Universidad Autónoma de Madrid
Escuela politécnica superior

Degree in Computer Science Engineering

DEGREE FINAL PROJECT

MUSIC SCORE CONVERSION FROM PAPER TO DIGITAL FORMAT

Mónica Villanueva Aylagas
Tutor: Alberto Suárez González
June 11^{th} 2015

ABSTRACT

Abstract

The goal of the present project is the design and development of an Optical Music Recognition (OMR) system as a desktop application. This software is able to recognize musical symbols from a scanned music score and convert them into an editable digital format.

The scope of the project covers the melodic line of the music including the detection and recognition of staves, notes, rests and modifiers such as accidentals and dots.

The algorithms used for this task are mostly unique designs based on techniques researched from different paper and available OMR systems. These algorithms have been tested with potential users who have been taken into account due to the importance given to the usability of the software.

Resumen

El objetivo del presente proyecto es el diseño y desarrollo de un sistema de reconocimiento óptico de caracteres musicales (OMR) en forma de aplicación de escritorio. Este software es capaz de reconocer símbolos musicales desde una partitura escaneada y convertirlos a un formato digital editable.

El alcance del proyecto cubre la línea melódica incluyendo la detección y reconocimiento de pentagramas, notas, silencios y modificadores como alteraciones y puntillos.

Los algoritmos usados para realizar esta tarea son mayoritariamente diseños únicos basados en técnicas investigadas en diferentes artículos y sistemas OMR disponibles. Estos algoritmos han sido probados con potenciales usuarios, que han sido tomados en cuenta debido a la importancia dada a la usabilidad del software.

KEY WORDS

Key words

Computer vision; Heuristics; Image processing; Music; MusicXML; OMR; Python; Score;

Palabras clave

Heurísticas; Música; MusicXML; OMR; Partitura; Procesamiento de imágenes; Python; Visión por ordenador.

GLOSSARY

- **AOMR**: Adaptive optical music recognition.
- **Bar line**: Vertical line that separate measures.
- **Ledger**: Additional lines added to a note when it is placed above or below the stave because of it high or low pitch.
- **OMR**: Optical music recognition.
- **Slur**: Curve line that join notes.
- **Stave**: Five long and equidistant horizontal lines in which the musical symbols are written.
- **Voice**: Musical line played by a single instrument. There are some polyphonic instrument that have assigned two voices, such as the piano or harp.

ACKNOWLEDGMENTS

First of all, I would like to dedicate this work to my parents, for providing me with the opportunity of study this degree and supporting me unconditionally in every problem and decision faced.

I would like to thank Alberto Suárez for proposing this wonderful project and give me the opportunity to develop it, for your guidance and your interest. Also, thanks to Daniel Garnacho for advise me to use the library OpenCV and Gabriela Sandoval for the last revision of this document.

I must not forget to mention the people who allow me to organize the different workshops: Coral Quevedo, Eduardo Lorenzo, Igor Nedeljkovic, Miguel Román and Adolfo Nuñez and definitely the invaluable help of the Vice Rector of Technologies, Sylvia Gil, for putting me in touch with Miguel Román to coordinate the UAM workshop.

Finally, thanks to all my friends for being there and specially to the musicians for giving me ideas and participate in the Arturo Soria workshop.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1. Motivation

The selection of this project among the long list of proposed works reflects the love and interest of the author for her two passions: computer science and music.

Traditionally, music scores have been written and copied by specialists, copyist, taking amounts of time and effort to individually produce each copy. However, taking advantage of the revolutionary invention of the printing press, Petrucci publish the first book of polyphony. This invention and its later improvements helped spreading music scores among more and more people.

Nowadays, most of the scores are in paper and its only digitalization is in image format. This allows only to read music, but not to take advantage of the full potential of a music score. The music field is in need of a way to convert sheet music to a more flexible format, like text have with optical character recognition (OCR) systems.

In particular, musicians could have access to scores in an editable format from which multiple functionality can be extracted. These types of software applications are optical music recognition (OMR) systems. They are far less developed than their counterparts for text.

The potential use for this type of application is wider than it could be imagined at first:

- Edition: A editable format interpreted by a music editor allows the correction of error in scores, the addition of voices and the inclusion of arrangements for learners, for example.
- Printing: The digital format allows the user to zoom in on a region of a complete score without loosing quality. This would help musicians with vision problems.
- Conversion to other formats: A fast conversion to other format such as MIDI can be performed from the digitized version of a music sheet. MIDI files are commonly used for choirs with little knowledge of music so that the singers can hear each melodic line separately.
- Study of music: Libraries such as *music21* can be used to extract information from scores and gather statistics from authors and their work.

- Storage and management: The storage of digitized scores is more comfortable regarding space and resources and it can be safer through replication. Also, the databases can use as search criteria the information acquired from the study in addition to the traditional name and composer. This way the search become more dynamic.

These are some of the possible uses for the program. Inevitably, users of the software will find further interesting applications.

1.2. Aims

The goal of this project is the design and development of a optical music recognition (OMR) system that identifies the main elements of a music score fed as a image to the program, and converts them into MusicXML, a format compatible with most music editors. The score images can be obtained by scanning a physical score or downloading scores from the Internet.

The symbols to be recognized by the system [see appendix A for description] is the minimal set that allow to reconstruct the melodic line of the score: the stave, time and key signatures, notes, rests and modifiers such as accidentals and dots.

1.3. Structure

The present report has a structure designed to guide the reader through the process of creation of the implemented OMR system.

The document has six chapters that describe the creation process and the system itself. Technical details and supplementary information are included in the appendixes. There is also included a bibliography section which contains the resources used to complete this work.

The current introduction expose the reasons to choose this project and its potential application as well as the scope of the work and the structure of the report. The following chapter, 2, describes the research and software available for OMR applications.

The chapters 3 and 4 explain the logic of the OMR system developed. The first one presents an overview of the functionalities and algorithms implemented. The second gives a technical view of the implementation of the application. The later chapter also details the software engineering steps followed in the development of the application.

The document ends with the evidence of the different test performed and the results of them and some final thoughts about the completed work and the future ideas for extend and improve the software.

Before the appendixes, the reader can find the bibliographic reference used in the report.

Given the complexity of the field, a basic guide in music theory has been included in appendix A for reference. For readers without experience in the field, reading this appendix is strongly recommended.

Some technical details and images are presented in the remain annexes. Appendix B includes images of original scores and outputs of the different programs used. Appendix C contains the graphics with the forms responses.

Finally, there is a user manual for the developed application and some links for downloading useful extras such as the API.

CHAPTER 2

STATE OF THE ART

2.1. Introduction

One of the first issues addressed in early stages of the project was the need for a novel application for conversion of sheet music to digital format. In this line, the author began an investigation covering any research in this matter, in form of articles and technical reports, as well as any commercial product that covered this functionality. What follows is a detailed reference of the resources found in both fields.

2.2. Research on OMR

In contrast to standard Optical Character Recognition (OCR), Optical Music Recognition (OMR) is a two dimensional problem in which the vertical dimension represent the note pitch and the horizontal one, the duration.

However, through the length of the years, the problem has developed from the problem of recognizing musical characters to try to learn from mistakes or adapt itself to the hand-script of a non printed score.

2.2.1. OMR

There are multiple algorithms that can be used to detect the different elements of a score. Following, there is a brief summary of the techniques described in Arshad et al. (2006) and other works. These represents the traditional work in the area.

Preprocessing

The starting point of most of the OMR systems is the binarization of the image. This process is computationally expensive. However, it reduces the complexity of the recognition algorithms by decreasing the amount of information to be processed.

The main two methods for binarization are fixed thresholding and adaptive thresholding.

- Fixed thresholding: Uses a precalculated threshold to discriminate between black and white pixels. It can be used in stable illuminated and controlled environments only. Its advantage is the reduced computational time needed to calculate the threshold.
- Adaptive thresholding: When the image has different lighting conditions, the threshold is calculated locally for small areas. There are a wide range of methods whose success rates depend on the original image. Some of these techniques [OpenCV (2014)] are Adaptive Mean Thresholding, Adaptive Gaussian Thresholding, Otsu's Thresholding [Vigliensoni et al. (2013)] or Niblack's Thresholding.

Rotation correction is another common preprocessing method, since most of the systems require that the image not be skewed. There are multiple techniques to determine the rotation degree that is used to correct the slant of the score. In this work it is assumed that this correction is not needed. In his thesis, Carretero de la Rocha (2009) describes three examples of transforms that can be used to solve this problem: Hough Transform, Fourier Transform and Radon Transform.

Stave line identification

The above mentioned transforms can be used also to detect the stave lines. Other algorithms that can be used for this purpose are Horizontal Projection [Sheridan and George (2004)], Correlation, Frequency Count [Fujinaga (1997)] and Template Matching.

- Horizontal Projection: This is the most common technique. It counts the number of black pixels in a row and creates a histogram with the data. The peaks observed correspond to the lines of the stave.
- Correlation: In this method, an ideal stave is compared with the original image. The space between lines is progressively increased until an optimal value is reached. It does not find the stave itself, but the gap between lines. Its drawback is that it is computationally complex algorithm.
- Frequency Counting: As in Correlation, it is used to find the value of the space between stave lines. It counts the number of consecutive white pixels. The highest frequency is selected as the sought value. It generally has a similar accuracy as Correlation. Moreover, it is more efficient.
- Template Matching: it compares the original image with a pattern of reduced area. The pattern is moved over the original one pixel at a time calculating the differences of the window.

There are also other more complex methods based on graph theory.

There are many different strategies to deal with the stave once it has been recognized. Most authors advocate for its removal to reduce the complexity of the subsequent processing. Others claim that it introduces modifications in the symbols that remain, so they just ignore the stave lines [Sheridan and George (2004)].

Figure 2.1: Different techniques to deal with the stave [Sheridan and George (2004)]

Musical object location

The algorithms described here are base on the premise that the stave lines are remove. They locate objects that are recognized by the methods described in the following section.

- Bounding Box Analysis: In this method a rectangle is drawn to encapsulate each remaining object after the stave has been removed. The boxes can be combined into one using heuristics that compare their positions.

- Flood Fill: The goal of this method is to determine the region connected to a given pixel. It can be used to locate symbols more or less complex. It must be used once the stave is removed, otherwise the objects will not be isolated, but linked by the stave lines. These objects can be primitive elements such as an accidental or high level symbols such as a group of eighth notes.

Symbol identification

This is the most creative and diverse area of the OMR problem. The number of algorithms is almost as varied as the symbols to recognize. Different authors have proposed diverse approaches for the same problem. Here is a brief description of some of them:

- Slicing: In this method the number of transitions between black and white pixels in a chosen section (row or column) is counted to identify an object. It is useful to recognize accidentals.

- Connectivity Analysis: This technique is used to find beams and slurs by locating paths that link two points.

- Vertical Projection: This algorithm is similar to Horizontal Projection. Running the two algorithms together creates a signature for a symbol that can be matched with signatures of symbols that have already been recognized.

- k-Nearest Neighbor (kNN): This method calculates the type of an object using a database of patterns and a defined metric for distances. Each object is characterized by the values of a set of features. Similarity between objects are quantified by a distance function.

- Neural Networks: It is possible to train a Neural Net to classify the objects. The recognition is fast, but there is a cost associated to training the net.

- Signature Analysis: This technique is based on calculating the relative distance between the symbol and one of the edges of its bounding box. It is used to recognize accidentals, rests, and clefs.

- Character Profiles: This algorithm uses the relative width and/or height of the symbol as a mean of recognition. It is used to identify accidentals, key signatures, beams, dots and rests.

Some of the algorithms described in the previous section such as Template Matching, Hough Transform or Bounding Box Analysis can also be used for recognizing symbols too.

Recognition of semantics of music notation

Graph Grammars, derived from the studies of Chomsky, can be used to correct recognition mistakes. The primitive elements that have recognized are arranged in a graph to create high level symbols; the use of context symbols and grammar rules can verify the validity of the recognition process.

2.2.2. AOMR

kNN and genetic algorithms

In this section it is presented a review of the contribution made by Ichiro Fujinaga to Adaptation Optical Music Recognition (AOMR) in his doctoral thesis in 1996, published the following year [Fujinaga (1997)]. In this work, Fujinaga explains what is an adaptive system and its advantages as well as a description of his implementation and used algorithms.

For Fujinaga, an adaptive system is that that can overcome modifications in the usage and learn and improve the success of recognition of both known and new objects. To do so, in the score are located objects and compared to those stored in a database. The processed data is used as training data that is used for recognition of the symbols in the score. The learning algorithms used to implement this system are k-Nearest Neighbor (kNN) classification and a genetic algorithm. Other techniques described earlier such as Horizontal and Vertical Projection are also used.

