Music Score Recognition

Piotr Wilczyński
Student ID: 1465528
BSc Computer Science with Year Abroad

Supervisor: Alan P. Sexton

School of Computer Science
University of Birmingham

UNIVERSITY OF
BIRMINGHAM

April 6, 2018

Abstract

The goal of this project is to explore various approaches to Optical Music Recognition. The developed software processes a given image of a music score and generates a MusicXML format file that can then be imported into a music notation program for edition or playback. The achieved result is a system that deskews a scanned image and handles simple music notation (including chords, double beams, ties). Part of the application is a user interface where processing markings are displayed as the recognizer proceeds.

All software for this project can be found at
`https://git-teaching.cs.bham.ac.uk/mod-ug-proj-2017/pxw328`.

Contents

Chapter 1

Introduction

1.1 Background information

Music Score Recognition is a niche research area that has enjoyed a steady trickle of papers over the years. Although there are already several approaches and solutions to it, there is still large scope for improvement in terms of accuracy and efficiency. Music notation is a very complex notion that makes optical music recognition far more difficult than character recognition.

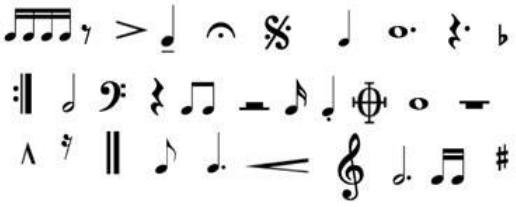

Figure 1.1: Some of the symbols used in music notation

Symbols used in music scores not only vary in shape and dimensions, but also have context-dependent shapes. It means that the same symbol might look different in different situations in order to make the music easier to read and play.

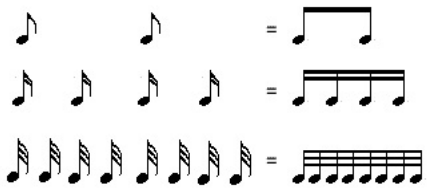

Figure 1.2: Equivalence of tailed and beamed notes

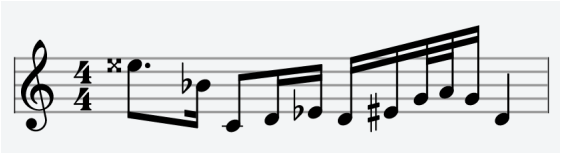

Figure 1.3: Beams that take different forms

Moreover, music scores involve a whole set of grammar rules that define which symbols appear next to others and modify their properties.

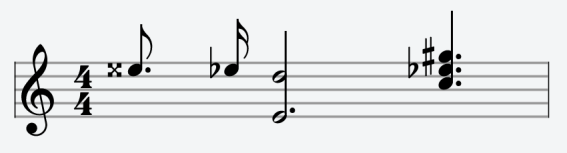

Figure 1.4: Notes whose pitch and duration are altered by accidentals and dots

The system described in this paper aims to address this problem by performing deskewing, line removal, connected component analysis, symbol recognition and music notation parsing on a given image of a music score. It then converts the obtained results into the MusicXML format. The user interface, by displaying the image with markings related to the current recognition stage, allows to inspect the errors made by the system and what caused them.

1.2 Terminology

There is a certain set of musical terms used throughout this paper. Those terms are explained by the following figures.

Note	American English	British English
	Whole Note	Semibreve
	Half Note	Minim
	Quarter Note	Crotchet
	Eighth Note	Quaver
	Sixteenth Note	Semiquaver

Figure 1.5: Note names

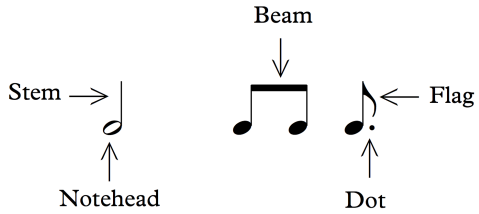

Figure 1.6: Note parts

Figure 1.7: Accidentals: flat, natural, sharp, double sharp and double flat

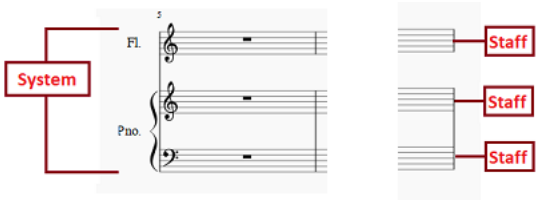

Figure 1.8: Systems and staves

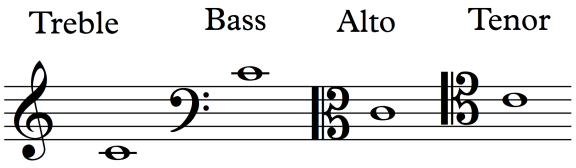

Figure 1.9: Clefs

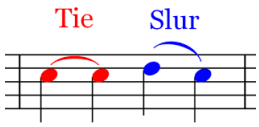

Figure 1.10: Tie and slur

1.3 Previous research

Sheet music has a logical and a geometrical interpretation: it is a time line of events of different lengths but it has to be broken up into pieces (systems) to be printed on paper [14]. It is also comprised of three dimensions: time, frequency and part/instrument. Essentially, it constitutes a set of notes with certain properties. Every part has a stave (five horizontal lines) and a clef that determines the location of pitches on it. A stave is split by vertical lines into bars, each having equally many beats. Bar lines are also used to indicate repetition. Bars are stretched horizontally to fit the page width rather than separated between systems. Empty staves (for voices/instruments that are currently not involved) are usually dropped. There is also textual information on the sheet - the title, composer, copyright etc.

Tardon [27] and almost all the other papers cited in this report state that if the image to be processed has been obtained by scanning, it is first binarized so that there are only black and white pixels left. Since stave lines occupy a relatively large fraction of the page width, they are easy to identify by horizontal projection (counting the number of pixels in every row and considering the highest numbers). According to Fujinaga [7], removing stave lines at the initial stage simplifies the later stages of the image processing. Projection can also be used to straighten the scanned image (by maximising the histogram maxima) [22, 26]. Another method to remove stave lines is to look for connected paths between the left and the right margin [23]. When lines are removed, fragments of them are left in place near components to avoid splitting the components [7]. Later, an algorithm can be run to patch components broken by line removal [1]. The stave space width and stave line height can also be estimated in advance and then used to detect the position and shape of the lines [25, 5].

A completely different approach that have also been researched into is recognition using projections [8]. It means that stave lines are not removed and components, which are whole staves in this case, are projected to see the number of pixels in each row and column. The values of the projection are used to locate and recognize the symbols.

Then, the symbols can be categorized into several classes: ones with fixed positions (clef, time signature), ones with easily identifiable features (vertical lines - could be detected with vertical projection), ones whose location is determined by the location of other symbols (dots, beams connecting quavers) and randomly placed ones (clef changes, ornamentation). At a later stage, filled note heads are distinguished from unfilled ones [7]. One of the features used in symbol recognition is rectangularity - how well a component fits its bounding box. Other features include the position relative to stave lines (bass clef extends from the first to the fourth stave line) and the number of peaks in a projection (a sharp has two peak in a vertical projection, a flat only one) [8].

In Rebelo's [23] and Rossant's [24] systems, symbols are assigned different labels so that it is possible to distinguish them at later stages. There are four main types of labelling algorithms: multi-scan, two-scan, hybrid and tracing-type [12]. One of the two-scan algorithms that have been researched is the run-based two-scan labelling algorithm [11]. Labelling is performed by traversing the image with a mask and looking at 8-connectivity. In the end, symbols are provided with bounding boxes so that their dimensions are

known. Some of them, such as clefs, will have sizes relative to the stave. During the recognition, symbols can be considered in their original form or with the stave lines removed. Algorithms used for detection include neural networks, statistical classifiers and classification trees. Some components, like chords or beams, can be recognized using finite-state machines [21]. The specific architecture involved in neural network recognition is called multilayer perceptron [22]. Moreover, a data set has to be collected that is then split randomly into a training set and a test set.

Music Recognition is essentially about constructing a logical model representing the notation on the sheet [22]. Once this has been done, the model can be automatically converted into an audio file [27].

Due to the logical structure of sheet music, its recognition is much more difficult than alphanumeric character detection [2, 13]. There are symbols related to several other symbols (signatures, accidentals, accents) and some symbols might take different shapes (joined quavers) [16]. Sheet music is also characterized by overlapping symbols and zones of high symbol density [22].

There are multiple ways of testing a recognition algorithm, e.g. by considering the number of wrong pitches, wrong lengths, added notes, missing notes etc. It is also possible to evaluate the recognizer by comparing MusicXML files, however this method is not ideal as the same music score might be represented by multiple MusicXML files [23]. For example, the order of notes in a chord does not matter for a MusicXML parser.

During the recognition, errors can be avoided by analyzing the music notation grammar [23, 4] and rhythm [15]. For instance, after recognizing a symbol that does not fit in with the previously recognized symbols, the system can backtrack the symbols and determine what symbol should be expected next. There are certain graphical rules that relate to the music notation, for instance accidentals are positioned before their corresponding note heads and they are in the same vertical position. Dots are above or after note heads and within a certain distance and the key signature is located right after the clef. The time signature can be used to verify the correct duration of the notes in a bar [24].

1.4 Rationale for further research

A lot of improvement can still be achieved in Optical Music Recognition [14]. For instance, recognizing complex musical symbols and elaborate sheet mu-

sic is a difficult task and the current approaches do not deal with it as effectively as with simple music notation [6]. As regards scanned images, the effects of illumination, distortion and inclination are still challenging in image processing [18]. More research can also be done into handwritten music recognition [22] and old music recognition to ensure its preservation [27]. Another related problem is making a music recognition system scalable and work across different platforms [3].

Rossant [24] claims that a large number of approaches to Optical Music Recognition are limited to graphical rules but do not analyze music syntax in terms of which components are likely to appear in close proximity on music scores. Additionally, most of the literature in the area is quite general, meaning that the methodology is described on a high level but exact algorithms are not provided. For instance, some papers say that vertical lines are removed or that beams are recognized by region growing, but a lot of problem solving and experimenting remains to be done in order to implement the method on one's own. By contrast, this report attempts to provide explanation and examples detailed enough for anyone to be able to implement the whole system themselves from scratch.

1.5 Limitations

The purpose of this project is to analyze some of the algorithms and methods used in Optical Music Recognition. There are a large number of methods, either mentioned or not mentioned in this paper, that have not been explored due to the time constraints of this project. The main intention was to make the whole system work fully and as effectively as possible and the design decisions were heavily influenced by how well the chosen methods performed on the test set of music scores.

The system described in this paper does not process handwritten or obsolete music notation. The focus was standard notation used nowadays. It does also not process all music notation, for instance dynamic markings or two-column chords.

Chapter 2

Specification

The software is all written in Java and part of it is a user interface that allows to load an image and to inspect the recognition process. The input of the program is an image file of a music score, either scanned or electronic, of a resolution of at least 300 DPI. The supported image file extensions include PNG, TIF, TIFF, JPG, JPEG, BMP and PBM. The user interface allows to display the processing done by the program on the image and inspect the errors. The output is an XML file defining the music on the sheet. An external music notation software is needed (e.g. *Finale*, *Sibelius*) for edition and playback.

2.1 Users

The intended users of this application are:

- Beginner instrument learners - being able to listen to a music piece makes it easier to learn to play it.
- Composers wanting to edit a music score or arrange it for a different instrument - they will not have to input the whole music score into music notation software manually.
- Anyone wanting to listen to a specific music score or to a different arrangement of a music score of which there is no available online recording.

Chapter 3

Design

The image processing is split into several stages as that is the usual approach to Optical Music Recognition. Every stage can be separately displayed and debugged on the user interface.

