note7));
```

```
    // organ synthesizer instantiation for note 6
    organ_note N6(.clk(clk), .rst(rst),
       .note(note_in6), .wave_out(note6));

    // organ synthesizer instantiation for note 5
    organ_note N5(.clk(clk), .rst(rst),
       .note(note_in5), .wave_out(note5));

    // organ synthesizer instantiation for note 4
    organ_note N4(.clk(clk), .rst(rst),
       .note(note_in4), .wave_out(note4));

    // organ synthesizer instantiation for note 3
    organ_note N3(.clk(clk), .rst(rst),
       .note(note_in3), .wave_out(note3));

    // organ synthesizer instantiation for note 2
    organ_note N2(.clk(clk), .rst(rst),
       .note(note_in2), .wave_out(note2));

    // organ synthesizer instantiation for note 1
    organ_note N1(.clk(clk), .rst(rst),
       .note(note_in1), .wave_out(note1));

    // organ synthesizer instantiation for note 0
    organ_note N0(.clk(clk), .rst(rst),
       .note(note_in0), .wave_out(note0));

    // waveform blender instantiation
    blender wav_blend(.clk(clk), .rst(rst),
        .note7(note7), .note6(note6),
        .note5(note5), .note4(note4),
        .note3(note3), .note2(note2),
        .note1(note1), .note0(note0),
        .wave_out(wave_out));
```

## Appendix D: Processing System Source Code

### Note list

```
/*****************************************************************************
 * File name:    notes.h
 * Description:  Note definitions based on pitch
 *****************************************************************************/
#ifndef SRC_NOTES_H_
#define SRC_NOTES_H_

#define A0       1
#define Bb0      2
#define B0       3
#define C1       4
#define Cs1      5
#define D1       6
#define Eb1      7
#define E1       8
#define F1       9
#define Fs1      10
#define G1       11
#define Gs1      12
#define A1       13
#define Bb1      14
#define B1       15
#define C2       16
#define Cs2      17
#define D2       18
#define Eb2      19
#define E2       20
#define F2       21
#define Fs2      22
#define G2       23
#define Gs2      24
#define A2       25
#define Bb2      26
#define B2       27
#define C3       28
#define Cs3      29
#define D3       30
#define Eb3      31
#define E3       32
#define F3       33
#define Fs3      34
#define G3       35
#define Gs3      36
#define A3       37
#define Bb3      38
#define B3       39
#define C4       40
#define Cs4      41
#define D4       42
#define Eb4      43
#define E4       44
#define F4       45
#define Fs4      46
#define G4       47
#define Gs4      48
#define A4       49
#define Bb4      50
```

```
#define B4       51
#define C5       52
#define Cs5      53
#define D5       54
#define Eb5      55
#define E5       56
#define F5       57
#define Fs5      58
#define G5       59
#define Gs5      60
#define A5       61
#define Bb5      62
#define B5       63
#define C6       64
#define Cs6      65
#define D6       66
#define Eb6      67
#define E6       68
#define F6       69
#define Fs6      70
#define G6       71
#define Gs6      72
#define A6       73
#define Bb6      74
#define B6       75
#define C7       76
#define Cs7      77
#define D7       78
#define Eb7      79
#define E7       80
#define F7       81
#define Fs7      82
#define G7       83
#define Gs7      84
#define A7       85
#define Bb7      86
#define B7       87
#define C8       88

#endif /* SRC_NOTES_H_ */
```

### Initialization header

```
/*****************************************************************************
 * File name:           init.h
 * Description:         Declares function prototypes and global variables;
 *                      defines and redefines parameters
 *****************************************************************************/
#ifndef SRC_INIT_H_
#define SRC_INIT_H_

//-----------------------------------------------------
// header files
//-----------------------------------------------------
#include "xparameters.h"
#include "xgpio.h"
#include "xtmrctr.h"
#include "xscugic.h"
#include "xil_exception.h"
#include "xil_printf.h"
```

```
#include "stdio.h"
#include "xiicps.h"
#include "sleep.h"
#include <stdbool.h>
#include "notes.h"

//----------------------------------------------------
// Function prototypes
//----------------------------------------------------
int Gpio_Init();
int Timer_Init();
int InterruptSystemSetup(XScuGic *XScuGicInstancePtr);
int IntcInitFunction(u16 DeviceId, XTmrCtr *TmrInstancePtr, XGpio *BtnInstancePtr,
    XGpio *SWInstancePtr);