Given an unknown object, the kNN is used to compare it with the stored symbols by means of their feature vector. The genetic algorithm is used to accelerate the speed of finding a near-optimal set of weights.

Years later, Fujinaga and other researchers developed a "toolkit for the creation of domain-specific structured document recognition applications by domain experts with limited programming experience" [MacMillan et al. (2002)].

Ensembles

Another attempt to use learning methods in the task of adaptive OMR is the approach developed by Byrd and Schindele (2006). The method used here is the construction of a classification ensemble using PhotoScore 3.10, SmartScore 3.3 Pro, and SharpEye 2.63. as black-box classifiers. The argument is that using different classifiers, the error made by one of them can be compensated by the success of the others.

The authors studied the accuracy of each of the tools in different fields (pitch, duration, lose of information...) in order to develop rules for the triangulation of the multiple-recognizer OMR. However, the system has not been implemented yet. Relatively small improvement was expected from the use of these technique.

Logistic regression

The last research found in this area [Ben-Dayan and Giloh (2013)] is a system whose goal is to reproduce music at the same time that a composer is writing it, reading this score through a camera. Most of the recognition steps implemented for this solution are described in the section on OMR.

The most interesting ideas applied in the project are the capture of video frames and the techniques used to detect duration and non-notes symbols. The orientation of the score is calculated using a kind of QR code in the corners of the page. The learning machine is actually a one vs all Logistic Regression classifier previously trained. This simple and efficient technique is preferred over a neural network because of the realtime nature of the video based system.

2.3. Software

After a extensive and exhaustive search on the Internet, three different types of software for OMR were found: desktop applications, mobile applications and services. The last two are outside the scope of this project, so they will not be described. Following is a summary of the different desktop software and some examples of their usage.

Comparisons between images can be found in a annex [B.2] to lighten the reading of critical information.

2.3.1. Proprietary software

In the following sections the reader can find the description and output of two proprietary applications for OMR. This kind of software does not allow the modification or redistribution of the code without the agreement of its author. In most cases the applications are distributed with pay per use licenses.

PhotoScore

This software for optical recognition of musical characters is available both for Windows and for Mac OS. The editor Sibelius includes a lite version of this program.

The program is developed by Neuratron and can be purchased for the price of 299€ in their website `http://www.neuratron.com/index.htm`. A free trial version with reduced features can also be downloaded from this site.

PhotoScore accepts TIFF, PDF, bitmap and PhotoScore files as an input. Different output formats are available: PhotoScore files, PhotoScore files for Sibelius, MIDI, MusicXML, NIFF and WAV.

Some testing has been done to check the reliability level of the program.

1. Simple score
   The example shows the digitization of a simple score, the minimum that the application should read, in the view of the author. The chosen score is a musical arrangement of an original music score of the 2004 film *The Chorus* called *Caresse sur l'Ocean*.

   The outcome, as it can be appreciated in the figure B.2, is almost perfect. It also recognizes many of the pencil annotations. Nevertheless, these are precisely the origin of the few flaws that can be seen.

2. Complex score
   The second example is a scanned version of *Salut d'Amour* by Edward Elgar. This score is more complex not only because of its extension but for the types of notations that includes: piano defects in a violin particella, ornaments, and strings notations, for instance.

The software detects multiple errors and warns if the printed score option is wrong and needed the handwritten one. Nevertheless, despite the flaws found, the program does digitize.

The recognition accuracy is deficient in general. Nevertheless, there are some passages in which the melodic line can be followed [see figure B.4].

The study of this program done by Byrd and Schindele (2006) conclude that PhotoScore is the most accurate software of the three researched for text annotations.

SharpEye

Music scanning program available for Windows 95 or later versions of the Windows operative system.

The software is developed by the company Visiv and can be purchased for 162.07€ in their web site `http://www.visiv.co.uk`. The complete version is available for a 30-day free trial.

SharpEye can process a scan of the score or read input files with the formats BMP and TIFF. The output can be saved as MIDI, NIFF, MusicXML or a SharpEye File.

1. Simple score
   The recognition is perfect. As PhotoScore, it recognizes the pencil symbols. Furthermore, it does not introduce foreign elements. This can be due to the lack of letters in the digitization with this software (title, composer...) [see figure B.6].

2. Complex score
   The errors made in this score are quite similar to the ones made by PhotoScore. The overall output is unsatisfactory [see figure B.8] and even worse than with PhotoScore.

In the paper written by Byrd and Schindele (2006), SharpEye is also studied. They conclude that this software is the most accurate among those considered in their assessment. Their claim is that SharpEye is the most accurate program in all tested features except for duration. It has problems recognizing beams, which eventually lead to misinterpretation of duration.

2.3.2. Open source

Audiveris and OpenOMR are two open source tools that recognize music characters. Open source applications allow the user to modify and/or redistribute the software completely free and without the explicit permission of the author, provided some conditions are met, which depend on the license.

Audiveris

Open source tool for optical music recognition, written in Java and published as free software under the terms of the GNU General Public License. It is available for free download from: `https://audiveris.kenai.com/`

Audiveris accepts bitmap, GIF, JPG, PNG, TIFF and PDF formats as input. Possible outputs formats are MusicXML and PDF. It includes plugins for Finale, Finale Notepad and MuseScore.

1. Proposed sample
   The score of the opera *Carmen* by Bizet, *L'amour est enfant de bohème*, was included in a collection of samples in the software distribution. However, the results are not completely satisfactory, as can be seen from figures B.9 and B.10.

   The errors appear mainly in the first two staves. The ones in the following staves are related with annotations, glissandi and 8^{va}, for instance.

2. Simple score
   The pitch digitization is acceptable but not entirely satisfactory [see figure B.12]. The time signature does not match and there are multiple overlapping symbols, as well as modifiers that do not appear in the original score.

3. Complex score
   For this example, numerous errors are present. Unlike PhotoScore, Audiveris outputs a failure message and does not continue processing.

OpenOMR

Open source optical music recognition (OMR) tool for printed music scores. It is also written in Java and published as free software under the terms of the GNU GPL.

It can be downloaded freely from the official website `http://sourceforge.net/projects/openomr/`

OpenOMR differs from the rest of the analyzed software in the output. It allows a user to scan a music score and play it through the computer speakers.

The result of the recognition is difficult to match with the original score because of the numerous error and the output format.

CHAPTER 3

DESIGN: PROBLEM APPROACH

3.1. Introduction

The purpose of this chapter is to explain in a clear and complete way the algorithm designed to recognize the symbols in a music score, independently of whether the reader is a musician or an engineer.

The structure followed defines the problem and specifies the scope of the project taking into account the limited time and resources available for the work. Following, there is a complete depiction of the different steps performed for recognizing music symbols.

3.2. Project definition

The goal of this project is to automatically recognize the symbols in a musical score and generate a digitized version compatible with most music editors.

The process usually starts with a printed score. In order to feed it as the program input, it is necessary to convert it into an image by scanning it. Another way of obtaining suitable scores as input is to download scores from the Internet. In both cases, the accepted formats are images such as .bmp, .pgm, .sr, .jpeg, .png or .tiff formats. An important clarification is that it does not admit PDF as an input format. The reason is that the library used to convert the file into an array of pixels does not allow to process images stored as PDF files.

After the processing is complete, the result or output obtained is an XML (Extensible Markup Language) that is the digital transcript of the input score. The XML used is MusicXML [MakeMusic (2011)], which is an standard compatible with most music editors currently available. The goal is that this output, once interpreted by a editor, should be equivalent to the original score.

Both input and output descriptions must be clarified at some point: not every image is suited as an input. The output can also exhibit some variability in some cases. Images should be straight, that is, they must not have any skew. Furthermore their quality should be sufficiently high so that they can be successfully recognized; tests were made using at least 300 dpi (dots per inch) or downloading scores from the MuseScore database. The similarity between the original input and the output is delimited by the scope of the project. In this case, the main goal is to retrieve the basic structure of the score; a monophonic melodic line.

An additional and important requirement is that the software produced should be easy to use by anyone, regardless of their skill. The application developed should run on different platforms and operating system.

The scope of the project includes recognizing the stave, notes, rests and modifiers, namely accidentals and dots. The user provides the parameters that structure the music, namely time and key signatures. The implemented solution is not designed to recognize complex rhythms, polyphonic music, annotations and another important features of sheet music. These are left as objectives for future development and improvements of the application.

3.3. Design

The score is loaded into the program as a grayscale image. Each pixel in the image takes a value between 0 (black) and 255 (white). The value of a pixel can differ depending on the quality of the original input. The technique used to discriminate black from white pixels needed for running the rest of the recognition is binarization; however, it is not used as a typical preprocessing. In order to reduce the computational effort, the score is not completely binarized. Instead, it compares the value of a certain pixel with a threshold when needed.

The recognition process consist of the following steps: first of all, the stave is recognized. Then the symbols are identified. These include notes, rests and modifiers, which can be accidental or dots. With all the information gathered, the writing can take place.

Once the recognition is completed, the output is printed in one batch into the XML file in order to keep coherence. If the information were written as it is acquired, the notes would not allow modifiers. This approach has the drawback that if something goes wrong during a recognition and the process cannot finalized, no partial output could be obtained.

To meet the portability and usability requirements, the software is packaged as a standalone application that can be distributed and run without the necessity of installing any library or changing to a different operative system.

3.3.1. Stave identification

The initial step in most optical music recognition (OMR) processes is the recognition of the stave [A.2.1].

The project has used a Horizontal Projection algorithm [2.2.1], like the one described in the state of the art, with some adaptations. Instead of adding all the black pixels along the horizontal direction and creating a histogram, it compares the normal length of the detected horizontal line with the size of the image and decides whether it is a stave line or not. To complete the stave recognition, this process is repeated until the end of the image is reached. The distance between the last line of a stave and the first one of the following is divided in two and equally distributed among the two areas. Each one of these areas is taken as the Bounding Box of the stave and it includes a stave and above and below, half of the additional space between staves.

Taking advantage of the complete processing of the staves in the image, some parameters are calculated for later use as relative distances. These parameters are the average thickness of the stave line (henceforth t), the average distance between stave lines (henceforth d) and for each area, the position of the upper line (henceforth u).

Parameter nomenclature	
d	Average distance between stave lines.
t	Average thickness of a stave line.
u	Relative position of the upper line of the stave on a given stave area.

Table 3.1: Parameter nomenclature

The stave lines, contrary to the common practice in literature, are not erased. Instead, they are used later for reference. In this manner the problem of destructing or deforming the remaining music symbols is avoided. The stave Bounding Boxes are also stored for further analysis of these glyphs.

3.3.2. Symbol identification

The following step in the processing of the score is the location and recognition of the symbols written on the stave. Different algorithms are used to identify the different types of symbols.

The location procedure finds the symbols in the stave area and creates Bounding Boxes around them to isolate them. These delimited areas are the regions later analyzed for its recognition. Their common feature is that their areas are not stored unlike the stave area, because it is assumed that their Bounding Box is sufficiently adjusted so that nothing inside them is part of another glyph.

The recognizing process is constituted by two stages: the spatial location of the symbol and the recognition of their features.

Notes

The procedure to recognize notes starts with the location of heads using patterns chosen by the users. For notes [A.3.1], the features to recognize includes pitch and duration. With the position of the head the pitch can be acquired while for identifying durations the strategy involves finding the stem and recognizing how may beams of flags has attached. The beams and flags are identifiable because they are divergences from the stem that are thicker than the stave lines.

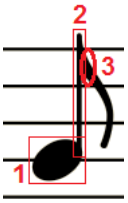


Figure 3.1: Recognition process for a note

The head notes have an oval shape of height equal to d and they are of three different types. Whole notes are white and do not have stems, half notes are also white but do have a stem. Finally, quarter notes and smaller ones have black heads and a stem. The number of divergences (beams or flags) of the stem identify the type of note with black head.

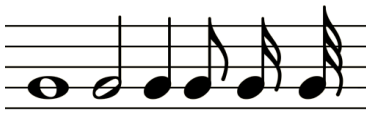


Figure 3.2: Types of heads and notes

The technique used for the location of notes is Template Matching [2.2.1]. First, it is necessary to identify a pattern to match each head. There are five possible note heads in a score: whole notes on a line and on a space, half notes on a line and on a space and black headed notes. However, the Template Matching algorithm used, accessible through the library OpenCV [Bradski (2000)], admits the use of a threshold of similarity that can be adjusted to reduce the number of patterns to three: whole, half and black heads.

The application forces the user to select patterns for the different types of notes heads. To do so, the user clicks on the center of the head, and the program creates a Bounding Box which contains it. This area is a square that has a side length of $3d$, that is a bit larger than the head of the note. In a later step the application tries to adjust this area finding the limits of the head [see number 1 in figure 3.1] .