3.1 Deskewing

An algorithm similar to Stochastic Gradient Descent is used to find the angle by which the image has to be rotated. It means that it is rotated by a very small angle until the error stops decreasing and the optimum angle is found. The direction of rotation also has to be determined.

The first challenge is to come up with a good error function. Horizontal projection is needed to see the number of pixels in each row. There has to be at least five peaks in the projection that correspond to five stave lines.

Some of the functions tested were the difference between the minimum and the maximum value in the projection or simply the maximum value. That did not work as sometimes the maximum value would increase even though the image was rotated in the wrong direction.

A much better function proved to be the number of visible peaks in the projection. A threshold of half of the image's width was placed on the projection to pick out the peaks. The image is rotated in one direction until the number of peaks starts decreasing or an angle of 3 is reached. If, after that stage, the number of peaks is still smaller than five, the direction is flipped. After flipping the direction, the image is rotated until the number of peaks is at least five and starts decreasing.

That function was still not ideal as on some images all the stave lines are already visible as peaks in the projection even if the image is still slightly skewed. An even better function is the sum of the peaks in the projection.

Different step sizes were tried to find the largest value for which the algorithm works (larger step sizes make the algorithm faster). The step size of 0.05 degrees was found the optimum.

Binarization was not part of this project, however simple pixel thresholding is done in order to be able to process some scanned images.

There are multiple ways of rotating the image. For example, the image can be binarized beforehand and then simple matrix rotation could be used, however using the built-in function to smoothly rotate a greyscale image proved to cause less distortion.

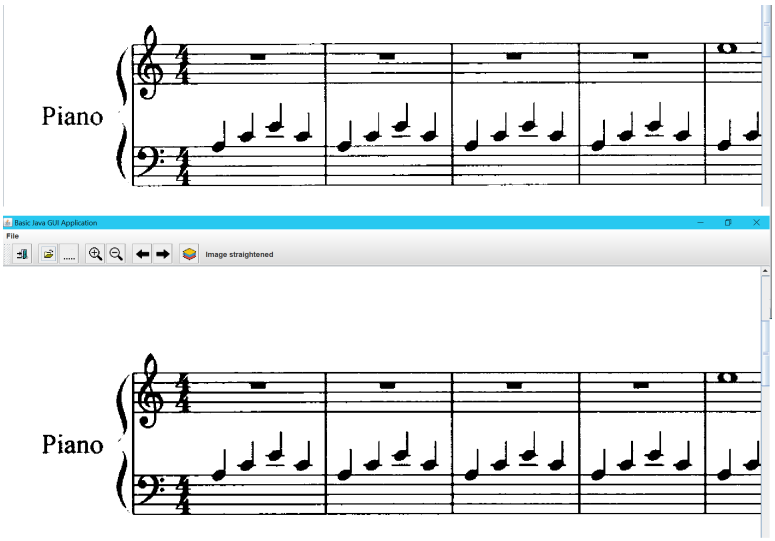

Figure 3.1: The top image was first binarized and then rotated using matrix rotation. The bottom image was rotated smoothly as a greyscale image and then binarized. The bottom one appears to have smoother lines

An important thing to note is that the image should always be rotated from the original image. Otherwise, it becomes more blurred/distorted with every rotation.

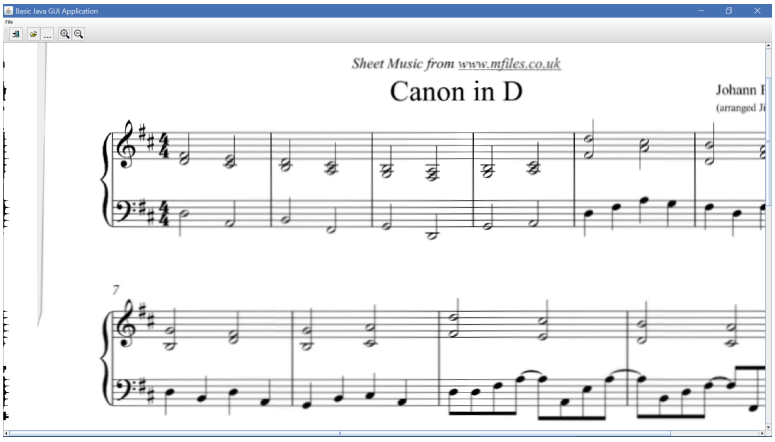

Figure 3.2: An image that has been rotated by a small angle multiple times

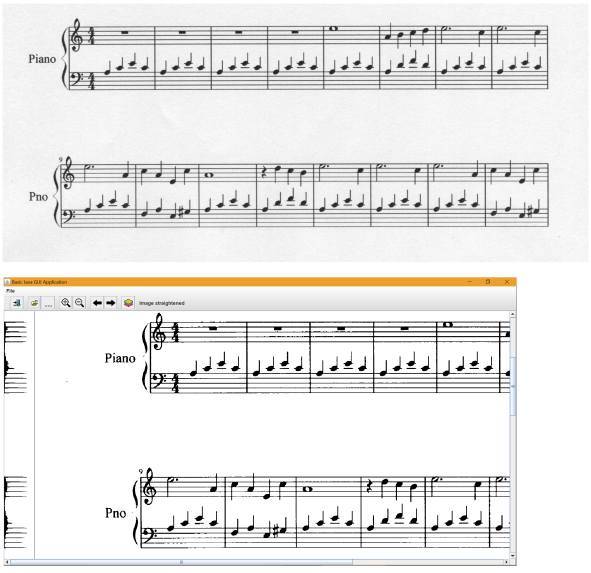

Figure 3.3: A scanned image and its deskewed version along with the projection where peaks are at stave line level

3.2 Stave removal

Since it turned out relatively easy to remove the staves and it seemed to make recognition more straightforward, it was decided to follow this approach. As described in most of the references cited in this paper, stave removal is done using horizontal projection.

The simplest idea is to remove all the pixels in rows where there are peaks in the projection. However, this does not remove whole lines due to distortion caused by the image being scanned.

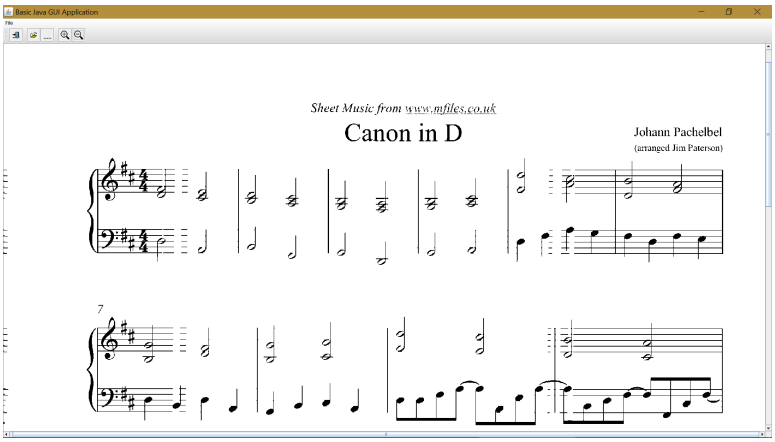

Figure 3.4: Stave lines not removed in full

A much better idea is to remove extra pixels above and below. A stave line is defined by a number of consecutive values above the threshold in the projection. Let n be the vertical coordinate of the top of the line and m the vertical coordinate of the bottom of the line. All the pixels between (n-1) and (m+1) are removed, however only if (n-2) and (m+2) are white. The condition is added so that symbols on the music score are not divided. Later, the following turned out to be a better method: the pixel is next to a component if (n-1) and (m+1) are black; the pixel is next to a component if (n-2) or (m+2) is black; if the pixel is not next to a component, all pixels from (n-2) to (m+2) are removed.

It is useful to find the beginning of the stave line as some scores do not have an initial bar line. The first pixel after which at least 5 consecutive pixels are removed is taken to be the beginning of the stave line. It is not

just the first removed pixel as the first removed pixel might belong to the brace on the left of the stave.

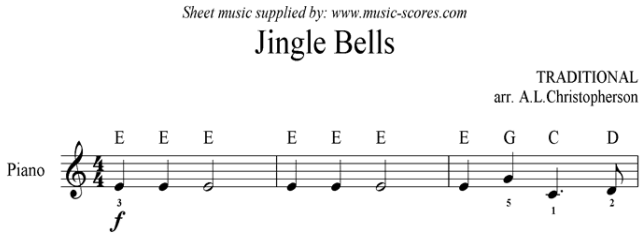

Figure 3.5: Stave without the initial bar line

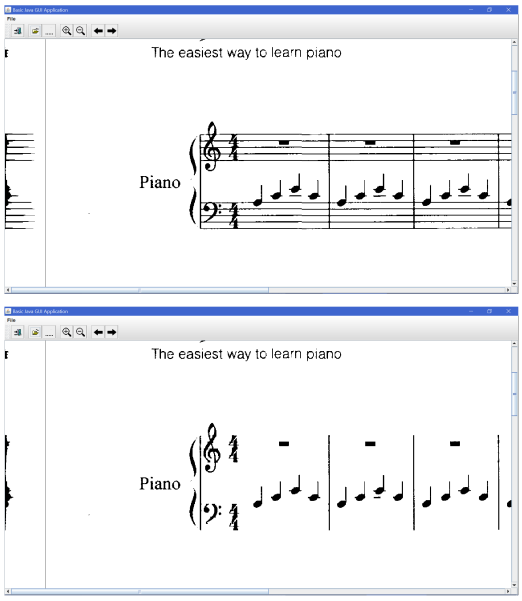

Figure 3.6: Stave removal

The average distance between two adjacent stave lines, which is the same as stave space width (SSW), is used as a measure in later stages.

3.3 Vertical line removal

The approach examined in this project involves removing all vertical lines. This is done so that it is easier to recognize note heads using their dimensions and to separate notes connected by beams. Moreover, with this approach, chords can be split into single note heads using the stave space width.

Figure 3.7: Beamed notes

To properly remove vertical lines, it is important to determine the width and height of every line. Simply removing every few pixels that appear in a column would remove almost all pixels from the image.

Since scanned lines are distorted (they are not perfectly straight, there are extra pixels on the sides, there might be a pixel gap), tracing pixels is not enough to find a line's dimensions. Initially, a 2x2 mask was used to trace the line. A position would still be considered part of the line if there was at least one black pixel in the mask. The image was scanned column by column and width was found first. Then, to determine the height, a width by 1 mask was moved above and below. If there were at least 0.8*height pixels in the mask, the height was incremented.

The reason why the mask was not just moved down but also up is that the first scanned pixel of the line might not lie at the top of it.

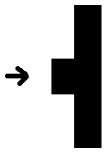

Figure 3.8: First pixel in the middle of the line

The next challenge is that note heads are damaged as they have a number of columns of pixels inside them.

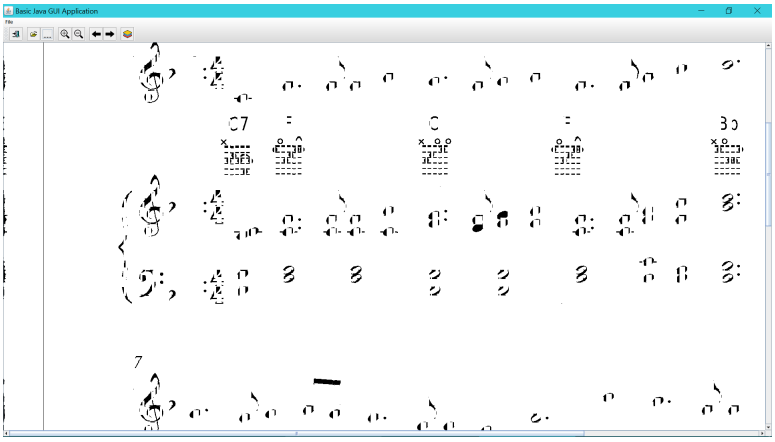

Figure 3.9: Damaged note heads

A solution seems to be to increase the minimum height of a line. However, chords of three or more notes are as high as the smallest line that should be removed.