//----------------------------------------------------
// Parameter definitions/re-definitions
//----------------------------------------------------
#define INTC_DEVICE_ID      XPAR_PS7_SCUGIC_0_DEVICE_ID
#define TMR_DEVICE_ID       XPAR_TMRCTR_0_DEVICE_ID

#define BTN_DEV_ID          XPAR_AXI_GPIO_BTN_DEVICE_ID
#define SW_DEV_ID           XPAR_AXI_GPIO_SW_DEVICE_ID
#define LEDS_DEV_ID         XPAR_AXI_GPIO_LEDS_DEVICE_ID
#define MUTEN_DEV_ID        XPAR_AXI_GPIO_MUTEN_DEVICE_ID

#define INTC_TMR_INTERRUPT_ID XPAR_FABRIC_AXI_TIMER_0_INTERRUPT_INTR
#define INTC_BTN_INTERRUPT_ID XPAR_FABRIC_AXI_GPIO_BTN_IP2INTC_IRPT_INTR
#define INTC_SW_INTERRUPT_ID XPAR_FABRIC_AXI_GPIO_SW_IP2INTC_IRPT_INTR

#define BTN_INT             XGPIO_IR_CH1_MASK
#define SW_INT              XGPIO_IR_CH1_MASK

#define TMR_LOAD            0xFFFF0000
#define ARR_LEN 9

//----------------------------------------------------
// Global variables
//----------------------------------------------------
XGpio BTNInst, SWInst, LEDInst, MUTENInst;
XScuGic INTCInst;
XTmrCtr TMRInst;
XIicPs Iic;


#endif /* SRC_INIT_H_ */
```

## Initialization

```
/*****************************************************************************
 * File name:           init.c
 * Description:         Contains necessary initialization functions
 *****************************************************************************/
#include "init.h"
#include "intc_handler.h"

//----------------------------------------------------
// GPIO INITIALIATION FUNCTION
// - all GPIOs are configured and initialized
//----------------------------------------------------
```

```
int Gpio_Init()
{
    int Status;

    // initialize buttons to inputs
    Status = XGpio_Initialize(&BTNInst, BTN_DEV_ID);
    if(Status != XST_SUCCESS) return XST_FAILURE;
    XGpio_SetDataDirection(&BTNInst, 1, 0xF);

    // initialize switches to inputs
    Status = XGpio_Initialize(&SWInst, SW_DEV_ID);
    if(Status != XST_SUCCESS) return XST_FAILURE;
    XGpio_SetDataDirection(&SWInst, 1, 0xF);

    // initialize LEDs to outputs
    Status = XGpio_Initialize(&LEDInst, LEDS_DEV_ID);
    if(Status != XST_SUCCESS) return XST_FAILURE;
    XGpio_SetDataDirection(&LEDInst, 1, 0x0);

    // initialize MUTE control to output
    Status = XGpio_Initialize(&MUTENInst, MUTEN_DEV_ID);
    if(Status != XST_SUCCESS) return XST_FAILURE;
    XGpio_SetDataDirection(&MUTENInst, 1, 0x0);
    return XST_SUCCESS;
}

//----------------------------------------------------
// TIMER INITIALIZATION FUNCTION
// - initializes the timer and interrupt controllers
//----------------------------------------------------
int Timer_Init()
{
    int status;

    status = XTmrCtr_Initialize(&TMRInst, TMR_DEVICE_ID);
    if(status != XST_SUCCESS) return XST_FAILURE;
    XTmrCtr_SetHandler(&TMRInst, TMR_Intr_Handler, &TMRInst);
    XTmrCtr_SetResetValue(&TMRInst, 0, TMR_LOAD);
    XTmrCtr_SetOptions(&TMRInst, 0, XTC_INT_MODE_OPTION | XTC_AUTO_RELOAD_OPTION);

    // Initialize interrupt controller
    status = IntcInitFunction(INTC_DEVICE_ID, &TMRInst, &BTNInst, &SWInst);
    if(status != XST_SUCCESS) return XST_FAILURE;

    return XST_SUCCESS;
}

//----------------------------------------------------
// INITIAL SETUP FUNCTIONS
//----------------------------------------------------
int InterruptSystemSetup(XScuGic *XScuGicInstancePtr)
{
       // Enable interrupt
       XGpio_InterruptEnable(&BTNInst, BTN_INT);
       XGpio_InterruptGlobalEnable(&BTNInst);

       Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,
                    (Xil_ExceptionHandler)XScuGic_InterruptHandler,
                    XScuGicInstancePtr);
       Xil_ExceptionEnable();

       return XST_SUCCESS;
}
```

```
int IntcInitFunction(u16 DeviceId, XTmrCtr *TmrInstancePtr,  XGpio *BtnInstancePtr,
               XGpio *SwInstancePtr)
{
    XScuGic_Config *IntcConfig;
    int status;

    // Interrupt controller initialisation
    IntcConfig = XScuGic_LookupConfig(DeviceId);
    status = XScuGic_CfgInitialize(&INTCInst, IntcConfig, IntcConfig
       ->CpuBaseAddress);
    if(status != XST_SUCCESS) return XST_FAILURE;

    // Call to interrupt setup
    status = InterruptSystemSetup(&INTCInst);
    if(status != XST_SUCCESS) return XST_FAILURE;

    // Connect timer interrupt to handler
    status = XScuGic_Connect(&INTCInst,
                           INTC_TMR_INTERRUPT_ID,
                           (Xil_ExceptionHandler)TMR_Intr_Handler,
                           (void *)TmrInstancePtr);
    if(status != XST_SUCCESS) return XST_FAILURE;

    // Connect button interrupt to handler
    status = XScuGic_Connect(&INTCInst,
                           INTC_BTN_INTERRUPT_ID,
                           (Xil_ExceptionHandler)PlayerControl,
                           (void *)BtnInstancePtr);
    if(status != XST_SUCCESS) return XST_FAILURE;

    // Connect button interrupt to handler
    status = XScuGic_Connect(&INTCInst,
                           INTC_SW_INTERRUPT_ID,
                           (Xil_ExceptionHandler)TempoControl,
                           (void *)SwInstancePtr);
    if(status != XST_SUCCESS) return XST_FAILURE;

    // Enable button interrupt
    XGpio_InterruptEnable(BtnInstancePtr, 1);
    XGpio_InterruptGlobalEnable(BtnInstancePtr);

    // Enable switch interrupt
    XGpio_InterruptEnable(SwInstancePtr, 1);
    XGpio_InterruptGlobalEnable(SwInstancePtr);

    // Enable button, timer, and switch interrupts in the controller
    XScuGic_Enable(&INTCInst, INTC_TMR_INTERRUPT_ID);
    XScuGic_Enable(&INTCInst, INTC_BTN_INTERRUPT_ID);
    XScuGic_Enable(&INTCInst, INTC_SW_INTERRUPT_ID);

    return XST_SUCCESS;
}
```

## Audio control header

```
/*****************************************************************************
 * File name:           audio.h
 * Description:         Defines function prototypes and necessary parameters
 *                      for audio generation
 *****************************************************************************/
```

```
#ifndef __AUDIO_H_
#define __AUDIO_H_

// Audio controller base address redefinition
#define AUDIO_BASE      XPAR_ZYBO_AUDIO_CTRL_0_BASEADDR

// Slave address for the SSM2203 audio controller
#define IIC_SLAVE_ADDR                  0b0011010

// I2C Serial Clock frequency (Hz)
#define IIC_SCLK_RATE                   100000

// SSM internal registers
enum audio_regs {
    R0_LEFT_CHANNEL_ADC_INPUT_VOLUME    = 0x00,
    R1_RIGHT_CHANNEL_ADC_INPUT_VOLUME   = 0x01,
    R2_LEFT_CHANNEL_DAC_VOLUME          = 0x02,
    R3_RIGHT_CHANNEL_DAC_VOLUME         = 0x03,
    R4_ANALOG_AUDIO_PATH                = 0x04,
    R5_DIGITAL_AUDIO_PATH               = 0x05,
    R6_POWER_MANAGEMENT                 = 0x06,
    R7_DIGITAL_AUDIO_I_F                = 0x07,
    R8_SAMPLING_RATE                    = 0x08,
    R9_ACTIVE                           = 0x09,
    R15_SOFTWARE_RESET                  = 0x0F,
    R16_ALC_CONTROL_1                   = 0x10,
    R17_ALC_CONTROL_2                   = 0x11,
    R18_NOISE_GATE                      = 0x12,
};