The storage of the pattern selected by the user allows the use of the Template Matching algorithm to search all the occurrences of the pattern, within a given similarity, along the complete score. Even though the area is not kept, the coordinates of the head are stored for additional study.

Other methods such as Vertical Projection [2.2.1] could be used to perform this task. They were finally dismissed because the former algorithm was already implemented by a library, and the previous one could lead to a more complex recognition since the Vertical Projection does not only locate notes, but it can also identify other symbols, such as accidentals. Furthermore, the pattern selection allows the user to run the application for a completely different end: algorithmic composition. This is a new field in music that uses rules for composition to create structures of great complexity, for example. In this case, the user can play with the patterns and different parameters to modify a traditional score.

Concerning the recognition of the characteristics, for notes it is necessary to discover both pitch and duration. Using the Template Matching algorithm with different patterns for whole and half notes removes the necessity to run an algorithm to identify its duration. For black headed notes, however, it is required to distinguish between quarter notes, and smaller ones.

To do so the software finds how many divergences there are along a stem, ignoring stave lines, whose t has been previously estimated. The stem [see number 2 in figure 3.1] is found using a variation of the Vertical Projection algorithm in both sides of the head. The divergences [see number 3 in figure 3.1] are discovered searching around the stem identifying patches of consecutive black pixels. If the thickness of these patches of black pixels is greater than the stave line, a flag or beam is located. The duration it is derived from the maximum number or beams associated to the stem of the note [see figure 3.3].

Every note, regardless their duration, has also a pitch that has to be recognized. Using the calculated values of u, d and t, and the coordinate for the Y axis of the head center, it is possible to deduce the position of the note respect to the stave and from here its pitch value.

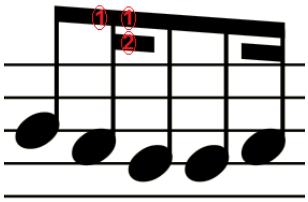


Figure 3.3: Note duration

Once these features have been recognized, the information is stored. The Bounding Box around the note (not only its head) is deleted from the stave area to avoid interferences and noise in the recognition of other type of symbols. The goal is limited to recognizing melodic lines, assuming that no more notes are in the same column, that is to say the application is not capable of recognize chords. This is the reason why the area is adjusted horizontally to its width, but not vertically.

Modifiers: Accidentals

The recognition of accidental [A.3.3] types is reduced to the most commonly used ones leaving the less used symbols such as double accidentals or microtonal notation for future work.

The area around the head center of the detected notes is a region in which accidentals are likely to be found. Their grammar location is preceding a head note. If in a gap of size d before the head noise is detected, an accidental is located and the recognition process begin.

There are three types of accidentals recognized: sharps, flats and naturals. Each one of them is differenced by an algorithm based on its Character Profile [2.2.1]. Each type of accidental has a unique shape that can be used to identify them; in this case, the recognition of the type of the accidental is based on the relative distance from the right side of the symbol to its Bounding Box. Sharps have black lines above and below the center of the symbol. Naturals have a single line going down and flats have none.

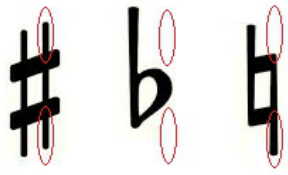


Figure 3.4: Distinctive areas for each accidental

The information obtained from this recognition is included in the note data. The accidental area is the cropped and discarded from the stave area to avoid future mistakes.

The Bounding Box around the accidental is adjusted using the same principles as in the recognition of note symbols. Accidentals can be located using Vertical Projection, but it seems slower computationally to use this technique since the position of the note heads are already known and grammatically the accidental position is right next to it.

Rests

Rests [A.3.2] receive the same treatment as notes: the area is discarded, but the coordinates of their center are stored for additional study.

Next in order, rest of different duration are recognized. Determining the location of rests is more complicated than that of notes, because the lack of templates to match and the variety of shapes. The algorithm designed for this task is a mixture of a Signature Analysis [2.2.1] (based on the relative width of the symbol) and of Character Profile [2.2.1] (based on the relative position), and different strategies are used to recognize long rests and short rests.

Long rests include whole and half rest because of their similarity. Their shape is that of a rectangular black box. The difference between them is the position in the stave where they are found.

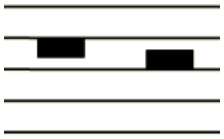


Figure 3.5: Whole and half rests

The general method to find these kind of rests is looking for their relative width, which is greater than d, but not much larger that twice that size, depending on the score printer. To identify whole rests, the consecutive black pixels must be found below the fourth stave line while the half rest is placed above the third line.

Short rests include quarter, eighth and smaller rests. Since there are no long rests, because they have been removed from the stave area, any accumulation of black pixels less than $2d$ and greater than $d/2$ is likely to be a rest. These values have been estimated empirically. To really discriminate the duration of the rest is critical to examine the second space: quarter rests are wider, eighth rests are thinner and from the sixteenth rests to smaller ones, the pattern detected in the third space is repeated in different spacings.

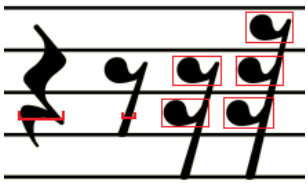


Figure 3.6: Distinctive features for small rests

Rests are described in the literature to be recognize through Character Profile [Carretero de la Rocha (2009)] or Signature Analysis [Chung and Lee (1995)]. The described algorithm has a good overall performance, and it is an approach designed by the author herself mixing this ideas.

Modifiers: Dots

The final step of the recognition process is the dots [A.3.3], which are used to increase 50% the duration of a note. As for accidentals, their position is known by music grammar. Dots are located at the right side of the symbol at the height of the head of a note or the center of the rest (usually the third space).

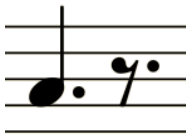


Figure 3.7: Position of a dot regarding the symbol

Once a possible dot is detected, the way of determining it is really a dot is to calculate its height or width (in this case it should not matter which) that would have to be a relative value between the half and a third of *d*. These values have proved to be the most successful in the tests performed.

The presence of a dot is registered as additional information associated to the symbol (note or rest) it modifies. Once more, the width of the Bounding Box of the glyph adjusted to the width of the symbol but covers the complete stave height. As in the other cases, the dot area is remove from the original stave area.

This algorithm is based on the Signature Analysis technique read on Arshad et al. (2006) reducing the area of search by means of the grammatical position of the dot regarding the symbol it modifies.

CHAPTER 4

SOFTWARE

4.1. Introduction

In the following sections, the reader will find a technical description of the steps reached for the design, development and deployment of the OMR software developed in this project.

The project meets the results of the requirements, the programming language and the limited time and workload esteemed by 12 ETCS.

4.2. Agile software development

The problem addressed is fairly complex. In fact none of the commercial application for OMR [2.3] or researchers [2.2] have accomplished satisfactory results. Given the complexity and the uncertainties associated with the project, an agile development methodology was chosen.

The agile development is characterized by short iterations with adaptive cycles in which any part of the development process can be affected, from requirements, to testing through design and codification. There is a special relevance in the incremental development, continuous integration and powerful and effective testing process to the detriment of extensive amounts of intermediate documentation, which does not make much sense in a ever-changing development.

The benefits of this option meet with several requirements, such as the extension of the project, with the change at any moment of the requirement elicitation, the "deliver of working software frequently" [Beck et al. (2001)], that is, having at any moment functional prototypes to show the client (in this case, both musicians and the tutor).

The resulting structure of this project was the development of two totally functional prototypes with a preliminary phase of analysis and a design and coding stage mingled with minor unit testing and a more extensive epoch before the final release of the prototype in which the integration tests were executed and ultimately, system testing checked by the author before the workshops and the final users at them [5.4].

4.3. Requirements elicitation

Taking advantage of the expertise of the author, the scope of the work was determined before consulting any of the potential users.

The main goal, and with it the metric to evaluate the system, was minimization of errors. Therefore, the extension of the work covered a reduce area of score recognition, maximizing the effectiveness. Also, in a early stage, a survey [C.2] was designed and distributed among musicians in order to gather information about what they believe would be useful in an OMR application. Their answers were taken into account in the final design.

4.3.1. Functional requirement

Functional requirements are referred to requirements that explain what the system must accomplish to meet the expectations of the final user.

In this case, the functional requirements have been growing with the consecutive prototype iterations. A complete list of the functional requirements, sorted by modules is below.

- **Interface**: Module in charge of the interaction of the user with the application.

FR.1. The software must provide a way to open the score:
  FR.1.1. Chose directory.
  FR.1.2. Chose file name.

FR.2. The software must provide a way to save the output:
  FR.2.1. Chose directory.
  FR.2.2. Chose file name.

FR.3. The software must allow to set global score parameters:
  FR.3.1. Time signature.
  FR.3.2. Key signature.

FR.4. The software must allow to choose patterns of notes:
  FR.4.1. Whole note head.
  FR.4.2. Half note head.
  FR.4.3. Black head (quarter note or shorter value).

FR.5. The software must have a help button.

- **Recognition**: Module that reads the score and recognizes its musical elements.

FR.6. The software must accept image formats. At least PNG and JPG.

FR.7. The software must recognize staves.

FR.8. The software must recognize notes:
  FR.8.1. Pitch.
  FR.8.2. Simple rhythms (e.g. tuplets are excluded).
  FR.8.3. Accidentals.
  FR.8.4. Dots.

FR.9. The software must recognize rests.
  FR.9.1. Simple rhythms.
  FR.9.2. Dots.

- **Writing**: Module that converts the recognized elements into MusicXML, which is compatible with most music score editors.

FR.10. The output must have a format that is compatible with mos music editors.

4.3.2. Non-functional requirements

Non-functional requirements establish restrictions on how the system will perform the functional requirements. The main goal was to be as user friendly as possible so as not draw back any potential user because of the complexity of the software. To do so, the priority requirements were:

NFR.1. Portability: The application can run on different platforms with different operating systems.

NFR.2. Usability: This application is easy to use and the way of operation is intuitive.

The aim of turning the work into a open source project in order to keep developing it in the future has given rise to further non-functional requirements:

NFR.3. Maintainability: Measures the efficiency with which a system can be fixed after a failure, or after a bug has been reported.

NFR.4. Extensibility: The functionality of the system can be increased with a reasonable effort.

NFR.5. Documentation: Explanatory documents will be used throughout all the development process, both for the requirements an the code.

4.4. Software design

In this section will be specify the design process for the software, including a high level view of its structure as well as its detailed design. Accompanying the software design, it will also appear the test plan and a time schedule for its development.

4.4.1. Programming language

Before starting to design, it was necessary to set a programming language, or at least, a paradigm to focus properly the design. Regarding this, it was as fine a moment as any other to choose a language.

Initially the thought was using Python because of its unfamiliarity as a school-learned language and its appeal in the labor market as a way of standing out. Following, there are exposed some features that make this language fit for this project.

General

On the one hand, its learning curve can be reduced by the Java knowledge acquired in subjects during the degree, and likewise Java, is portable so it meets with this requirement [**NFR.1.**]. Also, as a pro to its use, Python has a great number of IDE's that can help with it coding and management, as Eclipse, which is a common one at the school.

It has also been taken into account the programming speed, to obtain frequent functional prototypes, following the agile methodology chosen [4.2]. It is esteemed that "Python code is typically 3-5 times shorter than equivalent Java code, it is often 5-10 times shorter than equivalent C++ code" [Python (1997)]

In case of deciding to turn the project into a open source one, the legibility of the code thanks to its compulsory indentation makes it much easier to understand to outside programmers.

Specific

- Image processing: The best libraries for this task are written for MATLAB. The fall down of this language is that it is not free, to use it you must purchase a license. Other language with good libraries is C++, though is not very agile for developing, which is bad for trying new ideas and the compromise with an agile methodology, here. Python is the middle ground between them both. It allows the programmer to use MATLAB style for numerical computing with libraries like numpy and scipy and have binding to libraries like OpenCV.

- MusicXML: The only libraries found that automates writing MusicXML are MusicXML Library and music21. The first one is written for C++ and the second one for Python, which proves the language election a good one.

4.4.2. High level design

In order to keep the software maintainable and extensible, it has been divided into four modules, three of which coincide with functional requirements.

The GUI module contains classes that manage the graphical user interface. It creates the different windows with which the user interacts and manage the call back methods for each action. It is also in charge of calling other modules in the recognition process per se, that is, it plays the role of the main.