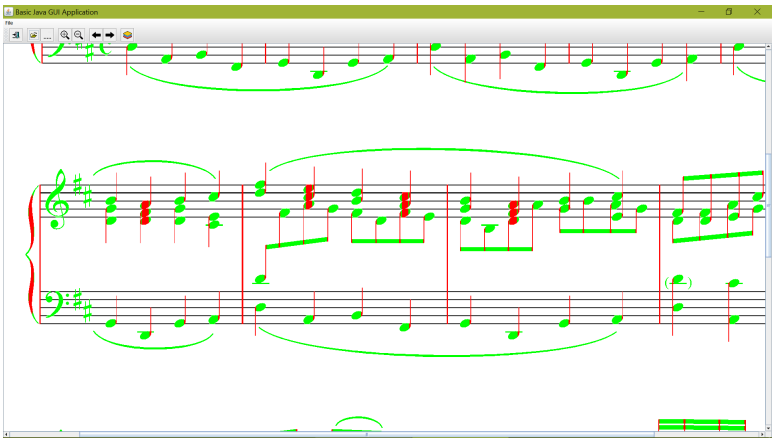

Figure 3.10: Damaged chords of three notes

A better solution might seem decreasing the maximum width of a line. This, however, still does not stop the algorithm from breaking note heads.

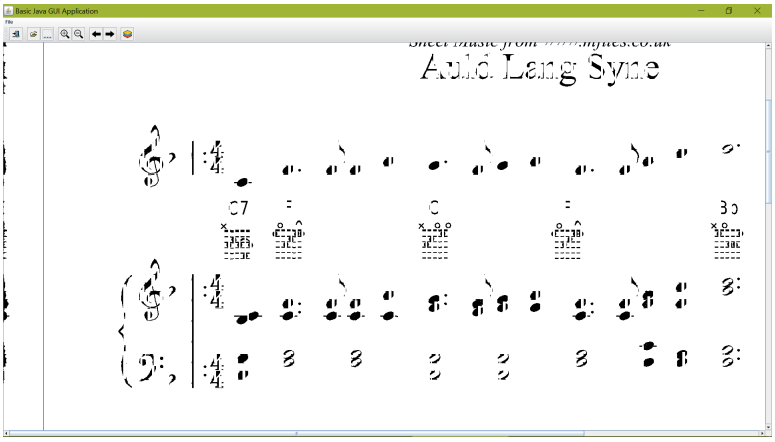

Figure 3.11: Damaged note heads

Another idea was to mark pixels as visited when looking for vertical lines and to not process the visited pixels. This brings other issues: the left line shown below would be removed as two separate lines. Later on, it will not be obvious that there is one line connecting the beam and the bottom note head. The right line might be measured as too short to be removed.

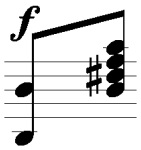

Figure 3.12: Beam

What achieved the desired result is an idea that was already used for stave removal. When a vertical line is removed, pixels are not removed if there is a pixel on the left or right of the line.

There were still cases when it did not work. Since some of the pixels in a distorted line might have been shifted to the side, tracing the line to determine its dimensions was difficult.

Figure 3.13: Part of a line left

It turned out that it is better to scan the image row by row instead. This is because a line is only a few pixels wide and rotation and binarization is much more likely to push pixels out of the line sideways rather than up or down. When scanning the image column by column, pixels sticking out to the left will be first found even though they are not at the top of the line.

Since the image is scanned row by row, width is found first, then height, then width again to the left and right. This is needed due to distortion and other components (like crotchet rest) containing columns of pixels in them. There has to be at least 0.6*height pixels in the column for it to extend the line's width. The initial width is assumed to be 2 pixels because if the top of the line was shifted to the side, the height would not be determined correctly. The line is not removed if its width is above SSW/3.

When the height is being found, the line is traced down until at least SSW pixels are off (1 or 2 pixels away from the width by 1 mask being moved down to find the height) or there is a gap (no pixels) of at least SSW/6 pixels. This is because some lines are distorted and pushed to the side.

Figure 3.14: A line whose height would not be determined correctly without initializing the width to 2 pixels (left) and a line that is removed even though it is damaged and distorted (right)

The pixels in the line are not removed if either of the following conditions holds: the first and second pixels to the left are black; the first and second pixels to the right are black. 2 extra pixels to the left and right are removed to wipe distortion.

Crotchet rests are split by vertical line removal. To tackle that, pixels are removed from a vertical line only if there are at least SSW/2 consecutive pixels to be removed. On the other hand, that causes parts of stems not to be removed if they are crossed by ledger lines. The additional condition is that if the vertical line is a stem (height of at least 3*SSW), there has to be at least SSW/4 consecutive pixels to be removed for them to actually get removed.

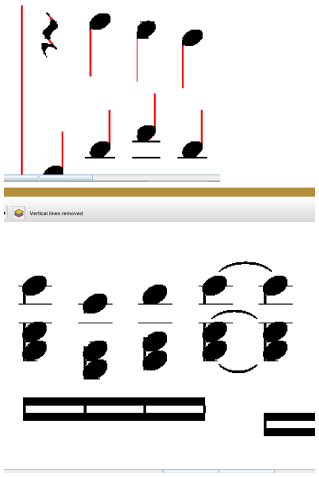

Figure 3.15: A crotchet rest that was damaged by vertical line removal (top image) and parts of stems that did not get removed because there were not enough consecutive pixels to be removed (bottom image)

Vertical projection cannot quite be used to locate vertical lines globally. This is because, unlike stave lines, they are relatively short and will not create peaks in the projection. Those peaks might as well be created by things that are not vertical lines.

Originally, the height threshold for vertical lines was 3 SSWs so that only bar lines and note stems were removed, but notes above or below stave lines have shorter stems - as short as flats and sharps. Some experiment was done not to remove vertical lines from flats and sharps. One idea was to check if there are pixels on both sides of the line or just one, but quavers and sharps have all pixels on both sides. It was decided that accidentals will be recognized with vertical lines removed. The horizontal beams of sharps and naturals are recognized separately.

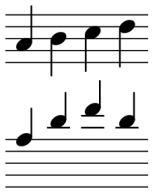

Figure 3.16: On-stave and off-stave notes. The bottom notes have shorter stems as they are not on the stave

3.3.1 Bar line detection

It is useful to detect bar lines in advance so that it is known how many staves there are in a system. Since a bar is the most high-level unit in MusicXML, it also makes it easier to convert all the notation into this format.

The simplest bar line recognizer looks for a path of connected pixels from the first to the fifth stave line. However, if there are multiple staves in a system, bar lines extend all the way from the top of the first stave to the bottom of the last stave. This allows to determine the number of staves in a system. Thus, a bar line is any vertical line that starts at the top of a stave and ends at the bottom of a stave.

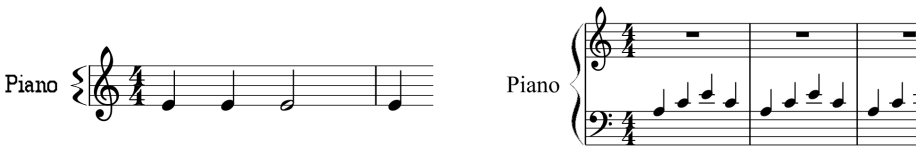

Figure 3.17: One-stave system vs two-stave system

Some note stems also cover the whole stave and might be misrecognized as bar lines. It is checked how many pixels there are around a supposed bar line. If there are enough for a note head or a beam, the line is not considered a bar line any more. There might be a tie or a slur crossing a bar line, but

the number of pixels in a tie/slur around a bar line is much lower than in a note head or a beam.

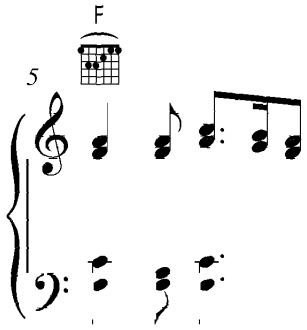

Figure 3.18: Note stems wrongly removed as bar lines

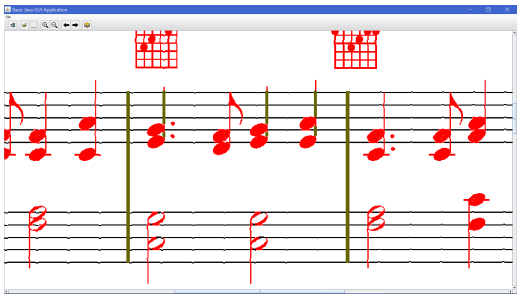

Figure 3.19: Note stems wrongly recognized as bar lines

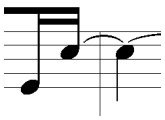

Figure 3.20: Bar line crossed by a tie

It is assumed that the first bar line extends across all the staves and it is used to determine the number of staves per system. If there is no first bar line, it is assumed that there is only one stave per system.

3.4 Patching

Stave and vertical line removal split some of the components into several smaller ones making the recognition more difficult. Most affected are minims and semibreves located between stave lines - removing staves also removes some of their tops and bottoms. This happens because there are some pixels that do not have any pixels above or below them. Following the idea of Fujinaga [7] and Bainbridge [1], a patching algorithm was implemented as part of this system.

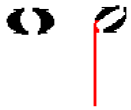

Figure 3.21: Damaged semibreve and minim. The red part of the minim is removed as a vertical line

Those damaged components have one common feature - they all consist of two arcs curved in the opposite direction. That feature can be used to make an algorithm that finds those arcs and connects them back together.

It might seem an idea to move an image of a minim/semibreve through the image to find their occurrences on the music score by comparing pixels and then fix those occurrences. This is not feasible as moving a mask across the whole image that compares all the pixels in it is very slow.

A blur and sharpen algorithm was tested for connecting broken lines. It creates a matrix with the dimensions of the image initialized to all 0s. For every pixel in the image, the number 2 is added to its corresponding cell in the matrix and the number 1 is added to adjacent cells. At the end, only values 2 are translated back into black pixels. It did not manage to repair the lines; it made the components less sharp and therefore more difficult to recognize in the last stage.

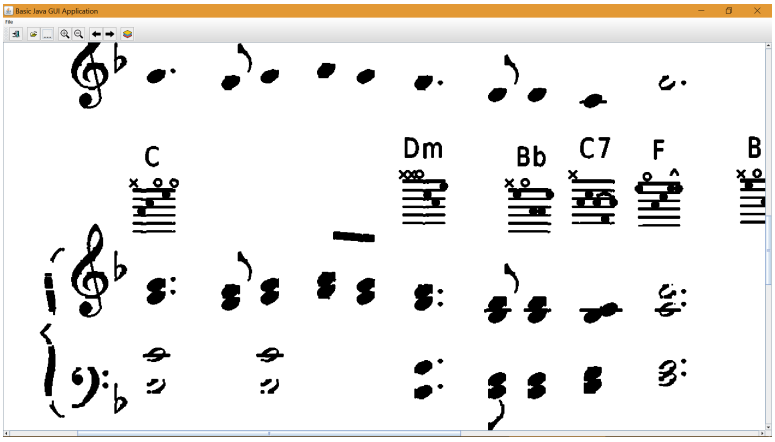

Figure 3.22: Blur and sharpen

Another idea was to leave some pixels to the left and right of the ones near components when removing staves. That, however, left too many pixels on components and left, for example, sharps and notes connected as can be seen in the figure.