// Audio controller registers
enum i2s_regs {
    I2S_DATA_RX_L_REG = 0x00 + AUDIO_BASE,
    I2S_DATA_RX_R_REG = 0x04 + AUDIO_BASE,
    I2S_DATA_TX_L_REG = 0x08 + AUDIO_BASE,
    I2S_DATA_TX_R_REG = 0x0C + AUDIO_BASE,
    I2S_STATUS_REG        = 0x10 + AUDIO_BASE,
};

// Function prototypes
unsigned char IicConfig(unsigned int DeviceIdPS);
void AudioPllConfig();
void AudioWriteToReg(u8 u8RegAddr, u16 u16Data);
void HarmonizerPlay(int noteArray[][ARR_LEN], double beatArray[],
       int tempo, int index);

#endif
```

## Audio control

```
/************************************************************************
 * File name:           audio.c
 * Description:         Function definitions for audio configuration and
 *                      music generation
 ************************************************************************/
#include "intc_handler.h"
#include "audio.h"

//----------------------------------------------------
// Variable definitions
```

```
//----------------------------------------------------
u32 *(baseaddr_mus) = (u32 *)XPAR_ORGANSYNTH_0_S00_AXI_BASEADDR;
int temp = 0;
int volume;
static bool done = false;

//----------------------------------------------------
// I2C CONFIGURATION FUNCTION
// - Initializes the I2C driver and sets the I2C
//   serial clock rate
//----------------------------------------------------
unsigned char IicConfig(unsigned int DeviceIdPS)
{
    XIicPs_Config *Config;
    int Status;

    // Look up the configuration in the config table
    Config = XIicPs_LookupConfig(DeviceIdPS);
    if(NULL == Config) {
        return XST_FAILURE;
    }

    // Initialize the IIC driver configuration
    Status = XIicPs_CfgInitialize(&Iic, Config, Config->BaseAddress);
    if(Status != XST_SUCCESS) {
        return XST_FAILURE;
    }

    // Perform a self-test to ensure that the hardware functions
    // correctly.
    Status = XIicPs_SelfTest(&Iic);
    if (Status != XST_SUCCESS) {
        xil_printf("IIC FAILED \r\n");
        return XST_FAILURE;
    }
    xil_printf("IIC Passed\r\n");

    // Set the IIC serial clock rate.
    Status = XIicPs_SetSClk(&Iic, IIC_SCLK_RATE);
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }

    return XST_SUCCESS;
}

//----------------------------------------------------
// AUDIO PLL CONFIGURATION FUNCTION
// - Performs various functions such as resetting
//   codec software, setting ADC/DAC volumes, and
//   setting sampling rate at 48kHz
//----------------------------------------------------
void AudioPllConfig() {

    // Perform Reset
    AudioWriteToReg(R15_SOFTWARE_RESET,                0b000000000);
    usleep(75000);
    // Power Up
    AudioWriteToReg(R6_POWER_MANAGEMENT,               0b000110000);
    // Default Volume
    AudioWriteToReg(R0_LEFT_CHANNEL_ADC_INPUT_VOLUME,  0b000011111);
    // Default Volume
    AudioWriteToReg(R1_RIGHT_CHANNEL_ADC_INPUT_VOLUME, 0b000011111);
```

```
    AudioWriteToReg(R2_LEFT_CHANNEL_DAC_VOLUME,         0b101111111);
    AudioWriteToReg(R3_RIGHT_CHANNEL_DAC_VOLUME,        0b101111111);
    // Allow Mixed DAC, Mute MIC
    AudioWriteToReg(R4_ANALOG_AUDIO_PATH,               0b000010010);
    // 48 kHz Sampling Rate emphasis, no high pass
    AudioWriteToReg(R5_DIGITAL_AUDIO_PATH,              0b000000110);
    // I2S Mode, set-up 32 bits
    AudioWriteToReg(R7_DIGITAL_AUDIO_I_F,               0b000001010);
    AudioWriteToReg(R8_SAMPLING_RATE,                   0b000000000);
    usleep(75000);
    // Activate digital core
    AudioWriteToReg(R9_ACTIVE,                          0b000000001);
    // Output Power Up
    AudioWriteToReg(R6_POWER_MANAGEMENT,                0b000100010);
}