The recognition module is conformed by a set of classes that recognize each element of the score. These classes are independent from one another, so that it is possible to change, remove or create one without interfering with the rest. In this version of the program, the recognition classes are bound to distinguish staves, notes, accidentals, rests and dots.

Finally, the writer module is the unit that sees to get all the collected information written into MusicXML [MakeMusic (2011)], the standard way of writing digital music sheets.

The module no specifically requested in the functional requirements is the object module. Its classes are storage structures for notes and rests, an intermediate step between the recognition of all symbols, that can modify the element, and their actual writing.

4.4.3. Detailed design

Diagrams

A UML diagram helps with the design and the understanding of the system. Here the class diagrams generated for each one of the prototypes is presented. The modules and methods are explained in the following sections.

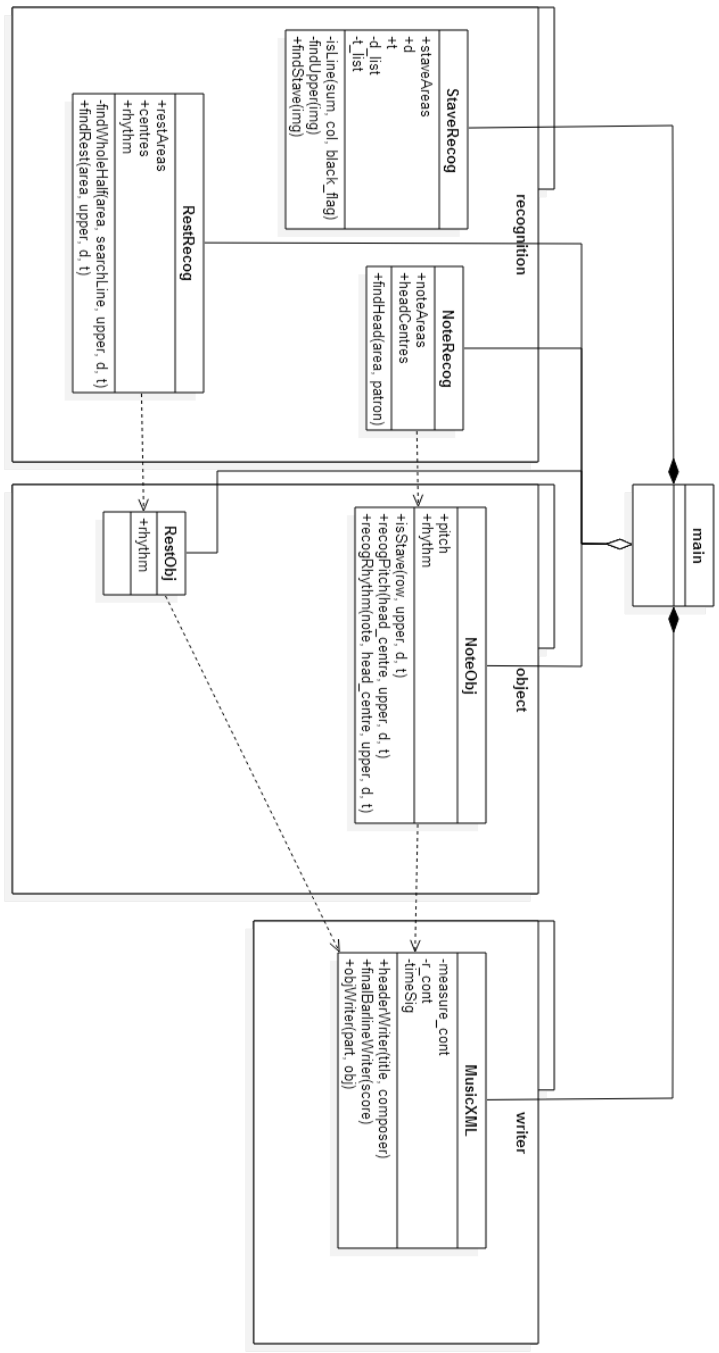


Figure 4.1: Class diagram for prototype 1

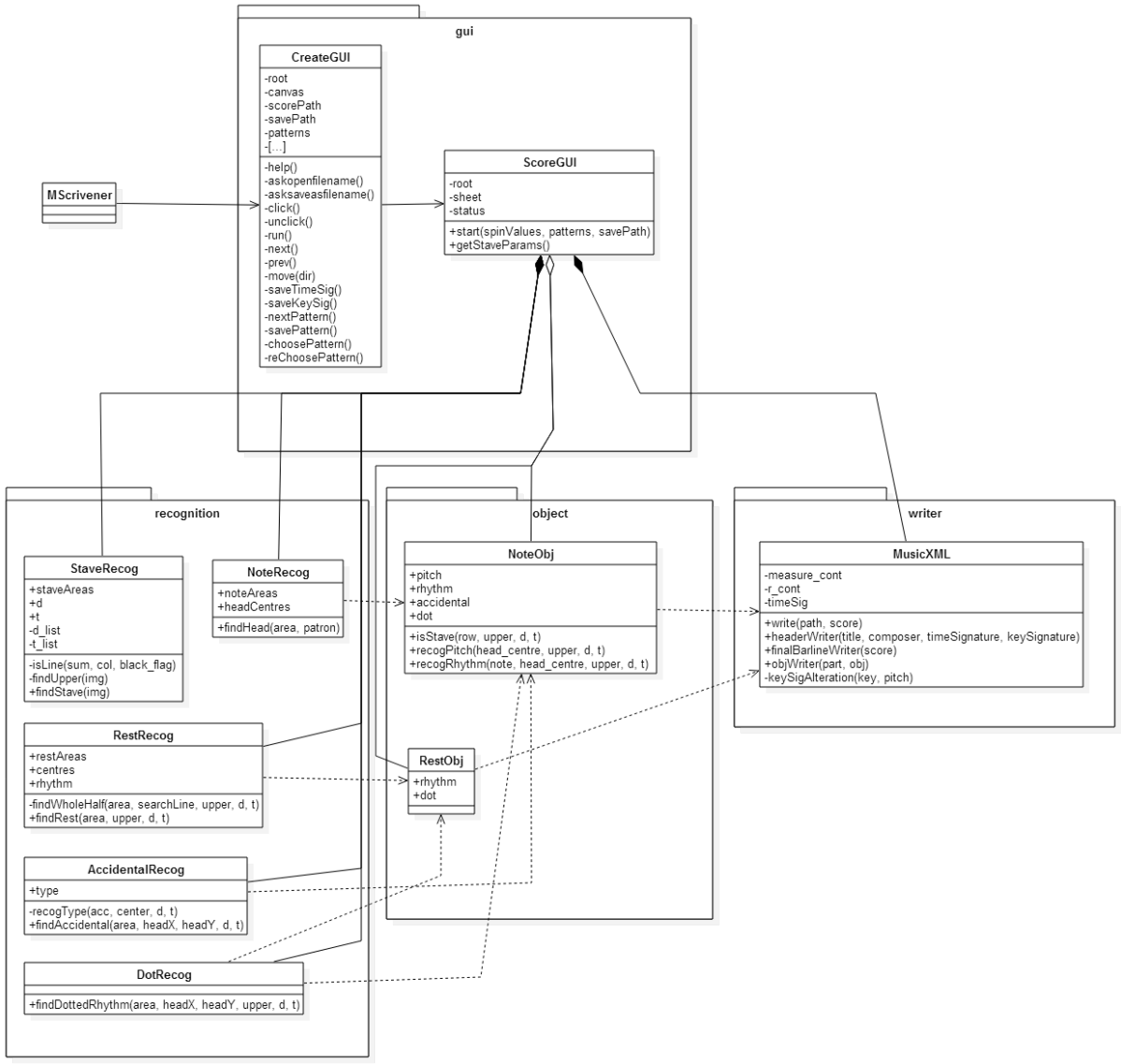


Figure 4.2: Class diagram for prototype 2

In order to control the time variable, a potential risk to all project and a special one to this, a Gantt diagram was design, and included here to show the temporal planning. The actual usage of time and with it the deviation from the planning is presented below.

For the sake of clarity, the Gantt diagrams are displayed in four sections: Initial tasks; design, coding and testing for both prototypes; and a whole overview.

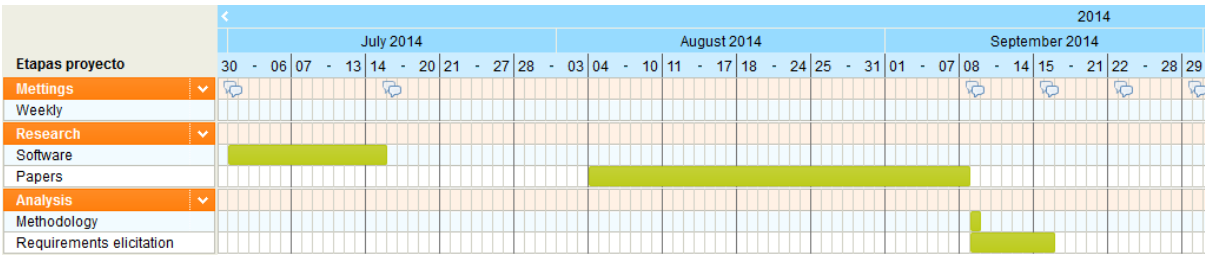


Figure 4.3: Temporal planning for pre-development activities

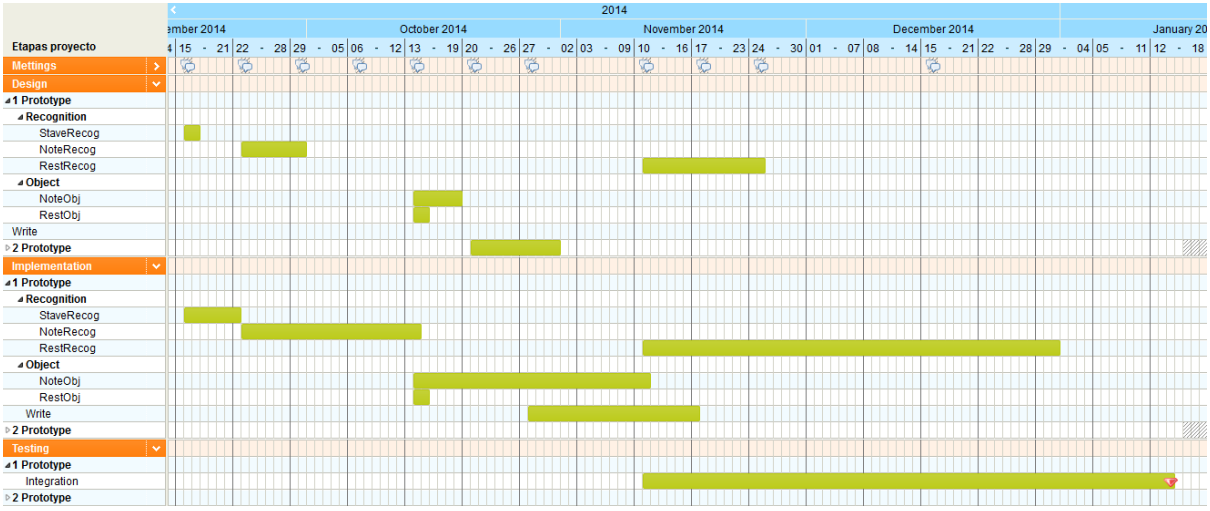


Figure 4.4: Temporal planning for prototype 1

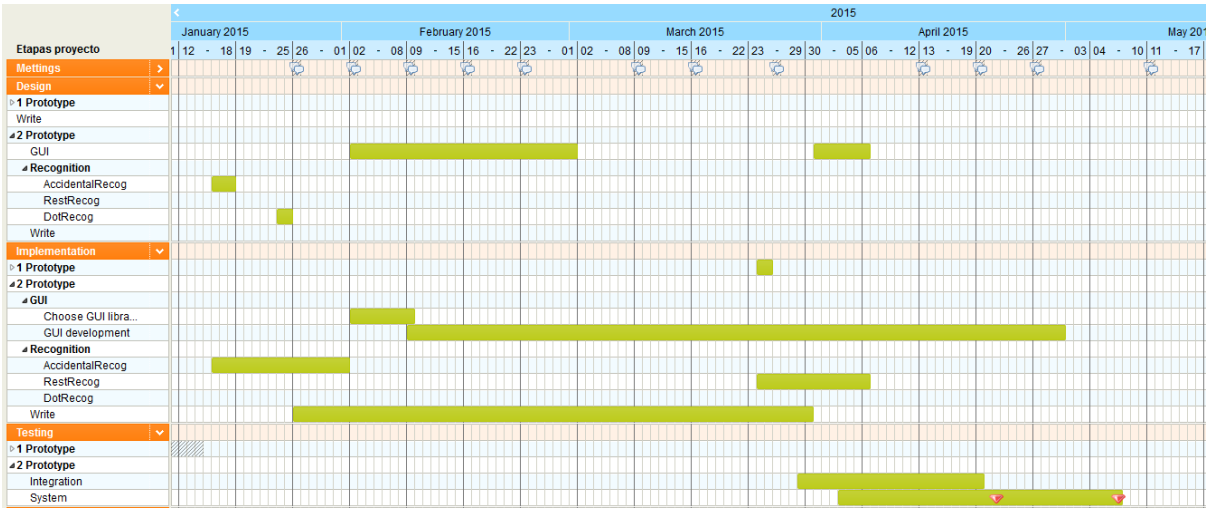


Figure 4.5: Temporal planning for prototype 2

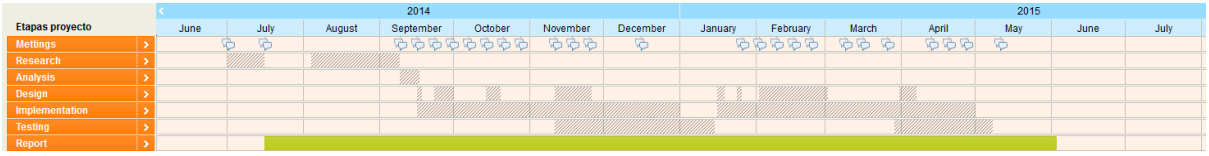


Figure 4.6: Temporal planning overview

The symbols in the shape of speech bubbles are marks to indicate weekly meetings with the tutor. The diamond shaped marks point out the end of a iteration in the first prototype and the workshops dates in the second one.