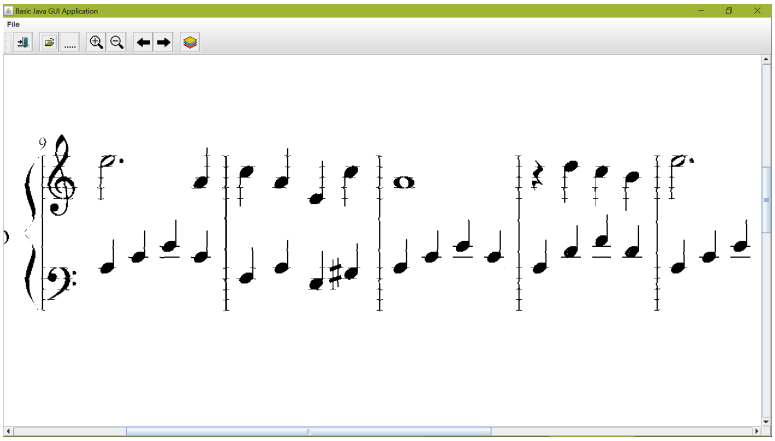

Figure 3.23: Stave removed leaving extra pixels around components

An algorithm was invented that moves along every stave line and finds pairs of paths from this stave line to the next. It then links those paths if they are close enough and slightly curved. Its simple version does the patching, but there are a number of cases where it stitches together components that should not be connected.

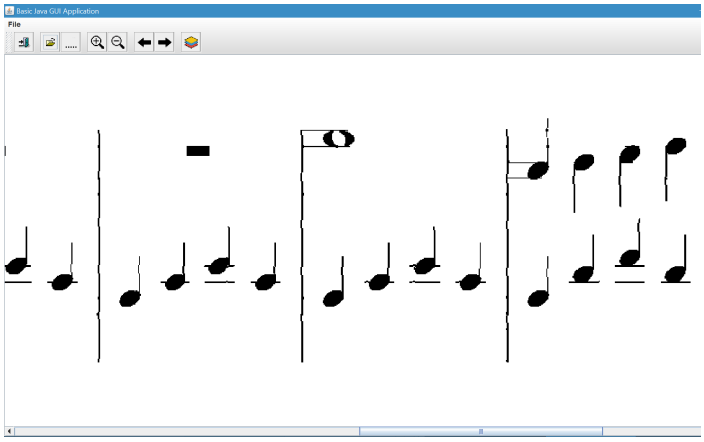

Figure 3.24: Simple patching algorithm

The way the algorithm works is that it keeps moving down from one stave line to the next. At every step, it goes to the pixel directly below if it is black, otherwise it goes down diagonally. One idea for dealing with edge cases was to check whether the left path has at least some number of left diagonal moves and the right path has at least some number of right diagonal moves, however the paths found on parts of a minim are both curved leftward. Moreover, some semibreve halves just have straight lines and the paths found will have 0 diagonal moves.

Figure 3.25: Semibreve with two paths found

Some solutions were tried to distinguish semibreve halves from other components, such as looking at the surface of the component (the surface can be obtained by counting pixels to the left and right when tracing the path), however it did not work well as some components have similar surfaces. Eventually, it was decided not to patch up semibreves and to recognize their halves separately.

Some experiment was done to decide whether patching should be done before or after vertical line removal. Some minims are damaged by vertical line removal so much so that there is no longer a path between stave lines.

On the other hand, removing vertical lines after patching will damage the components again. A solution to that turned out to be using the current (damaged) image to trace stave lines and find paths' starting positions, but using the previous image (with vertical lines) to trace the paths - that way holes are located after line removal but components are identified based on what they looked like before.

Figure 3.26: Damaged minim one part of which no longer connects two stave lines

To prevent the algorithm from connecting notes to accidentals or bar lines, it is checked whether there is a vertical line going from the end of the path to the previous stave line. Sharps and flats have vertical lines in them and so the algorithm will not stitch them to note heads any more. If that vertical line is part of a vertical line that is at least 3 SSWs long, it is probably part of a stem and so the note is patched up anyway. Additionally, nothing at the beginning of the stave is patched up so that clefs do not get connected to anything.

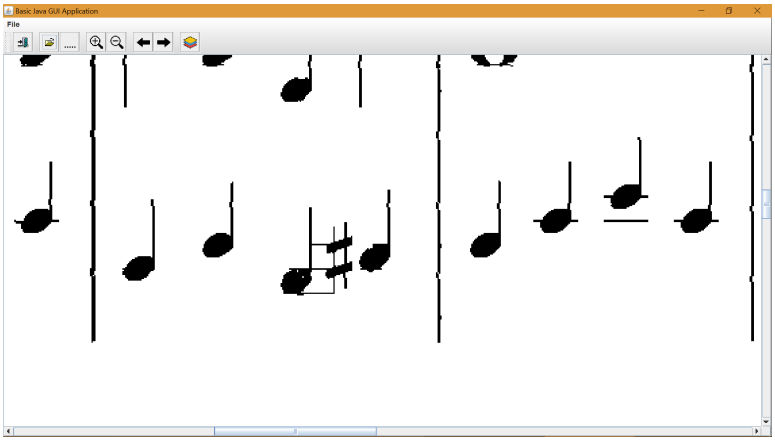

Figure 3.27: Sharp connected to a note

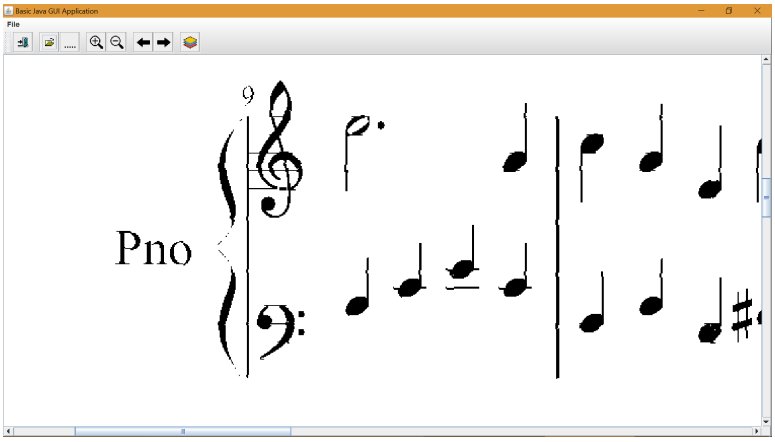

Figure 3.28: Clef connected to the initial bar line

3.5 Labelling

A connected component is a set of pixels in which any two pixels are connected by a path. Diagonal moves are allowed in the path. Connected Component Analysis, also used by Rebelo [23] and Rossant [24], was implemented to label all the components.

The image is scanned row by row with the mask showed in the figure. Two pixels are considered to belong to the same component if they have a common edge or corner. If the current pixel in the mask is black but the rest are white, a new label is assigned to it. If the current pixel is black but some of the other pixels are also black, one of the labels of the rest of the pixels is assigned to the current pixel.

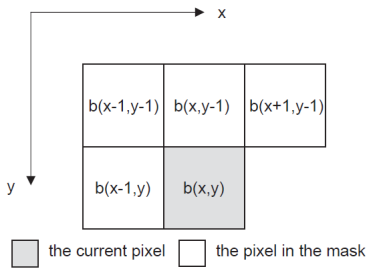

Figure 3.29: CCA mask

Whenever any of the pixels in the mask have different labels, that information is recorded as label equivalence. It means that the two or more labels are actually the same. This has to be done due to the fact that multiple labels might be assigned to the same component. For instance, the bottom part of the treble clef shown in the figure is not horizontally connected to the rest of the component. Therefore, when scanning the image row by row left to right, there is a spot where one of the pixels of the symbol is the current pixel in the mask and there are no other pixels.

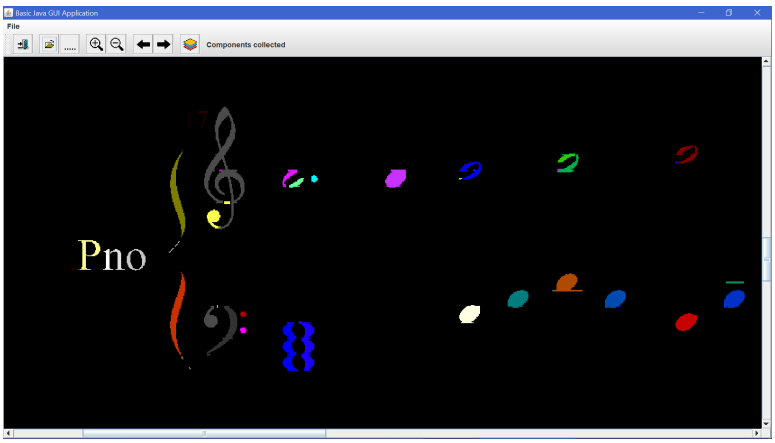

Figure 3.30: Labelled components

The next step of Connected Component Analysis is resolving label equivalences. The first label assigned to a component is its main label and the other ones are replaced with it.

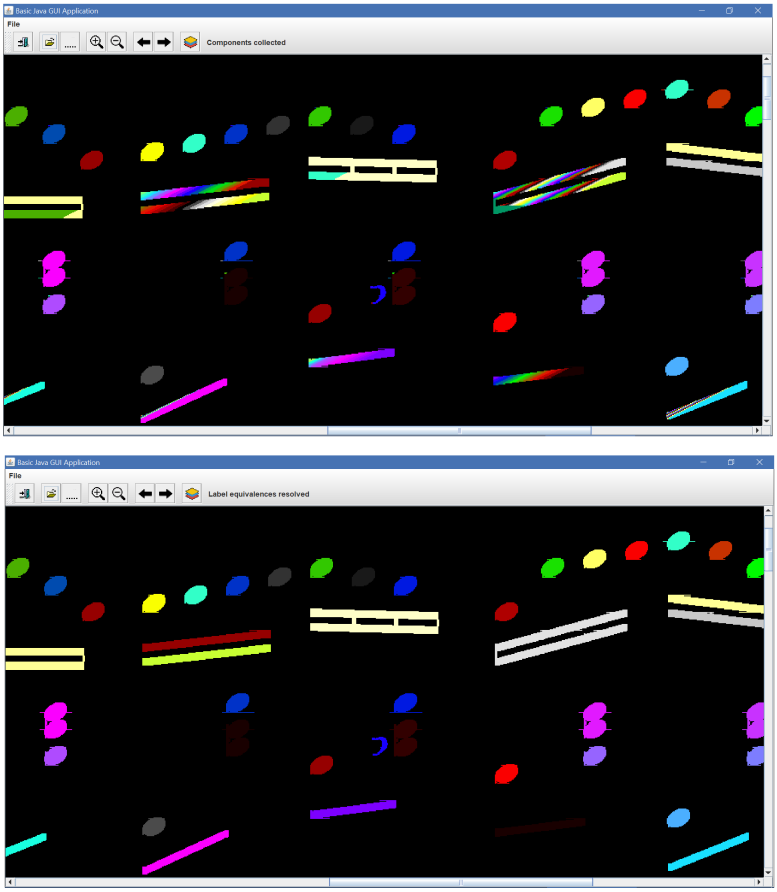

Figure 3.31: The image before and after label equivalence resolution

Lastly, bounding boxes are drawn so that the dimensions of each component are evident.

Figure 3.32: Bounding boxes

3.6 Symbol recognition

In this stage, the previously drawn bounding boxes are coloured to indicate what they were recognized as.