//----------------------------------------------------
// REGISTER WRITE FUNCTION
// - Allows writing to one of the audio controller's
//      registers
//----------------------------------------------------
void AudioWriteToReg(u8 u8RegAddr, u16 u16Data) {

unsigned char u8TxData[2];

u8TxData[0] = u8RegAddr << 1;
u8TxData[0] = u8TxData[0] | ((u16Data >> 8) & 0b1);

u8TxData[1] = u16Data & 0xFF;

XIicPs_MasterSendPolled(&Iic, u8TxData, 2, IIC_SLAVE_ADDR);
while(XIicPs_BusIsBusy(&Iic));
}


//----------------------------------------------------
// AUDIO GENERATION FUNCTION
// - Receives notes and beats and turn them to audible
//   sounds
//----------------------------------------------------
void HarmonizerPlay(int noteArray[][ARR_LEN], double beatArray[],
    int tempo, int index)
{
    done = false;
    temp = 0;
    u32 out_left, out_right;
    int space = noteArray[index][8];

    // Sends four of the input notes to slave register 0
    *(baseaddr_mus + 0) = 0x00000000 + noteArray[index][0] +
        (noteArray[index][1] << 8) +
        (noteArray[index][2] << 16) +
        (noteArray[index][3] << 24);
    // Sends four of the input notes to slave register 1
    *(baseaddr_mus + 1) = 0x00000000 + noteArray[index][4] +
        (noteArray[index][5] << 8) +
        (noteArray[index][6] << 16) +
        (noteArray[index][7] << 24);

    // Play sound for the duration of the beat
    while (!done)
    {
```

```
        out_left =  (u32)(*(baseaddr_mus + 2));
        out_right =  out_left;

        Xil_Out32(I2S_DATA_TX_L_REG, out_left);
        Xil_Out32(I2S_DATA_TX_R_REG, out_right);

        temp++;

        if(temp == (int)(tempo * beatArray[index]))       {
            // If note break is 1, add a little space in between
            // plays
            if (space == 1) {
                *(baseaddr_mus + 0) = 0x00000000;
                *(baseaddr_mus + 1) = 0x00000000;
                for(int i = 0; i < (space * 225000); i++);
            }
            else {
                *(baseaddr_mus + 0) = *(baseaddr_mus + 0);
                *(baseaddr_mus + 1) = *(baseaddr_mus + 1);
                for(int i = 0; i < 1; i++);
            }
            done = true;
        }
    }
    return;
}
```

## Interrupt handler header

```
/*****************************************************************************
 * File name:           intc_handler.h
 * Description:         Declares interrupt handler function prototypes and
 *                      necessary variables
 *****************************************************************************/
#ifndef SRC_INTC_HANDLER_H_
#define SRC_INTC_HANDLER_H_

#include "init.h"

// function prototypes
void PlayerControl(void *baseaddr_p);
void TempoControl(void *baseaddr_p);
void TMR_Intr_Handler(void *baseaddr_p);

// variable definitions
int btn_value, sw_value;
int note_count;
int tmr_count;

#endif /* SRC_INTC_HANDLER_H_ */
```

## Interrupt handler

```
/*****************************************************************************
 * File name:           intc_handler.c
 * Description:         Defines the interrupt handlers for pushbuttons,
 *                      switches, and timers
 *****************************************************************************/
```

```
#ifndef SRC_INTC_HANDLER_C_
#define SRC_INTC_HANDLER_C_

#include "intc_handler.h"
#include "audio.h"
#include "sheetBeat.h"
#include "sheetNotes.h"

#define LENGTH sizeof(beatArray)/sizeof(beatArray[0])
#define TEMPO 1750000

static int tempo = TEMPO;