The unit testing does not appear explicitly because it occurs at the same time that each of the developments.

Gui

The software was designed as a desktop application. In the first prototype the interaction was made through the terminal and the patterns were given as preedited images. A complete GUI was developed in the second iteration of the development process.

The gui module is conformed by two classes, one creates the graphic interface, and the other one functions as the application flow controller.

- CreateGUI: The constructor allows to generate the main window and initialize the associated parameters. This class also contains auxiliary methods and callback methods. The former includes a method that calculate useful values or change the display.

  - calcPosition: Recalculates a given coordinate taking into account the window scroll. It is used to determine the position of a click on the image that displays the score.
  - changeImgCanvas: Changes the image in display in the main windows canvas. It is used to present graphically a new score when the user opens a file.
  - adjustArea: Tries to adjust an image area so that it is as close as possible to the head of a note. If it detects that the resulting area is smaller than the estimated size of a note head, it does nothing.

  The later consist of methods that have to be executed when interacting with the interface or that are called inside other callback methods.

  - help: Opens a dialog giving some help.
  - askopenfilename: Opens a window to search for the input file. Once chosen, saves the path and opens the image.
  - asksaveasfilename: Opens a window to choose how to save the output file.
  - click: Captures the click event.
  - unclick: Captures the unclick event.
  - run: Callback method that is executed when the user wants to start the recognition. It creates a wizard that will guide the user through the process of retrieving parameters and patterns.
  - next: Changes the wizard to the next window.
  - next2: Changes the wizard to the window after the next one. It is used to skip the windows that ask for a pattern if the pattern is no needed.
  - prev: Changes the wizard to the previous window.
  - move: Changes the wizard window and establishes the new buttons and messages to display.
  - saveTimeSig: Saves the time signature from the spinner boxes in order to pass it later to the writer.
  - saveKeySig: Saves the key signature from the spinner box in order to pass it later to the writer.
  - nextPattern: Indicates that the current pattern is not needed.
  - noBlack: Special case for the previous method. In case that there is no need of the black head pattern, as it is the last pattern required, the recognition process shall begin.

  - savePattern: Save the pattern chosen so that it can be used in the recognition process later.
  - choosePattern: Hides the wizard and activates the click and unclick callbacks to get ready for the user interaction that will lead to the acquisition of the pattern area.
  - reChoosePattern: In case the user decides that the pattern is not suitable, it discard the current pattern and calls the method choosePattern again.

- ScoreGUI: Its constructor receives some information from CreateGUI that will be needed through the class, like the main window (from where other widgets can be accessed) or the image that wants to be digitized and recognizes the staves in order to get thickness and distance between the stave lines. Save for the constructor, the class only has two methods, that are used by CreateGUI:

  - getStaveParams: Getter in order to reach the thickness and distance parameters of the stave. These parameters are needed to adjust the pattern area around the note head.
  - start: Calls for the different classes and its methods acting like a main in the recognition process. In order to add or remove functionality, this is the method that will have to be edited.

Recognition

This, and the following modules appear in the project since the beginning, though some of their classes were not. The recognition module has as many classes as individual symbols want to be recognized. The goal of the first prototype of the software is to obtain the melodic line. In this regard, the classes that were designed then were only Staverecog and NoteRecog and the first part of RestRecog. The remaining rests and alterations were included in the second iteration.

All of the recognition classes possess a find method. This is the main method called to recognize the symbol. They may also have auxiliary methods to determine within a certain symbol, their kind.

The symbols are divided between those who will create an object and those that are modifiers. The recognition classes of symbols that will create an object have public attributes such as the areas in which they appear and their vertical center.

- StaveRecog: Recognizes the position of the different staves along the image and calculates and/or stores some useful parameters such as the line thickness, the distance between lines or the position of the upper line in a stave and also a list of stave areas to analyze later.

  - findStave: Detects staves in an image using the auxiliary method isLine. It also calculates the thickness and distance parameters as an average value of all the gathered information.
  - findUpper: Calculates the position in which the upper line of a complete score or stave area is situated. It is useful to provide the program with a context in later stages.
  - isLine: Identifies stave lines.

- NoteRecog: Recognizes the position of different notes along a stave. It saves into variables the image area for further analysis and the head centers (horizontal and vertical). With this class information, the program will create an intermediate representation of a note in which accidental and dotted rhythms will modify its content before the writing.

  - findHead: Searches for the different pattern repeats, saves its area and deletes the image from it in order to avoid noise afterward.

- RestRecog: Recognizes the position of different rests along a stave. It saves into variables the image area for further analysis and the head centers (horizontal and vertical) as well as the duration of the rest. For rests there will also arise an intermediate representation. In this case, rest will not be modify by accidentals, but only by dots.

  - findRest: Scans looking for all the rests, whatever their type. Following the example given in the previous class, it saves its area and deletes it from the original score.
  - findWholeHalf: Auxiliary method that distinguishes between whole rests and half rests. It is called by findRest.
  - findQuarterOrSmaller: Auxiliary method that distinguishes between quarter and smaller rests. It is called by findRest.

- AccidentalRecog: Recognizes if there is an accidental for a note, and if so, its type is determined. This class can be extended in a future to recognize double accidental.

  - findAccidental: Scans the area next to a head note in order to conclude if it is modify by an accidental or not.
  - recogType: Once the accidental is found, this method resolve its kind. It is called by findAccidental.

- DotRecog: Recognizes if there is a duration alteration, namely a dot. This class can be extended in a future to recognize chained dots.

  - findDottedRhythm: Scans the area next to a head note so to determine if the note has a dot.

Object

All the classes have a data structure that has to be filled and that has to be able to be edited. Each class has a duration and dot variables which are their common features. For notes in particular, it has a pitch and accidental variable too.

As explained in the previous module, both classes were present in the first iteration of the life cycle, though not completed.

- NoteObj: Creates an note type object, characterized by its pitch and duration and modified or not by accidentals or dots. The constructor initializes the structure values.

  - recogPitch: Recognizes how high is the note head placed in the stave and translate this into a pitch, storing the value.
  - recogRhythm: This method is only used for black headed notes, because for whole notes and half note, it is already known which is the duration through the pattern used.
  - isStave: Auxiliary method that distinguishes a stave line from other symbols. It is used by recogRhythm.

- RestObj: Creates a rest type object characterized by its duration and modified or not by dots. Its values can be modify via its public attributes since it has no methods save the constructor.

Writer

This module has only one class in charge of writing all the necessary information. The need of an actual output made this module and class critic and it is the reason why it appears in the project almost since the very beginning.

- MusicXML: It is the only class of the module. Its constructor initializes values that control how many notes are and must be within a measure.
  - write: Dumps all the stored data into a MusicXML file, that will be the output of the software.
  - headerWriter: Set the meta-data and structure information for the score: name, composer, time signature and key signature.
  - finalBarlineWriter: Writes a final barline at the end of the score.
  - objWriter: Writes the objects obtained in the previous module into the score.
  - keySigAlteration: Keeps in mind the accidentals established by the key signature and the other modified notes in the same measure to add an implicit accidental to a note.

4.4.4. Test plan

Even though the testing phase is interlocked with the design and the codification primarily, the structure and planning is clear: There are three stages in the testing process, the first one is the unit testing, that will prove that the code is correct; the second one is the integration testing, that will confirm that the modules work together; and lastly the system test will verify that the software runs in multiples platforms and environments.

- Unit testing: During the development of each class, and even for each method, small proves are tested to check every thing works as intended. Some of the unit tests designed are obtaining the number of staves (StaveRecog), notes (NoteRecog) and rests (RestRecog) in a score; the getting the pitch and duration of all notes in a stave (NoteObj) or reading the output with a music editor (MusicXML).
- Integration testing: This is really an incremental integration testing. The addition and testing of different modules is done gradually, so in case of a failure, delimit the error. The first test are based on terminal written output and once the MusicXML class is functional, they can be switched to human comparison between the original image and the XML open by a music editor.
- System testing: With the last prototype completed, it is necessary to test if the software can be run in different operating systems, without the libraries nor Python installed. The first system test is having place at the some computers to which the author has access and the final test will be with real users and their machines.

4.5. Development

This section will describe module by module the libraries and algorithms behind the implementation of the software, keeping in mind that the solution implemented is based on heuristics devised by expertise.

4.5.1. Gui

The requirement of user friendliness [**NFR.2.**] forced the development of a graphical interface that could be easily used by any musician no matter how versed in computer science. The idea was that even the most computer fearing users could install and use the software. To accomplish this, the disposition of menus and the design of the wizard was as standard as possible.

The library used to develop the GUI is the Python's de-facto standard GUI, Tkinter. This allow us to hope that the library is not prone to disappear or fall off maintenance.

- CreateGUI: Uses the library to create a window with a menu and a status bar. In the center of the window there is a canvas that changes with the score election of the user. Almost all of the widgets are data attributes, owned by the class so that the Python garbage collector does not eliminate them. The opening of images is done through another Python library, PIL, compatible con Tkinter.

  Tkinter does not have a special widget to create wizards. To accomplish the task, an auxiliary window is created, and its content stored in an array. Every time that the wizard moves forward or backward, the window content is updated depending on the page it is supposed to be in. Some pages of the wizard also have a image attached, that can be a example of the pattern intended or the actual pattern chosen. In the first case, the images are stored withing the program and merely loaded into the window. In the second case, the user is force to select the pattern. To do it, the methods that respond to click events are bound and unbound as soon as the pattern is selected. The wizard is hidden and the user can select their patterns. Then, the program will adjust it taking into account the head size. Having ended the wizard guide, the recognition process can begin calling the start method of ScoreGUI.

- ScoreGUI: Its constructor uploads the image to analyze and locates the status bar that will update the information for the user. Then finds the staves in the score and shows its number through the status bar. The algorithms used for the recognition will be explained in the following module, here will only be explained the information flow and call order.

  The other main method in this class is the start method. It is in charge of calling the rest of the modules as follow: configure the writer using the class MusicXML and the time and key signature obtained through the GUI. Then the recognition begins. For every stave, it finds any note, distinguishing between whole, half and back by means of the patterns gathered. For each note, it identify its pitch and duration, though for the two first types this is trivial. Until the second iteration this was all the recognition that took place, followed by its writing.

  In the final product the note recognition was followed by the accidental one, that modifies only noteObj's, and then rests and dots, that modifies notes and rests alike. Given the non-sequential behavior of the note recognition, at this point it is necessary to sort the objects before calling to the MusicXML method objWriter. The last step to finalize the process is to add a final bar to the score and dump it into a file stored in disk.

Its development has taken around 30 clock hours and last almost 4 months, since the beginning of the second iteration till its end.

4.5.2. Recognition

The base of the recognition module has two pillars, one is the chosen image processing library, OpenCV [Bradski (2000)], and the other one is the series of developed algorithms explained

in the previous section [3]. Once the symbol is recognized, it either creates an intermediate representation of it, or modifies an existent one.

- StaveRecog: The finding of the stave is based in the algorithm Horizontal Projection [3.3.1], read in the literature. The space between the stave is equally distributed among the two areas and at the end of the recognition the thickness and distance between the stave lines is calculated as an average taking all the values observed.