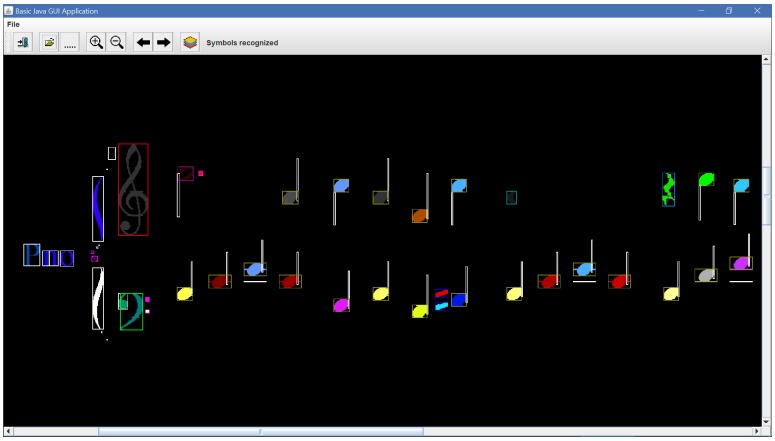

Figure 3.33: Coloured bounding boxes

All components that are out-of-bounds (on the side of staves or too far above/below) are discarded. Since a large number of symbols have similar graphical features, decision trees are used for symbol recognition. For instance, some symbols have similar dimensions or ratios of black to white

pixels. Beams, although they come in many different shapes, all have a number of lines going from the left to the right of their bounding boxes.

Recognition is split into two stages: in the first run through the components, auxiliary symbols are recognized (dots, beams, tails, ties, slurs).

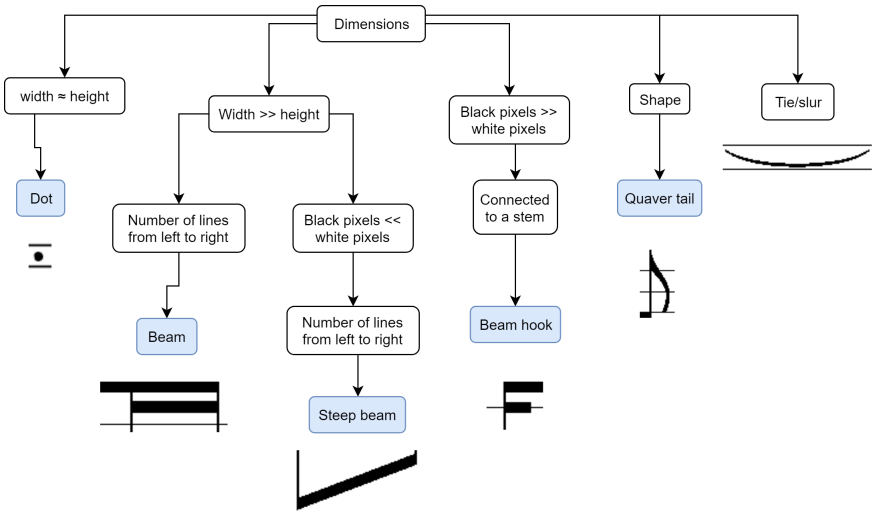

Figure 3.34: Initial decision tree

In the second run, the program loops through every system, every measure and every stave, determines the bounds of the current region (left and right: bar lines, top and bottom: the middle between two staves), collects the components located in the region and sorts them by X position. This is to make sure that all symbols will be added into the XML file in the order they appear on the music score. Symbols recognized in the second run are clefs, notes, rests, metre and accidentals.

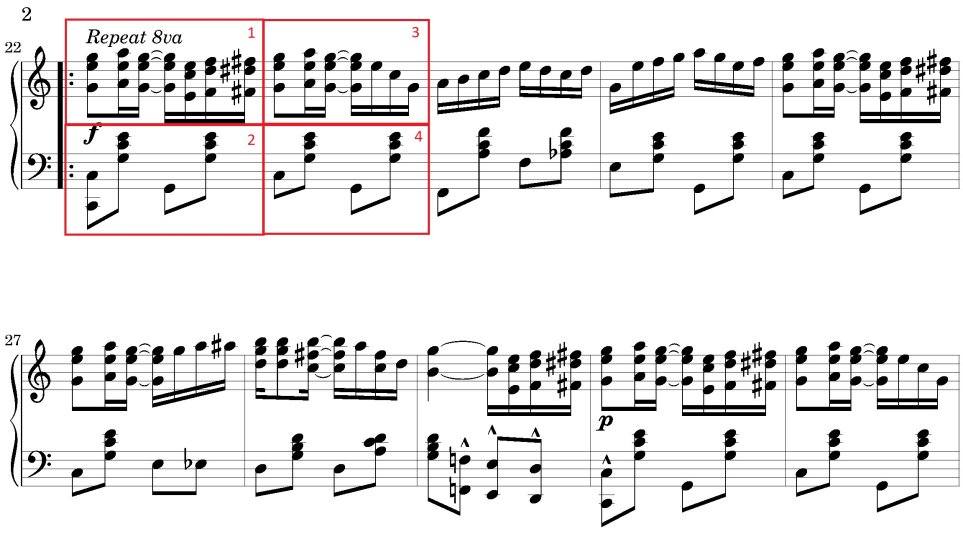

Figure 3.35: The regions that scores are split into. This structure is determined by the syntax of MusicXML and the order in which musical notes should be played

At the top of the decision tree, symbols are split into groups based on their dimensions. That method was also described by Fujinaga [8]. For instance, the treble clef is 6 SSWs wide and 2 SSWs high and the bass clef is 3 SSWs wide and 2 SSWs high. The height of a node head is always 1 SSWs as one of its possible positions is exactly between two stave lines. When checking if a symbol has certain dimensions, ranges are used instead of exact values as some of the symbols may vary in size from score to score and small parts of it might have been cut off by line removal. So, for a symbol to have the width of a treble clef, its width would have to be between 6 and 8 SSWs.

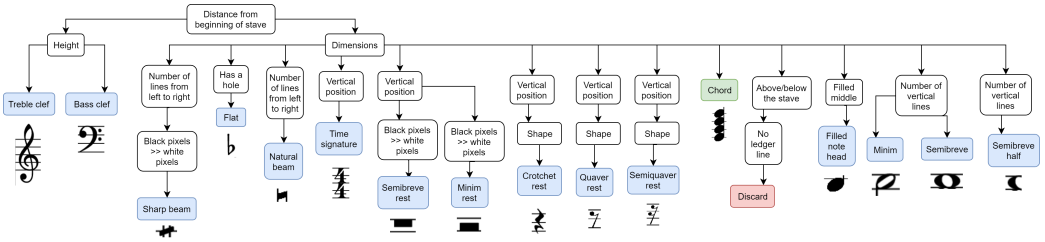

Figure 3.36: Further decision tree

Further division into groups involves examining the ratio of black to white pixels of a component (as was also done by Fujinaga [7]) and splitting a component into a grid and comparing the number of pixels in each cell to the symbol's image stored in the training set.

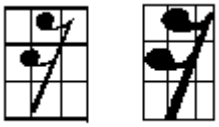

Figure 3.37: Grid comparison - one symbol is from a music score and the other one is stored in the training set of symbols. The number of pixels in each cell has to be similar for the symbols to be considered the same

One idea for recognizing note heads was checking whether they are on or between stave lines. However, the distance between a stave line and the middle of a stave space is very small and, since verifying the symbol's position cannot be exact due to noise and distortion, it is not worth following this method.

One thing to note is that note heads' bounding boxes have variable width due to the potential presence of ledger lines.

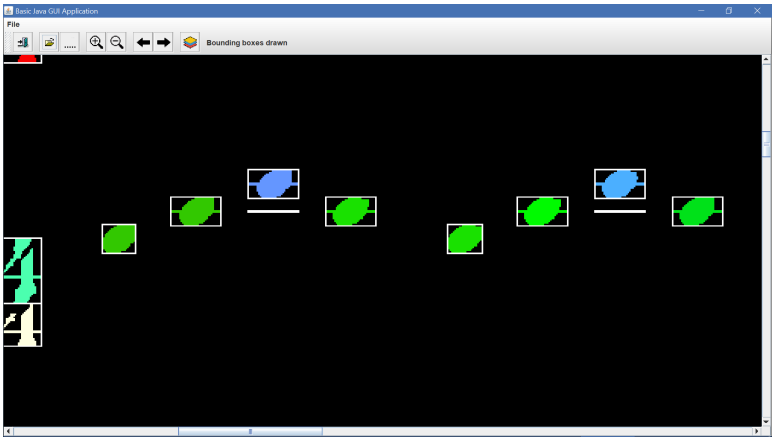

Figure 3.38: Note heads with and without ledger lines. Note heads with ledger lines have wider bounding boxes

To distinguish between crotchet and minim note heads, the methods tried were ratio of black to white pixels and grid verification. Neither of them proved to work well as the number of pixels in note heads varies too much due to distortion. What hit the nail on the head is checking the number of pixels in the middle - there will always be many fewer pixels in the middle of an unfilled note head.

Figure 3.39: Minim and crotchet note heads

Distinguishing between minims and semibreves is quite straightforward as semibreves are never connected to a stem. However, an additional feature that can be used is that semibreves have two vertical lines extending all the way from top to bottom and minims do not have any vertical line. Peaks in the projection are used to find the number of vertical lines [8].

Figure 3.40: Semibreve vs minim. It can be noticed that a semibreve has two vertical paths connecting its top to its bottom

Figure 3.41: Vertical projection can also be used to distinguish between flats and sharps. A flat has one vertical line and a sharp has two

When looking for peaks in the projection, two peaks very close to each other are considered the same peak. This is because a peak is defined to be a series of equal values greater than the adjacent values and holes in components might split a peak into two.

When notes are processed, first their pitch is determined by their vertical position, the clef and the current pitch alterations. All the possible vertical positions relative to the current stave are encoded as integers and looped through. Pitch information stemming from the clef and the key signature is stored throughout the stave and pitch alterations from accidentals is stored throughout the measure. Next, previously recognized dots, beams and tails are looped through to check whether any of them is in the position that indicates a relation between the two symbols. For example, a dot close to a note on its right extends its duration.

As mentioned earlier, beams have a number of lines going from the left to the right edge of their bounding boxes. All possible line equations are checked and if for any slope the number of lines is greater than 1/3 * the height of the bounding box or SSW/4, the component is recognized as a beam.

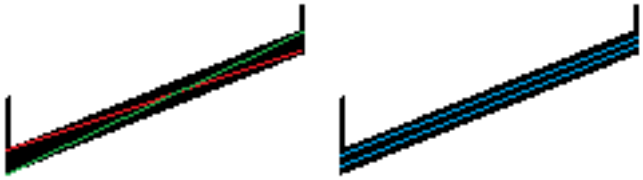

Figure 3.42: Different lines going through a beam. Along the actual slope of the beam the number of lines will be the greatest

There are two types of beams in the recognition stage: single beam and double beam that is still one component.

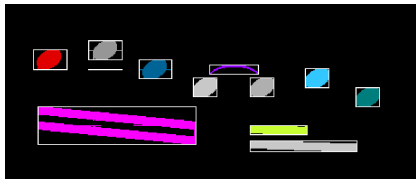

Figure 3.43: Double beam as one component and as two components

When determining whether the note is a beamed quaver or semiquaver, all the beams and vertical lines are looped through. Originally, only the distance between the note head and the beam was checked (without looking for the connecting stem), but sometimes that distance is quite large (larger than the distance between two staves).

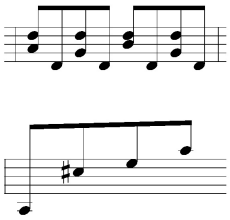

Figure 3.44: Very long stem between a note head and a beam

In the final version, it is checked whether there is a vertical line connecting the note head to a beam. Some ties/slurs have the same dimensions as beams; this method also prevents ties/slurs recognized as beams to actually affect any notes. If the number of such beams is 1, vertical projection is used to check if there is a double beam related to this note. The values in the vertical projection are larger where the beam is double. The vertical lines connecting note heads to beams are highlighted for debugging purposes.