//----------------------------------------------------
// PUSHBUTTON INTERRUPT HANDLER
// - Represent music player controller
//   8: Start tune from the beginning
//   4: Pause
//   2: Play
//   1: Forward to next note/rest
//----------------------------------------------------
void PlayerControl(void *InstancePtr)
{
    // Disable GPIO interrupts
    XGpio_InterruptDisable(&BTNInst, BTN_INT);
    // Ignore additional button presses
    if ((XGpio_InterruptGetStatus(&BTNInst) & BTN_INT) !=
            BTN_INT) {
            return;
        }
    btn_value = XGpio_DiscreteRead(&BTNInst, 1);
    if (btn_value == 8)
    {
        XGpio_DiscreteWrite(&LEDInst, 1, 0x8);
        note_count = 0;
        XTmrCtr_Stop(&TMRInst, 0);
        XTmrCtr_Reset(&TMRInst, 0);
    }
    else if (btn_value == 4) {
        XGpio_DiscreteWrite(&LEDInst, 1, 0x4);
        XGpio_DiscreteWrite(&MUTENInst, 1, 0x0);
        XTmrCtr_Stop(&TMRInst, 0);
    }
    else if (btn_value == 2) {
        sw_value = XGpio_DiscreteRead(&SWInst, 1);
        if (sw_value == 8)
            tempo = TEMPO * 1.5;
        else if (sw_value == 4)
            tempo = TEMPO * 1.25;
        else if (sw_value == 12)
            tempo = TEMPO * 2;
        else if (sw_value == 2)
            tempo = TEMPO * 0.75;
        else if (sw_value == 1)
            tempo = TEMPO * 0.5;
        else
            tempo = TEMPO;
        XGpio_DiscreteWrite(&LEDInst, 1, 0x2);
        XGpio_DiscreteWrite(&MUTENInst, 1, 0x1);
        XTmrCtr_Start(&TMRInst, 0);
    }
    else if (btn_value == 1) {
        XGpio_DiscreteWrite(&LEDInst, 1, 0x1);
```

```
        note_count += 1;
    }
    else
        XGpio_DiscreteWrite(&LEDInst, 1, 0x0);

    (void)XGpio_InterruptClear(&BTNInst, BTN_INT);
    // Enable GPIO interrupts
    XGpio_InterruptEnable(&BTNInst, BTN_INT);
}

//----------------------------------------------------
// SWITCH INTERRUPT HANDLER
// - Represent playback tempo controller
//      12: tempo is twice as slow
//              8: 0.5x speed
//      4: 0.75x speed
//      2: 1.25x speed
//      1: 1.5x speed
//   other: default
//----------------------------------------------------
void TempoControl(void *InstancePtr)
{
    // Disable GPIO interrupts
    XGpio_InterruptDisable(&SWInst, SW_INT);
    // Ignore additional switch changes
    if ((XGpio_InterruptGetStatus(&SWInst) & SW_INT) !=
            SW_INT) {
        return;
    }
    sw_value = XGpio_DiscreteRead(&SWInst, 1);
    if (sw_value == 8)
        tempo = TEMPO * 1.5;
    else if (sw_value == 4)
        tempo = TEMPO * 1.25;
    else if (sw_value == 12)
        tempo = TEMPO * 2;
    else if (sw_value == 2)
        tempo = TEMPO * 0.75;
    else if (sw_value == 1)
        tempo = TEMPO * 0.5;
    else
        tempo = TEMPO;

    (void)XGpio_InterruptClear(&SWInst, SW_INT);
    // Enable GPIO interrupts
    XGpio_InterruptEnable(&SWInst, SW_INT);
}

//----------------------------------------------------
// TIMER INTERRUPT HANDLER
// - Contains the music playback function
//----------------------------------------------------
void TMR_Intr_Handler(void *data)
{
    if (XTmrCtr_IsExpired(&TMRInst,0)) {
        if (tmr_count == 2) {
            XTmrCtr_Stop(&TMRInst, 0);
            HarmonizerPlay(noteArray, beatArray, tempo, note_count);
            XTmrCtr_Reset(&TMRInst,0);
            if (note_count == LENGTH - 1) {
                XTmrCtr_Stop(&TMRInst, 0);
                note_count = 0;
            }
```

```
            else {
                XTmrCtr_Start(&TMRInst,0);
                note_count++;
            }
        }
        else tmr_count++;
    }
}

#endif /* SRC_INTC_HANDLER_C_ */
```

## Driver

```
/*****************************************************************************
 * File name:           driver.c
 * Description:         Performs necessary configuration and initializations,
 *                      then starts the timer interrupt
 *****************************************************************************/
#include "intc_handler.h"
#include "audio.h"

//----------------------------------------------------
// MAIN FUNCTION
//----------------------------------------------------
int main (void)
{
    xil_printf("Entering Main\r\n");
    // Configure the IIC data structure
    IicConfig(XPAR_XIICPS_0_DEVICE_ID);

    // Configure the Audio codec's PLL
    AudioPllConfig();
    xil_printf("SSM2603 configured\n\r");

    // Initialise GPIO and NCO peripherals
    Gpio_Init();
    xil_printf("GPIO peripheral configured\r\n");

    // Initialize timer
    Timer_Init();

    while(1);

    return 0;
}
```