  The subareas where the stave can be found are stored into a list to loop through in order to find the rest of the symbols in it. The upper variable is calculated for each stave area according to the same algorithm.

- NoteRecog: This class has only one method in charge of finding all occurrences of a given pattern. This is achieved by using the Template Matching algorithm provided by OpenCV. It has a threshold of similarity set to a empirical value that avoid the flaw of finding only the exact match.

  All the matches are stored in lists, saving its area and head center coordinates. The algorithm does not work sequentially, so these values will be used, among other things, to sort the objects later. The occurrence of a note is remove from the original stave area in order to eliminate potential noise of successive recognitions.

- AccidentalRecog: This class owns two methods, the first one discovers accidentals, and the second one discriminate them by their kind. The early one, is based on the knowledge that an accidental only appears before a head note. So, if searching that area (contextualized by the head center coordinates and the size of the head), noise is found, the recogType method can be called.

  In this case, a Character Profile algorithm type, read in the literature, is used and explained with detail in another chapter [2.2.1]. It suffices to know at the moment that the type of the accidental is stored to modify the noteObj next.

- RestRecog: The main method of this class seeks along a stave area in places where the possibility of finding rests is higher. If an important amount of black pixels is found, the method calls its auxiliary functions and if they are really rests, fulfills the same routine as noteRecog, that is, saves its area and centers, but also the duration of the rest.

  The recognition of rests is implemented via a Signature Analysis and a Character Profile algorithms.

- DotRecog: Works under the same principle as AccidentalRecog, without the second part. If a dot is found, it affects the object to its left, either note or rest.

The whole approximation of real work has taken 28.5 hours, an average of more than 4h per class, but for RestRecog, that was divided into smaller and grater rests, taking more than this average each one. The development process of this module lasted for 7 months divided into two prototypes, the first one since September to January and the second one from January to May.

4.5.3. Object

This module works simply as a storage structure, closer to the music21 representation. In the note case, it is also necessary to find out its pitch and duration if unknown.

- NoteObj: As a structure, it stores the pitch and its modifiers, accidentals, and its duration and modifiers, dots. But before, there is the need of identify both the pitch and the duration.

  The pitch is determined using the upper line of the stave and the head center of the note; the relative position of the head in the stave provides the program with the information sought.

  The duration, on the other hand, is found using an expertise based algorithm. The note stem is identified and searched around to try to find divergences that are not the line staves.

- RestObj: The rests duration are already identify in the previous module, so this class is only a storage structure that saves its duration and modifications, dots, if any.

The whole module was codified in 8 hours, being the RestObj time negligible. It took 12 days.

4.5.4. Writer

Module that has only one class, MusicXML, that encapsulates the music21 library making its use more comfortable. Its methods help converting information to music21 library objects and dump its content into a real file in disk. It also deals with the amount of notes and rests stored into a measure and the number of accidentals in the key signature in order to ensure consistency. The methods that handle this are keySigAlteration and objWriter.

The first one receives a note and extract its pitch. With it, compares to the key signature and returns if the note is affected by it. The second one converts the object to a music21 object and then depending on the time signature, it keeps on adding objects until the measure is complete; when it is, it changes to a new one. An accidental within a measure affects all the notes with the same pitch, so it is necessary to hold a record of all the measure accidentals and check if a note is affected by them.

The MusicXML conversion, dealing with the music21 library was solve in around 10 hours. The extension of the work took almost all the project, around 7 months, since the writing of the first note to the resolution of the last symbols.

4.6. Validation suite

In order to run the tests, it is necessary to have a validation suite. The idea of this suite is to have five levels of complete scores plus a initial stave are to test at the beginning excluding the last one, this is the validation suite chosen. The first three sets are referred to ideal scores, taken from a database or from the user owned digitized scores.

- Easy: Simple scores with no accidentals, rests, dots, nor anything out of the first prototype scope.
- Medium: Scores more complicated but without annotations nor time/key signatures changes.
- Difficult: Any complete score.
- Scanned: If all the previous tests have been successful, then the difficulty of the score is out of concern. The idea of the suite is deal with low quality images, skew and similar problems.

- Handwritten: The last and more complicated examples are the handwritten ones. They have the scanned documents problems but also a new one, that is learning from the actual symbols drawn by each person. This set will not be tested, because is out of the scope of this work, but is interesting to keep it in mind.

This suite is not only chosen for the current project, but also for the future work to do. Examples of this scores can be found in the results section [5.2] along with its digitalization.

4.7. Software management

In order to maintain the project a "systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software" [IEEE (1990)], it was necessary to track all the process. For this particular project, it has been used the following methods and software to help them.

4.7.1. Documentation

Though using an agile development where the documentation was not the central pillar, the author thought it important for the open source ambition. It was also important in order to describe the process in this report.

To design a time plan and also to keep a record of the time spent in the different tasks, an online platform was research so that in any computer at any time the access was possible. To this avail, the selected platform was clockingit [Simonsen and Simonsen (2003)], a free application of project management that allows multiple access and communication for a whole work team, even though in this case it was not necessary.

Although this platform already has a Gantt chart feature, it is not as complete as it seems, so this particular diagrams [4.3] were build by another online tool, Tom's planner [Toms Planner (2009)], also collaborative and free. Additionally, it has a clean and straightforward way of picturing the information, which helps to its understanding.

For the design of the software and the construction of the UML diagram [4.1] the chosen application was StarUML, familiar to the author because of it previous use.

In the development stage, the IDE used has been Eclipse with the plugin PyDev for Python development. Another important issues during this stage is the code documentation, and as a result of this, there is a whole API of the program generated by Doxygen in HTML and LaTeX.

Finally LaTeX was also chosen to write this report because of its typographic quality and as a new tool to learn and out stand in the labor market. The platform used for writing LaTeX is Overleaf (former WriteLaTeX) [Writelatex (2011)], a collaborative writing and publishing system that allows you to view the result of your typing in real time.

4.7.2. Code

About code management, the first idea when the project was thought to be a partnership with another student was to use GIT as a version control software. But given the final decision to turn it to a single project and the difficulty of finding a good private repository lead to decide for a Dropbox management eventually. Each version of the project was handled like a commit, with the number of version and some comments.

CHAPTER 5

TESTS AND RESULTS

5.1. Introduction

In this section are presented the results of the validation suite [4.6] and a comparison between the results achieved by the developed software versus the applications researched in the chapter *State of the art* [2.3]. A concise description of the workshops conducted is also presented before a summary of the results.

5.2. Tests

Following the structure set for the validation suite, in the next pages the reader can find the actual output for a example of each one of the score levels with a description of the pros and cons compare to the original input.

The original scores come from the MuseScore database as ideal example of scores. This scores do not have skew issues, or noise related to the scanning process.

5.2.1. Easy score

As explained before, this is the validation test objective for the first prototype, and the representative score used to test it is *Variations on "Ah vous dirai-je, Maman"* by Mozart, also known as the famous *Twinkle, Twinkle, Little Star*. The version used is found in a book for beginners called Suzuki Method [Suzuki (1978)].

As can be observed in the figures below, the recognition is complete according to the started scope. The only non recognized symbols that appear are the instrument fingerings, which are the small numbers above some notes, and the small dots placed above certain notes called staccato.

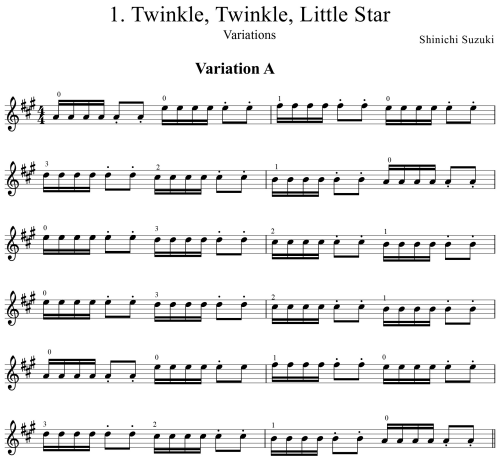


Figure 5.1: Mozart input

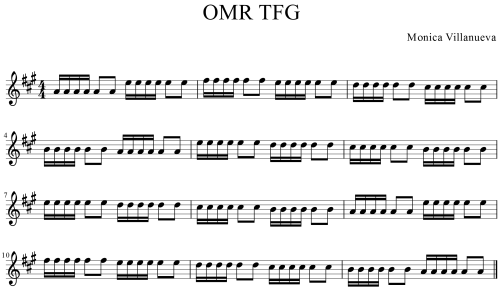


Figure 5.2: Mozart output

5.2.2. Medium score

It is difficult to find real scores that include the symbols to recognize leaving out the ones outside the scope of the project. Taking this into account, the solution given was to find scores with few non-covered symbols and avoiding to take into account those mistakes in the evaluation of the results.

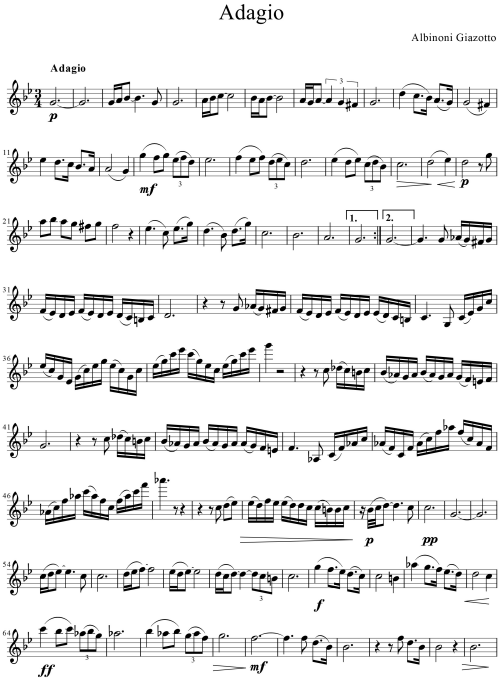


Figure 5.3: Albinoni input

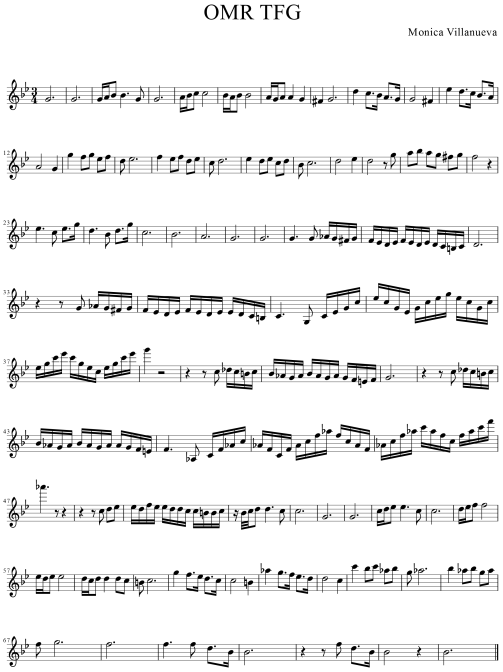


Figure 5.4: Albinoni output

The score chosen here is the *Adagio in G minor* by Albinoni, better known as the *Albinoni's Adagio*. The test proves that the recognition is complete. In this case, the differences (but not

errors) are due to complex rhythms, not covered by neither prototype and editable once the user has the output.

5.2.3. Difficult score

These scores have no limitations except that they have to be ideal, that is, downloaded for the MuseScore database or obtained from a MusicXML file. This way the original score will not be skewed or have defects from the scanning process. The set is designed so that testing it in the final prototype, its error are apparent and also, to asses the performance of future releases of the software once some of the future lines of work [6.2] are solved.

The score selected for this test is *Salut d'Amour* by Edward Elgar. As it can be appreciated from the figures, recognition is quite faithful. There are minor problems with some extremely high notes and some noise mistaken for rests. The remaining differences are due to uncovered functionality, such as annotations or harmonics (measure 37), because they are not half notes.