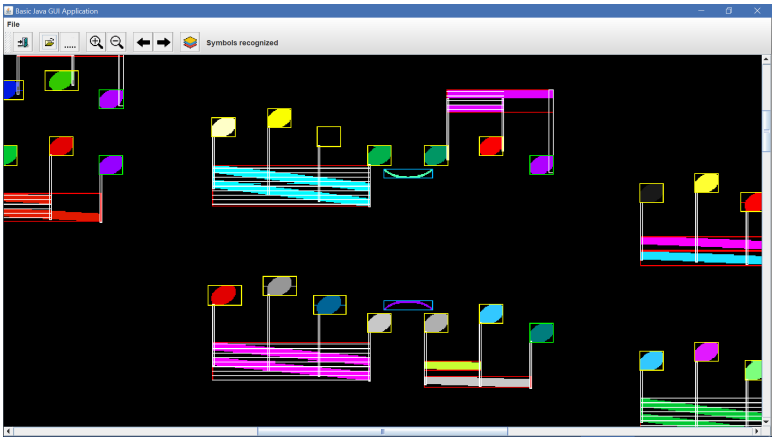

Figure 3.45: Highlighted stems

To distinguish between minim/semibreve rests and sharp beams, the ratio of black to white pixels is used. Sharp beams are sloping and so they have some white pixels in their bounding boxes. Beam hooks are easily distinguished from all those symbols because there has to be a stem connecting the note head to the beam.

Figure 3.46: Minim rest vs semibreve rest vs sharp beam vs beam hook

One of the encountered errors was that a note head fell inside a beam's bounding box and modified the beam's vertical projection so that there appeared to be a double beam at that place. The fix was to look only at the pixels labelled with the label of the component for which the bounding box was drawn.

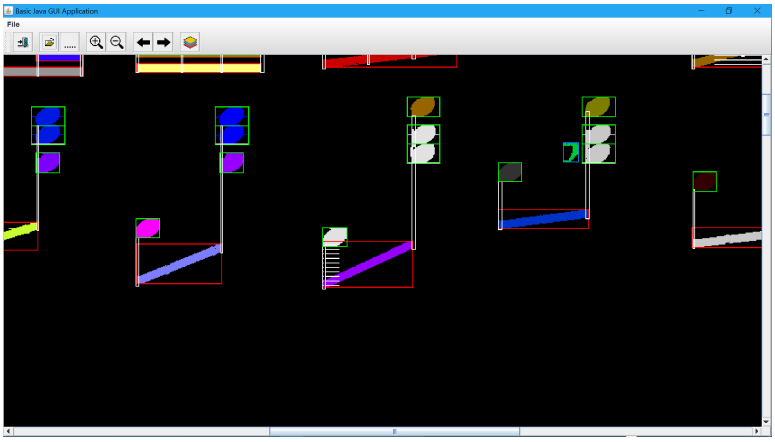

Figure 3.47: Note head within a beam's bounding box that causes the beam to be misrecognized as double in that place

Another interesting thing to note is that ties can be small and between notes or wide and above/below notes. This fact has to be considered during the recognition.

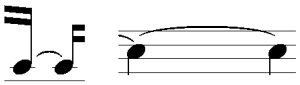

Figure 3.48: Different tie shapes

There are two types of chords - ones that appear in one column and ones that are spread across. The system described here only processes the former type. A component is recognized as a chord if its height is a multiple of SSW (because SSW is the height of a single note head). Then, it is simply divided into several components and each note head is processed separately. Some chords are not just one component; therefore, whenever a recognized note has the same x position as the previous one, it is put into a chord with the previous one.

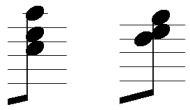

Figure 3.49: The left chord is in one column and it will be split into two components - the top note head and the bottom two note heads. The right chord is spread horizontally

It is important to note that sometimes the distance between a tail and its note head can be large - this happens in tailed chords. A sharp can also be further away from its note head than the standard distance.

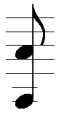

Figure 3.50: Tailed chord

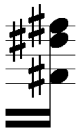

Figure 3.51: Sharp far away from its note head

The way flats and semibreve halves are distinguished is by the number of pixels in the area shown below. This is because flats have holes exactly there.

Figure 3.52: Semibreve half vs flat

Clefs and key signatures are recognized on every stave. Therefore, if there is a clef/key signature change at the end of a stave, it will be processed by

this system. When accidentals are recognized, it is checked whether their position indicates that they are part of the key signature.

Letters used for annotation or tempo marking might be recognized as notes. To prevent that, it is checked whether notes that appear outside the stave have ledger lines. This is done by looking at the horizontal projection of a component. There could be one or two ledger lines around a note. It does not work for all letters - for instance, the letter T also has a horizontal line. Therefore, when a newly recognized note is very close to the previous one and its duration is different, it is discarded as noise.

Figure 3.53: Letters that could be recognized as notes and notes outside the stave that have horizontal lines (ledger lines)

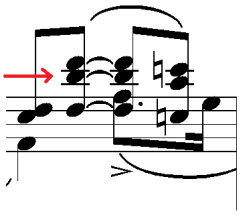

Figure 3.54: A note with two ledger lines

Initially, there was a grid check at every leaf of the decision tree to verify the recognition. It turned out that it is not needed (for instance, if a symbol has the dimensions of a filled note head and it has a filled middle, there is very little chance it is anything but a filled note head) and it makes the recognition stage slower.

Time signature consists of two numbers that are thick and, according to Fujinaga [8], difficult to recognize. Therefore, this system does not determine what time signature there is on the music score. It does label the first component on the stave that extends from the first to the fifth stave line as any time signature so that it is not recognized as anything else.

Chapter 4

Implementation

Java classes were organized so that there is a separate class for each stage of the recognition process. There are also classes that correspond to the MusicXML items (Measure, Note).

Pitch information taken from clefs, the key signature and accidentals is stored in a map where the key is one of the standard pitch names (C, D, E, F, G, A, B). When determining the pitch of a note that appears above the stave, the Y position of the first stave line was taken as a starting point and SSW/2 was subtracted from it in every loop iteration. That turned out to be a bad implementation decision as sometimes SSW is odd and the calculated position would become less and less accurate. In every other iteration of the loop, SSW is added/subtracted instead of 2*(SSW/2).

Distortion might cause a vertical line to be recognized as two vertical lines. Another bad implementation decision caused some notes to be recognized as semiquavers instead of quavers because of the presence of two separate vertical lines connecting it to a beam. As a fix, when a stem between a note head and a beam is found, the loop is broken.

An implementation-based efficiency improvement that was made is caching components' images. During the recognition, it is useful to extract a subimage of a bounding box from the main image to analyze it separately. That method is relatively slow; therefore, the first time the component's image is taken, it is stored as a cache. It reduced the overall time taken for the recognition stage.

Chapter 5

Project management

A logbook was kept throughout the project where all the ideas and thoughts were stored. It was important to save the solutions that proved not to work as the recognition stages were not fully implemented in order. Leaving a stage and coming back to it later requires remembering what has been invented for this stage already, what has been rejected and why.

Some literature review was done before every stage of the recognition. This was to see what has already been achieved in the area and what approaches have been taken. It made it easier to come up with new approaches and evaluate them in light of existing ones.

Every implemented feature was separately committed onto the remote repository after testing it. This helped to keep track of all the changes made and what remained to be implemented. It also helped in writing the report as it could be inspected when and how exactly every feature was implemented. Sometimes ideas that did not bear fruit were stashed.

The project did not quite follow the initial time line included in the project proposal due to the lack of understanding how long every stage would take. Nevertheless, time and effort was contributed to the project every single week since the beginning of the academic year.

Some stages, like line removal, turned out to be much more difficult and time-consuming than expected. Therefore, they were postponed till later and the focus shifted to the next stages. This was to make sure that a fully working system that performs the whole recognition process would be developed in time. Being able to process a very simple music piece was a bigger priority than, for example, making stave removal more robust and universal.

Chapter 6

Results and evaluation

6.1 Error classification

Errors in evaluating an Optical Music Recognition system can be categorized as follows:

- Omitted (not recognized) symbol
- Misrecognized symbol
- Recognized symbol that should not have been processed (e.g. a letter)

In terms of the music recorded in the output music score, the following classification can be used:

- Missing note
- Added note
- Note with wrong duration
- Missing bar line
- Mispitched note

The first four errors are more serious than the last one as they make the music sound unclear and much less like its original. This especially applies to multi-stave music scores: those errors cause the output audio to be 'out of

rhythm' - notes that should be played simultaneously are played at different times. However, added/removed notes in a chord are almost unnoticeable.

A missing bar line is problematic due to the MusicXML syntax - first, all the notes in the first stave are listed, then a backup value is added that specifies the duration of the whole bar and then the notes in the next stave are listed. If the bar is actually twice as long due to a missing bar line, the whole melody line on a subsequent stave will be shifted. This can be avoided by verifying the duration of the recognized bar, however it was not implemented in this system.

The last error type (mispitched note) makes the music less pleasant to listen to but not as confusing as shifted melodic lines.

6.2 Performance on certain music scores

Obviously, the system performs worse on scanned scores as there is more distortion and it is more difficult to correctly remove lines. Nevertheless, the main focus of the project was to make the system work on electronically stored music scores. More research would have to be done into, for example, binarization for the system to work better on scanned scores.

Music notation software called *Finale* was used to generate sheet music from the XML files outputted by the system.

6.2.1 Swan Lake

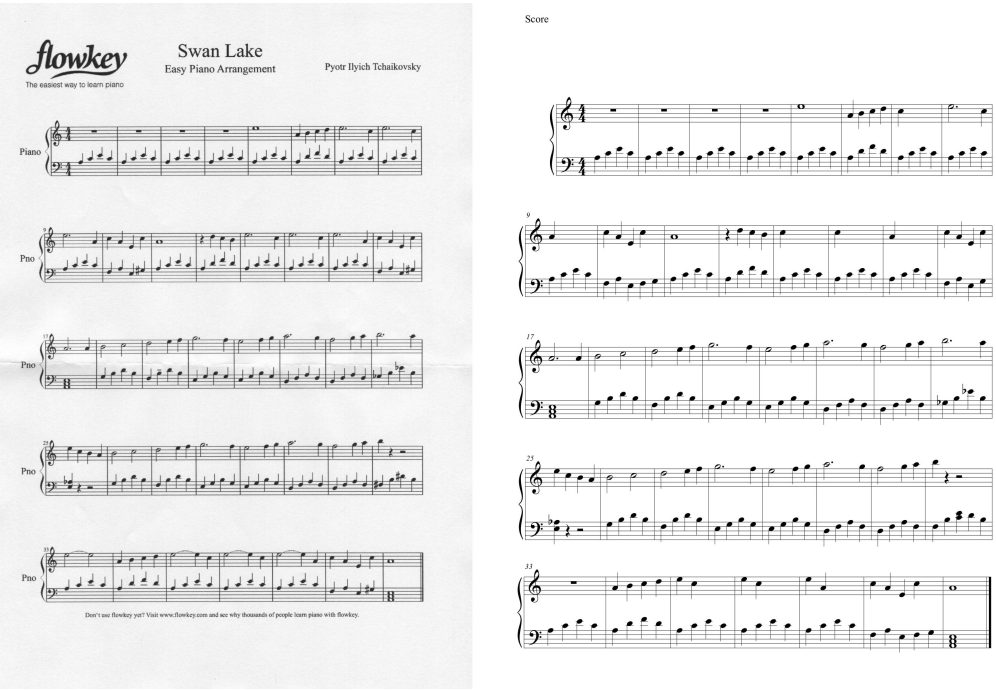

Figure 6.1: Original (scanned) vs output

In this score, some minims (e.g. bar 7) were not recognized due to the fact that they are too thick. A better binarization algorithm could potentially fix that.