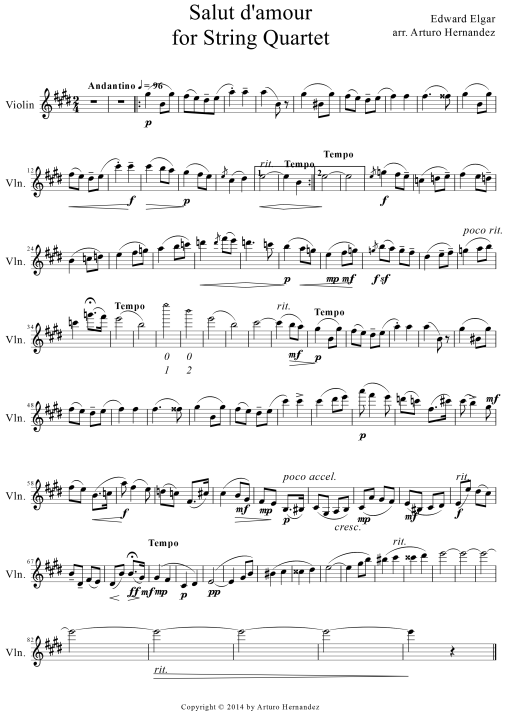


Figure 5.5: Elgar input

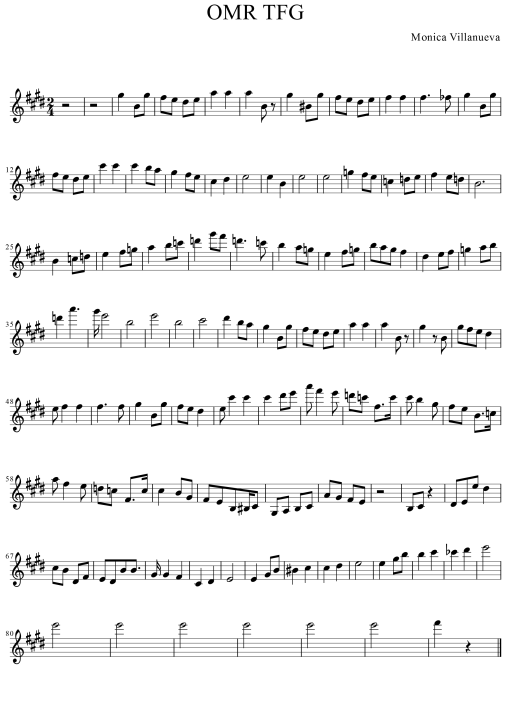


Figure 5.6: Elgar output

5.2.4. Scanned

The application has difficulties with scanned documents because the recognition algorithm is based on the horizontality of the stave lines. For this reason, parts of the score can be correctly digitized while the others may be higher or lower in pitch and have duration problems.

The example showed in figures 5.7 and 5.8 is an excerpt from the *Concerto in D minor for two violins* by Bach. In it the reader can see that the score is well recognized by fragments. In some places, there are rests that are simply noise in the scanned score.

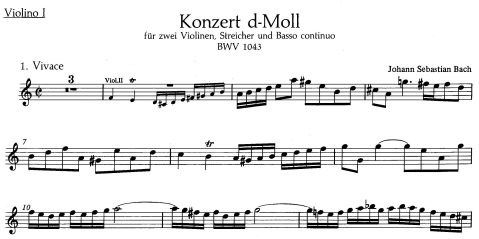


Figure 5.7: Bach input

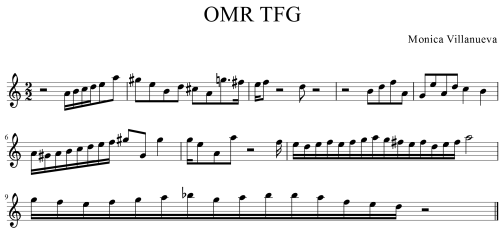


Figure 5.8: Bach output

5.3. Benchmarking

This section is meant to see the differences between the studied software and the developed one with the complexest and better recognized score, in this case, *Albinoni's Adagio*. The comparison is structured using free software first and the proprietary one later; since the last one works better, the similarity of outputs should grown until the possible perfection of the PhotoScore example, that would distance the resemblance again.

5.3.1. Audiveris: Open source

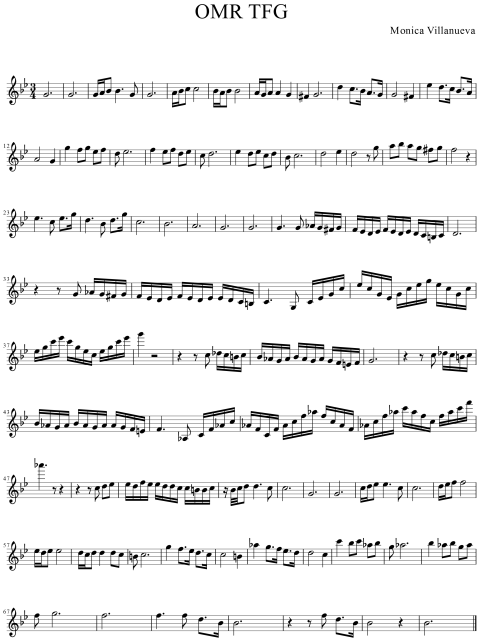


Figure 5.9: Albinoni MScrivener

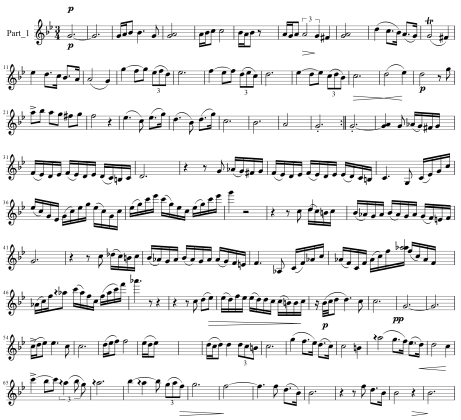


Figure 5.10: Albinoni Audiveris

The recognition mistakes committed by this software are exactly the ones tried no to reproduce in the developed application, namely try to recognize too much confusing symbols.

MScrivener identify notes and durations better than Audiveris. The reader can appreciate the difference examining measure number eight (recognize two notes) or fifty seven (the measure is not even complete).

It is true that Audiveris has a wider recognition, including ties and annotations. An example of both of them can be found in the first measure.

5.3.2. SharpEye: Proprietary

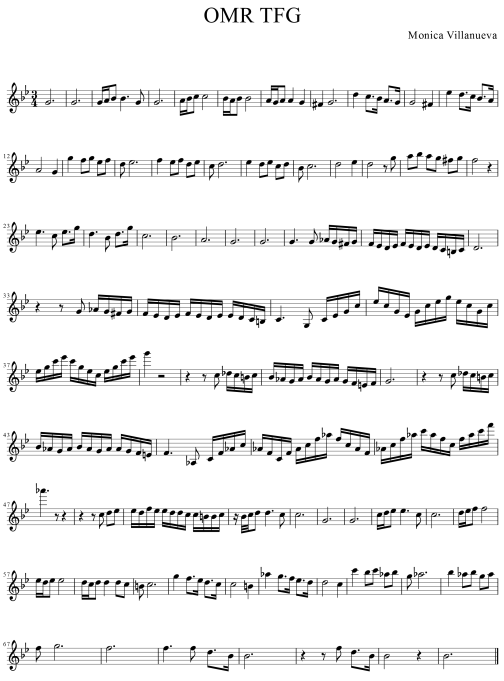


Figure 5.11: Albinoni MScrivener

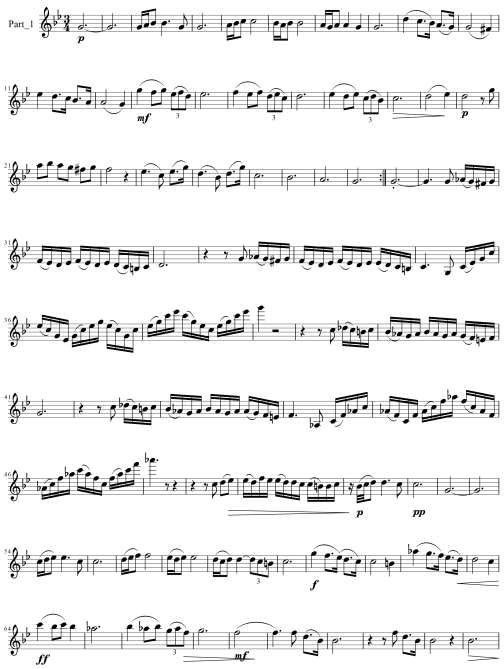


Figure 5.12: Albinoni SharpEye

This program is better than the previous one; as analysed before, the propriety software works better than the open source project available right now.

As a whole, the SharpEye seems to work perfectly, but looking more closely, it can be spotted a minimum loose of information, as in the seventh measure or the sixty fourth.

5.3.3. PhotoScore: Proprietary

PhotoScore seems to be the best option regarding notes and durations, though the rest of the annotations appear more cleanly in SharpEye.

The comparison between the developed software and this one makes no sense as such, but only as a future reference.

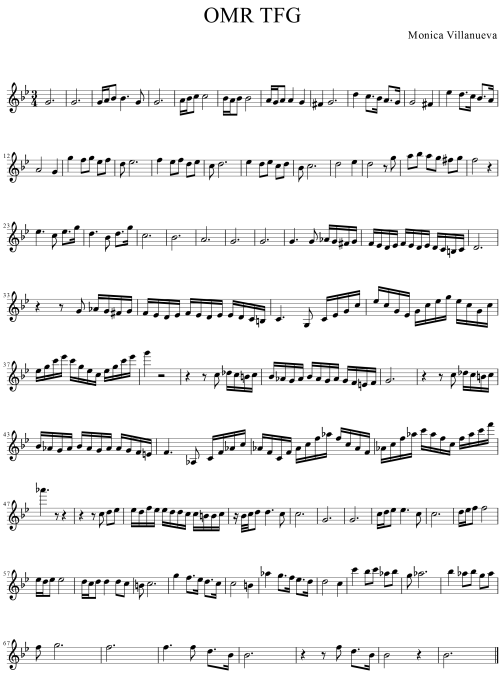


Figure 5.13: Albinoni MScrivener

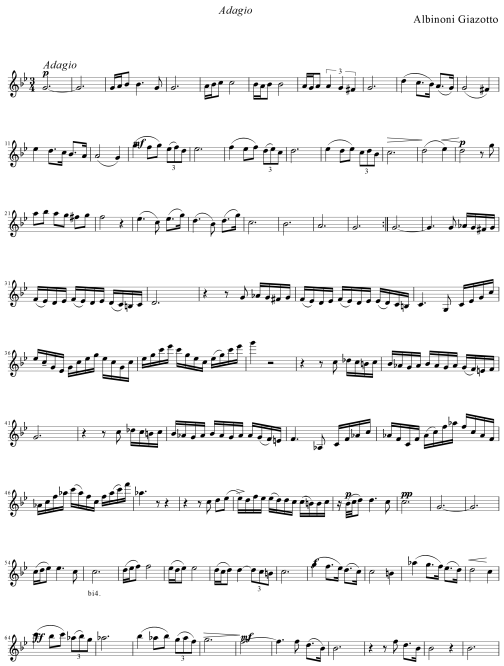


Figure 5.14: Albinoni PhotoScore

5.4. Workshops

To test the application in a real environment and retrieve data from real users, some workshops were organized in different surroundings. The users invited to participate in these workshops have similar, but different needs. The following sections describe the process and responses of the users in each workshop.

5.4.1. Universidad Autónoma de Madrid

On April 22, 2015 two workshops were carried out at the UAM. The first one, in the morning, was held with music teachers. The second one, in the afternoon, was imparted to a class of their students.

Both talks had the same structure. First a presentation of the project, explaining its motivation and possible uses, a brief description of the algorithms used and illustration of the functionality of the application a limitation through examples. At this point, during the workshop with students, the author conducted an experiment: letting a volunteer use the software without any help, to discover if the tool was really usable. The result of this experiment showed that the original mechanism for identification of pattern unnatural. This procedure required the user to draw a rectangle around the head dragging the mouse from a corner to its opposite. For this reason, the selection process was changed to a more natural one before the talk at the conservatory. This method consist in clicking in the center of the head as explained in 3.3.2.

The initial plan was to have the attendants to use the software and fill out a survey about the workshop. Unfortunately computers at the laboratory in the teaching faculty have problems

with portable executables, which made it impossible. Instead, the software was distributed among the participants under the promise that the would try it at home and fill the form.

The most attractive feature of the application was the possibility of using if free of charge and with almost the same functionality as the commercial software. Although what really interested them was its potential to recognize hand-written scores, reducing the time needed to digitize them in order to use them in the classroom.

The complete responses can be found in the appendix C.3. A brief summary of the outcome is that all users were knowledgeable with music editors, though not all of them were familiar with OMR's. Everyone agreed that the software was easy to retrieve, execute and use, despite the fact that it is written in English. As their fundamental use was edition, they found the parameter and pattern selection awkward.

No bugs were reported. Some of the polled admitted that they probably will use the tool in future.

5.4.2. Conservatorio Profesional de Música Arturo Soria

The talk at the conservatory took place on May 7^{th}. The structure was the same as the one used in the UAM workshop. The students used their own computers in the practical part of the workshop.

In this environment, there were two suggestions for improvement: the further development of the composition tool and a possible modification to extend the recognized symbols in order to cover special notation for flamenco guitar.