One of the vertical lines in the sharp in bar 10 was not properly removed and it was recognized as a note instead. That indicates that vertical line removal could still be improved.

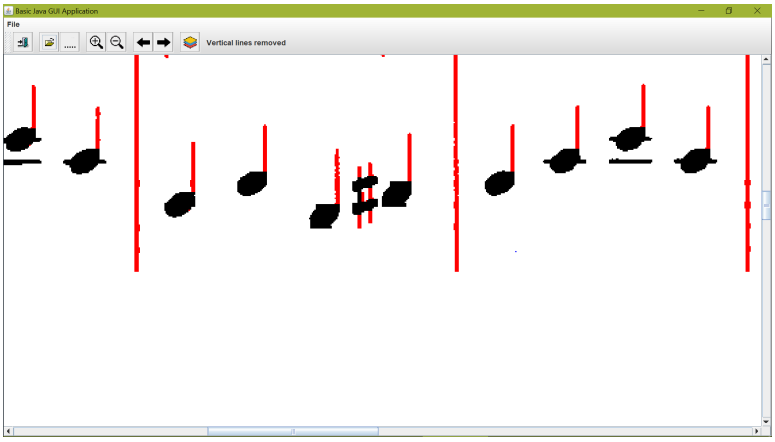

Figure 6.2: Wrongly processed sharp

Overall, there are a few added and missing notes, but most bars were recognized correctly and while playing the output score, it can be heard that it is still quite close to the original. It would definitely serve the purposes described in the specification section.

6.2.2 Auld Lang Syne

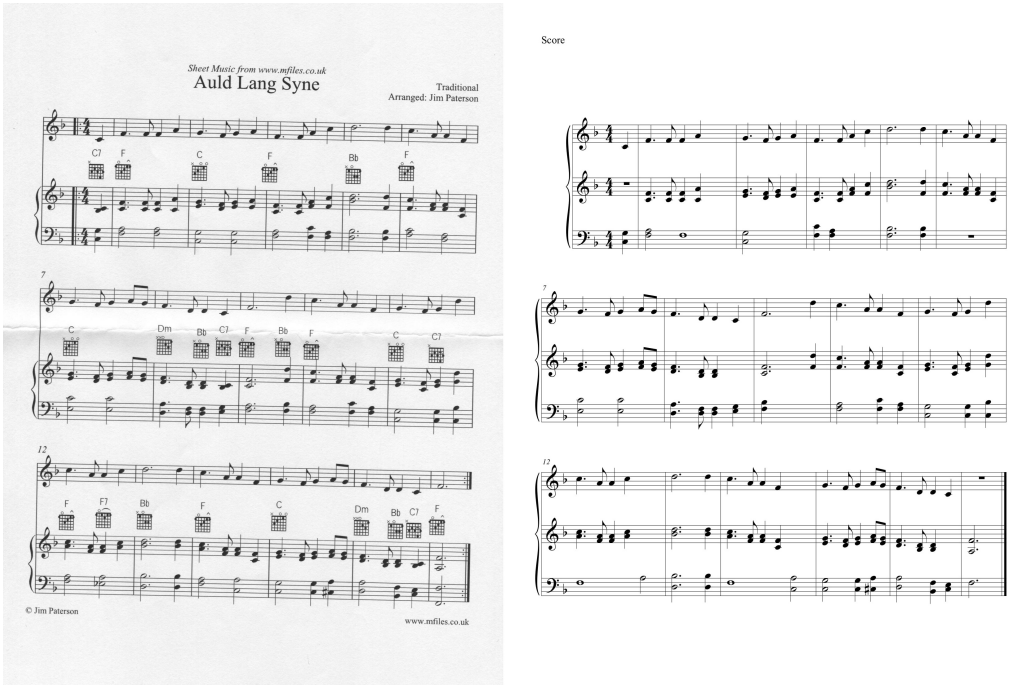

Figure 6.3: Original (scanned) vs output

Errors in this score (e.g. first system, bottom stave) are badly processed minims due to distortion (the input was a scanned image) and imperfect line removal.

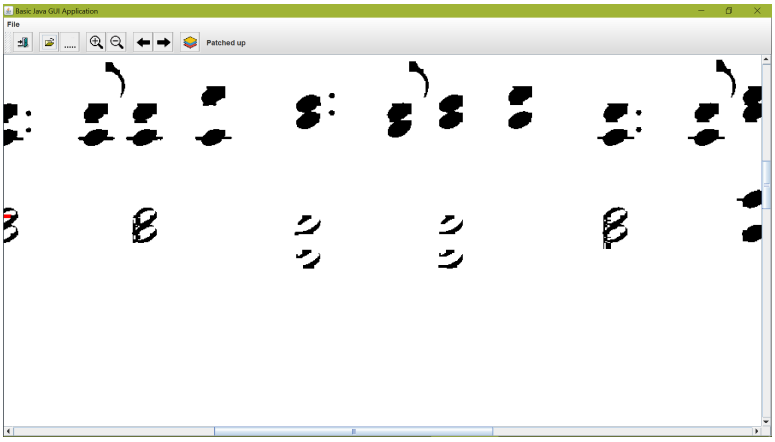

Figure 6.4: Minims split into two components

There are a few extra and missing notes but on the whole the output is quite close to the original. It can be noticed that 3 staves per system were correctly detected despite the guitar markings in between. The guitar markings used to be recognized as notes but this is not the case in the final release of the system.

6.2.3 Canon in D

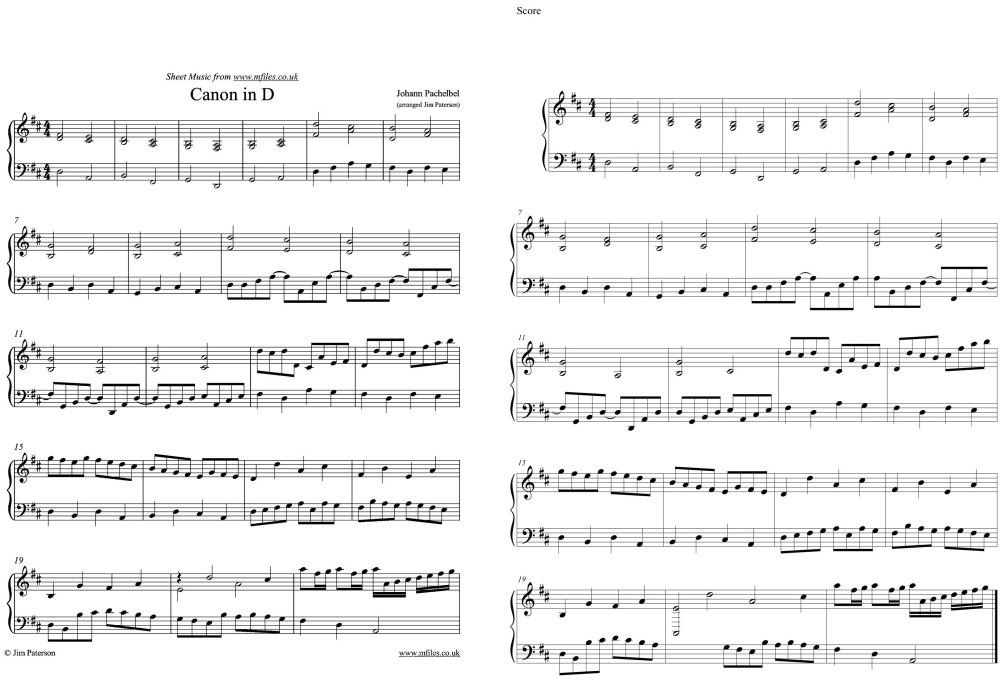

Figure 6.5: Original (electronic) vs output

There are again some damaged minims (bar 11 and 12) and a crotchet rest in bar 20 was misrecognized as a minim.

Overall, there are only 2 missing notes, one missing rest and one added note. The shift in the bottom stave in bar 20 comes from the fact that the system does not process multiple melodic lines in the same stave. This result demonstrates correct single and double beam recognition as well as tie recognition.

6.2.4 Relay station

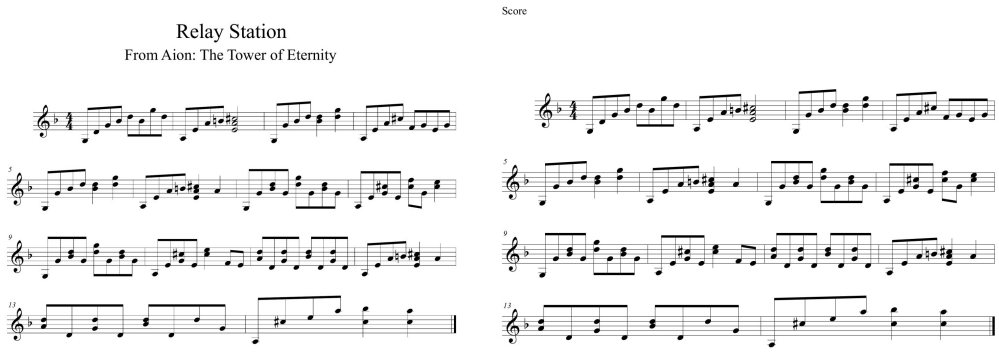

Figure 6.6: Original (electronic) vs output

This score is relatively simple, there are no errors whatsoever. It was used mainly to test accidental recognition - there are some naturals that are not that common on music scores.

6.2.5 Entertainer

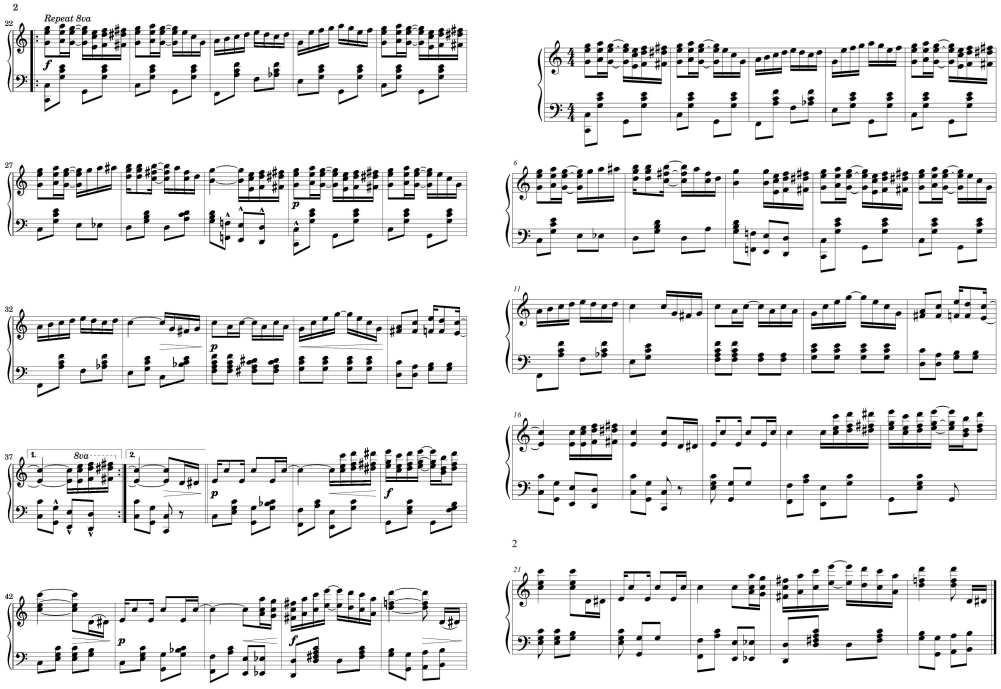

Figure 6.7: Original (electronic) vs output

This is the most complex music score that the system produced satisfactory output for. There are some dynamic markings that are not processed by the system (*forte* in bar 1, accents in bar 29, *piano* in bar 30). The repetition (bars 37-38) is not processed either.

Some notes are missing (bar 28 and 33, bottom stave) as the system does not process chords that are not in one column. Other notes (bar 33 and 39, bottom stave) are missing because some vertical lines were not removed. However, most of the missing notes do not stand out much when playing the resulting score as they are only part of a chord and there is another stave that was recognized correctly.

Generally, the performance on this score is very good and playing the output is not only interesting and enjoyable in itself, but also it could be used

as a learning aid and it would facilitate composing a different arrangement of this music piece. The recognition of complex beams and chords with accidentals is demonstrated by this result.

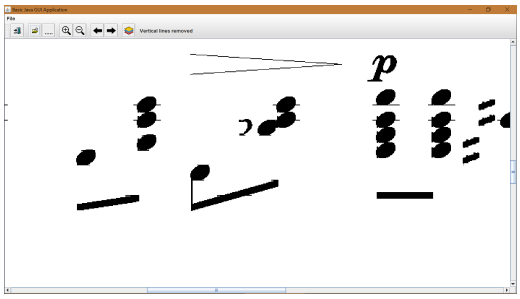

Figure 6.8: Unremoved vertical line and a chord spread across

6.3 Design choices evaluation

Stave removal overall works quite well, however more effort would have to be put into preserving minims.

Vertical line removal and patching took a large amount of time and, considering that there are still cases where they do not work correctly, they could probably be replaced by another, simpler, method.

Connected Component Analysis and decision trees perform without fault and, therefore, they are considered good design choices.

Right decisions to make were also recognizing semibreve halves separately, as that is easier than trying to stitch them back together, and removing vertical lines from accidentals, as that is also easier than trying to leave their vertical lines in place.

6.4 Efficiency

It takes approximately 12 seconds to fully process an electronic music score. Deskewing might take up to a minute, depending on the direction in which the image is skewed.

Chapter 7

Discussion

To sum up, recognition of the following has been implemented: treble clef, bass clef, sharp, flat, natural, key signature, semibreve (rest), minim (rest), crotchet (rest), quaver (rest), semiquaver (rest), beam, double beam, dot, tie, one-column chord, multiple staves. Recognition of the following has not been implemented: dynamics, time signature, glissando, two-column chords, multiple melody lines, in-score clef/time signature/key signature change.

Stave removal could still be improved. If there are a lot of beams in the same Y position, they are incorrectly removed as stave lines.

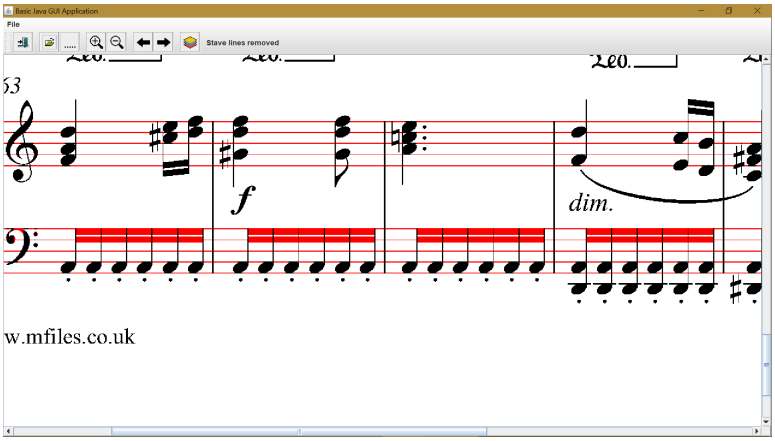

Figure 7.1: Beams wrongly recognized as stave lines

Glissandos are not processed by the system. If they appear on the given music score, they are incorrectly connected to notes by the patching algo-

rithm.

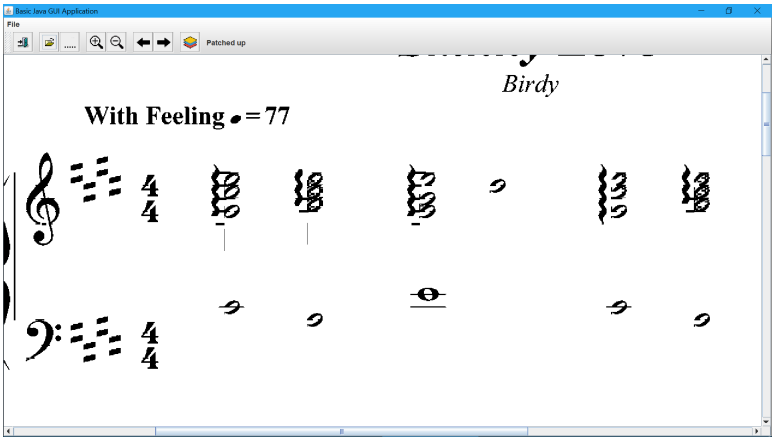

Figure 7.2: Glissandos connected to notes

As illustrated in the evaluation section, minim recognition is far from perfect. Some minims still remain damaged by line removal and are not recognized correctly. Hough transform could potentially be worth researching into for line removal.

As mentioned earlier, a good binarization algorithm would have to be implemented to process scanned scores. Some binarization algorithms that could be used were described by: Niblack and Sauvola [19], Otsu [20], Gatos [10] and Garg [9]. Additionally, a better rotation algorithm might come in useful. It might be worth experimenting with shearing [17].

The areas listed above would have been looked into had there been more time allocated to the project. If the project were to be done again, it might have been decided not to remove vertical lines and deal with them after labelling components (as suggested by Fujinaga [8]). Moreover, since line removal proved to be so challenging and time consuming, the focus would have moved to developing a fully working prototype much quicker.

Chapter 8

Conclusion

The system described in this paper combines existing, as well as newly invented, approaches to partially solve the problem of Optical Music Recognition. It is capable of converting some music scores into MusicXML with quite good accuracy, however there is a lot that remains to be explored and implemented. It is important to note that one of the goals of the project was to analyze, experiment with and evaluate various approaches to Optical Music Recognition. This report could certainly be used as a starting point for anyone without much experience in Music Score Recognition.

Bibliography

[1] David Bainbridge and Tim Bell. The challenge of optical music recognition. *Computers and the Humanities*, 35(2):95–121, 2001.

[2] Donald Byrd, William Guerin, Megan Schindele, and Ian Knopke. Omr evaluation and prospects for improved omr via multiple recognizers. *Submitted for publication*, 2010.

[3] G Sayeed Choudhury, Michael Droetboom, Tim DiLauro, Ichiro Fujinaga, and Brian Harrington. Optical music recognition system within a large-scale digitization project. In *ISMIR*, 2000.

[4] Bertrand Coüasnon, Pascal Brisset, Igor Stephan, and Couasnon Pascal Brisset. Using logic programming languages for optical music recognition. In *In Proceedings of the Third International Conference on The Practical Application of Prolog*. Citeseer, 1995.

[5] Christoph Dalitz, Michael Droettboom, Bastian Pranzas, and Ichiro Fujinaga. A comparative study of staff removal algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(5):753–766, 2008.

[6] David Doermann and Karl Tombre, editors. *Handbook of Document Image Processing and Recognition*. Springer, 2014.

[7] Ichiro Fujinaga and Bruce Pennycook. *Adaptive optical music recognition*. PhD thesis, McGill University, 1996.

[8] Ichlro FuJinaga. Optical music recognition using projections. 1988.

[9] Naresh Garg. Binarization techniques used for grey scale images. *International Journal of Computer Applications*, 71(1), 2013.

[10] Basilios Gatos, Ioannis Pratikakis, and Stavros J Perantonis. Adaptive degraded document image binarization. *Pattern recognition*, 39(3):317–327, 2006.

[11] Lifeng He, Yuyan Chao, and Kenji Suzuki. A run-based two-scan labeling algorithm. *IEEE transactions on image processing*, 17(5):749–756, 2008.

[12] Lifeng He, Yuyan Chao, Kenji Suzuki, and Kesheng Wu. Fast connected-component labeling. *Pattern Recognition*, 42(9):1977–1987, 2009.

[13] Wladyslaw Homenda. Automatic recognition of printed music and its conversion into playable music data. *Control and Cybernetics*, 25:353–368, 1996.

[14] Wladyslaw Homenda. Optical music recognition: the case study of pattern recognition. *Computer Recognition Systems*, pages 835–842, 2005.

[15] Rong Jin and Christopher Raphael. Interpreting rhythm in optical music recognition. In *ISMIR*, pages 151–156. Citeseer, 2012.

[16] Hirokazu Kato and Seiji Inokuchi. A recognition system for printed piano music using musical knowledge and constraints. In *Structured Document Image Analysis*, pages 435–455. Springer, 1992.

[17] Adolf W Lohmann. Image rotation, wigner rotation, and the fractional fourier transform. *JOSA A*, 10(10):2181–2186, 1993.

[18] Tam Nguyen and Gueesang Lee. A lightweight and effective music score recognition on mobile phones. *JIPS*, 11(3):438–449, 2015.

[19] Wayne Niblack. *An introduction to digital image processing*. Strandberg Publishing Company, 1985.

[20] Jin Soo Noh and Kang Hyeon Rhee. Palmprint identification algorithm using hu invariant moments and otsu binarization. In *Computer and Information Science, 2005. Fourth Annual ACIS International Conference on*, pages 94–99. IEEE, 2005.

[21] Christopher Raphael and Jingya Wang. New approaches to optical music recognition. In *ISMIR*, pages 305–310, 2011.

[22] Ana Rebelo, G Capela, and Jaime S Cardoso. Optical recognition of music symbols. *International journal on document analysis and recognition*, 13(1):19–31, 2010.

[23] Ana Rebelo, Ichiro Fujinaga, Filipe Paszkiewicz, Andre R. S. Marcal, Carlos Guedes, and Jaime S. Cardoso. Optical music recognition: state-of-the-art and open issues. *International Journal of Multimedia Information Retrieval*, 1(3):173–190, Oct 2012.

[24] Florence Rossant and Isabelle Bloch. Robust and adaptive omr system including fuzzy modeling, fusion of musical rules, and possible error detection. *EURASIP Journal on Advances in Signal Processing*, 2007(1):081541, 2006.

[25] Bolan Su, Shijian Lu, Umapada Pal, and Chew Lim Tan. An effective staff detection and removal technique for musical documents. In *Document Analysis Systems (DAS), 2012 10th IAPR International Workshop on*, pages 160–164. IEEE, 2012.

[26] Mariusz Szwoch. A robust detector for distorted music staves. In *International Conference on Computer Analysis of Images and Patterns*, pages 701–708. Springer, 2005.

[27] Lorenzo J Tardón, Simone Sammartino, Isabel Barbancho, Verónica Gómez, and Antonio Oliver. Optical music recognition for scores written in white mensural notation. *EURASIP Journal on Image and Video Processing*, 2009(1):1–23, 2009. Article ID: 843401.

Appendix A

Attached ZIP file structure

The directory named *src* contains all the source code. To run the program, *Music Score Recognition.jar* should be opened. To compile and run the source code, a Java IDE should be opened, a new project should be created in the same directory as all the unzipped files and the main method in *src/project/Main.java* should be run.

README.md contains the address of the repository.

The directory *symbols* contains images of symbols used by the program and the file *xmlHeader.xml* is appended to the top of the output XML file. Those files have to be in the same directory as *Music Score Recognition.jar* for the program to work.

Sample input can be found in the directory called *sheetmusic*.

The output file will be called *output.xml*.