As before, the results of the survey are available in the same appendix [C.3]. The conservatory lecturers and students are acquainted with both editors and OMR applications. There was overall agreement in the ease of use of the software. In this case, the parameter and pattern request is useful and appreciated because of the interest in algorithmic composition.

Once again, no bugs were reported. All of the surveyed at the moment of writing this report, expressed their willingness to use the software.

5.5. Results

The overall results of the project are fairly satisfactory. Te assessment of the suite of test scores shows that the original goals have been fulfilled. The comparison with available software both commercial and open source is favorable within the scope of functionalities. Finally the workshops highlight the users interest and satisfaction on the developed application.

The test by itself only proves that the functional requirements [4.3.1] are cross-validated. On the other hand, the benchmarking show how and where would be the software in the market, which are its strengths and what is the way forward to improve the tool. A proposal for future developments and improvement can be found in the next chapter [6.2].

At this point following the application developed is better that even proprietary software in some areas and could be competitive depending on the objective of the recognition. It also have the compositional aspect that not one of these applications consider: changing the patterns selected as heads or modifying the signatures from the real value can alter a original score into something different.

Finally, the experience with potential users have confirmed the original idea that the target public is not really large, supported by the number of people that have actually filled the survey.

Either way, the event showed that the non-functional requirements [4.3.2] have been met, in the same way the test suite validated the functional ones.

CHAPTER 6

CONCLUSIONS AND FUTURE LINES OF WORK

6.1. Conclusions

The project started a year ago as a vague idea put forward by the tutor, and worked out through a summer of research in the field. The expertise of the author and the combined efforts of the tutor and author meeting by meeting turned a nebulous project into the reality of what, today, MScrivener is.

It is far from a closed project. However, the work and result achieved is positive. Here, notions learned throughout the degree have been used, some that may be named are the ability to manage a software development project, taking into account the limitations and possible risks, designing one or more solutions, codifying them in a clear a efficient way and testing them. It has also been an enriching process where the author has learned more advanced things such as doing research or writing a technical report properly shaping (in a non native language) the flow of thoughts into written words.

As a musician herself, the author, this project was not just another work to hand over, it was something personal: build a software with application for musicians, composers, conductors and any amateur that could be interested. Despite the advice of many to make the software private, the decision of making it completely free and open source was made. The main reason was that in this manner, it is accessible to anyone, regardless of their economic situation. Without marketing support, its impact and dissemination will be grater if the application is free of charge. It also is easier to receive help being an opens source project.

The solution achieved in the first prototype was already promising. The main problems were resolved and a simple melodic line was recognized and written in MusicXML. With this first prototype as a prove-of-concept and some more development, the author organized a workshop in the Conservatorio Profesional de Música Arturo Soria and with music professors and students at the Universidad Autónoma de Madrid. With the second prototype finished and the software turned into a stand-alone executable under Windows and Linux the workshops were held with a more successful outcome than imagined. The participants ware enthusiastic with and its potential.

The endeavor and hard work hours have unfolded an endless amount of lessons whose learning will pay off for a lifetime. It is worth mentioning the following:

- Learning new skills: It includes competences intrinsic to the project, such as the grasp of different optical recognition techniques and the usage of diverse libraries, but also other abilities that will be useful in the labor market, like the learning of a new programming language, Python, and LaTeX as a way of generate clean an professional reports or research papers.
- Devise multiple solutions: For each challenge it was usual to design more than one possible solution in case that one of them was not successful or had poor performance. Some of these solutions were complex given the scope of the project. Some were relegated to future lines of work, but were never given up on as impossible.
- Reconcile with classes and work: While the development of the project, the author had to reconcile it with classes and work. This will be useful as a mean of balance work and private life in the future, as well as receive an idea of what could be to develop a personal project while working.

As a whole, the project is considered a success.

6.2. Future lines of work

The time constraint was the most important variable to take into account in the development of the project. Since the aim was to achieve a functional tool for musicians, the application was focused to minimize failures, gradually adding new functionality.

The high complexity of OMR has made it impossible to cover everything. Nevertheless, some of the challenges have already been analyzed and partially or completely addressed.

6.2.1. Developed functionality

There are some issues that are not completely solved. They can result in small disagreements with the original score: anacruses, double bars and skew in the image. All of these problems are already designed and coded, but are not integrated nor tested. This is the reason they have not been included in the final prototype.

The anacrusis, also known as a pickup, is an incomplete measure that can appear at the beginning of a score. The solution used for this issue is based on finding the first barline and changing the measure for the next symbols found. The algorithm used to find barlines is based on the Vertical Projection algorithm described in section 2.2.1.

The detection of double bars is similar to the one before, except that the algorithm must find the second line. It can also be applied to discover repeated bars if the algorithm also finds the associated dots.

The last problem encountered was the skew of the scanned scores. The complication is caused by imperfect paper alignment in printers and scanners. To scan a score, it must have been printed before. Both printers and scanners introduce can a skew. This is the only problem that it is not coded yet. Nevertheless some research has been made. To fix the skew of the image, OpenCV has certain functions that can be used, such as the Hough transform or the Fourier transform, as described in the literature [Carretero de la Rocha (2009)].

6.2.2. New functionalities

There are some new functionalities that were requested by participants in the different workshops but have not been developed yet: automatic recognition of the key, changes in the time signature, complex rhythms and scores with more than one part.

The last one is the easiest one. In a score written for more than one voice there is a name at the beginning of the stave; counting the number of staves that has a name before them will give the information needed. In the stave analyzing loop there has to be a data structure that stores each part separately depending on the iteration and number of voices.

Complex rhythms and time signature changes demand the digital recognition of digits. In the first case, these numbers appear above the notes or rest, so it is know which parts are affected. In the second case, the changes in the time signature appear at the beginning of a measure. Searching those areas is the optimal way to find them. Besides, it is important to choose an appropriate recognition technique: Template Matching for notes having samples stored inside the software or using a Neural Network.

The key recognition can be implemented using any of the methods just described.

6.2.3. Machine learning

The idea of using machine learning in the project has a large potential. For instance, a neural network can be trained to recognize any of the musical symbols of a score.

Another use is learning from past errors. It would be necessary to develop a new module that shows the score as a music edit does. Once the score has been digitized, ask the user to give feedback about the errors in transcription. That way the software itself can learn from its errors. Also, the data could be sent through the Internet in order to reach all the users, so that it can learn faster and from a wide rage of examples, and not be biased by particular users.

As in OCR, a Neural Network can be used to train the program so that it can understand had-written symbols [Ganis et al. (1998)]. The pattern selection technique can be used to ease the difficulty of the training for a novice user.

6.2.4. Target users

In the workshops, two separate types of target users were discovered: editors and composers. In order not to distract the previous with the parameter and pattern choosing or limit the work of the later eliminating them, the best solution is to divide the application and specialize each one of them separately.

In a more or less distant future, with both tools develop, it can be possible to devise a social network to share converted scores, or new creations composed with this second program, managed by a sort of online library in which the scores Id can be a QR code. That way, if a user came across a digitized score, it can access the original scanning the QR.

These are not unique solutions to these problems, but they are a good approach for the work yet to do.

BIBLIOGRAPHY

Arshad, Q.-A., Khan, W. Z., and Ihsan, Z. (2006). Overview of algorithms and techniques for optical music recognition. *Department of Computer Science-COMSATS Institute of Information Technology, Wah Cantt, Pakistan.*

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mallor, S., Shwaber, K., and Sutherland, J. (2001). The Agile Manifesto. Technical report, The Agile Alliance.

Ben-Dayan, B. and Giloh, I. (2013). Optical Music Recognition.

Bradski, G. (2000). The opencv library. *Dr. Dobb's Journal of Software Tools.*

Byrd, D. and Schindele, M. (2006). Prospects for improving omr with multiple recognizers. In *ISMIR*, pages 41–46.

Carretero de la Rocha, D. (2009). Sistema de reconocimiento de partituras musicales.

Chung, Y.-C. and Lee, G. C. (1995). *Recognition of Printed Sheet Music Using Hough Transform And Morphology Operations.* PhD thesis, Master thesis, National Taiwan Normal University, Taiwan.

Fujinaga, I. (1997). Adaptive optical music recognition.

Ganis, M., Wilson, C. L., and Blue, J. L. (1998). Neural network-based systems for handprint ocr applications. *Image Processing, IEEE Transactions on*, 7(8):1097–1112.

IEEE (1990). IEEE Standard Glossary of Software Engineering Terminology. Technical report.

MacMillan, K., Droettboom, M., and Fujinaga, I. (2002). Gamera: A python-based toolkit for structured document recognition. In *Tenth International Python Conference*. Citeseer.

MakeMusic, I. (2011). Official musicxml website. Accessed July 2014. `http://www.musicxml.com/`.

OpenCV (2014). Image thresholding. Accessed July 2014. `http://docs.opencv.org/trunk/d7/d4d/tutorial_py_thresholding.html`.

Python, S. F. (1997). Comparing python to other languages. Accessed July 2015. `https://www.python.org/doc/essays/comparisons/`.

Sheridan, S. and George, S. E. (2004). Defacing music scores for improved reognition. In *Proceedings of the Second Australian Undergraduate Students' Computing Conference*. Citeseer.

Simonsen, E. and Simonsen, E. (2003). Official clockingit website. Accessed October 2014. `http://www.clockingit.com/`.

Suzuki, S. (1978). *Suzuki violin school: violin part*, volume 1. Summy-Birchard Inc.

Toms Planner, N. (2009). Official tom's planner website. Accessed May 2015. `http://www.tomsplanner.com/`.

Vigliensoni, G., Burlet, G., and Fujinaga, I. (2013). Optical measure recognition in common music notation. In *ISMIR*, pages 125–130.

Writelatex, L. (2011). Official overleaf website. Accessed July 2014. `https://www.overleaf.com/`.

APPENDIX A

BASIC MUSIC THEORY

A.1. Introduction

This annex is meant for readers who do not have a deep knowledge of music theory or score reading. The goal is not to cover all of the symbols that can be found in sheet music, but to understand the problems posed by their recognition. The appendix is also used as a glossary of the musical terms employed in the report.

A.2. Structure

This section is devoted to those symbols that parametrize the score and its possible variations.

A.2.1. Stave

A stave is a set of five horizontal lines around which the score is written. Certain symbols, such as notes or signatures are located inside it, while others are placed above or below it.

A system is a set of staves joined by a bracket ("["). It represents that all those staves have to be played at the same time. Each one of the staves included in a system correspond to a different instrument. It is used in orchestra scores, for instance.

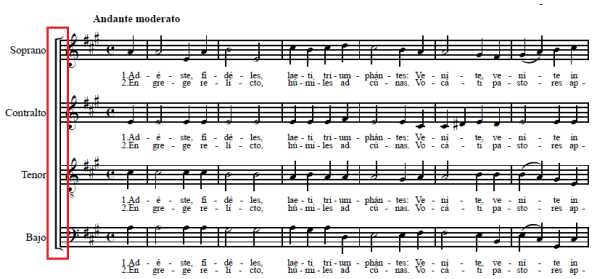


Figure A.1: Choir score with brackets

For polyphonic instruments, that have more than one stave assigned to them, the way of represent it is with braces ("{"). Some instruments that use this notation are piano and marimba, for example.

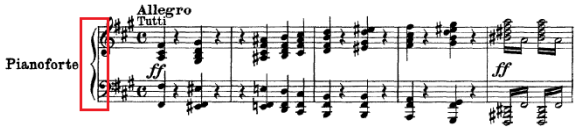


Figure A.2: Piano score with braces

A.2.2. Time signature

A time signature is represented as a mathematical fraction at the beginning of a score, and set the number of notes of each type that can fit into the same measure. It is written before any note and after the key signature, if any.

The numerator indicates the number of notes of the type set by the denominator that fit into a measure. The denominator number is mapped to a certain type of note (duration) as follows:

- 2 corresponds to a half note. For example 3/2 corresponds to three half notes per measure. The 2/2 time signature has a especial name and symbol, it is called a cut time or cut-common.
- 4 corresponds to a quarter note. With this denominator there is also a time signature with a special name, the 4/4, called common time.
- 8 corresponds to a 8^{th} note, and so on.

Figure A.3: Cut time

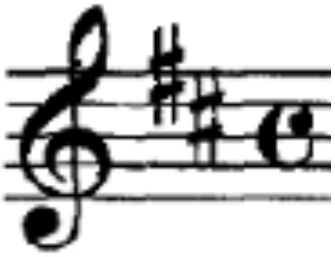


Figure A.4: Common time

Figure A.5: Six-eight time

The time signature choosing changes the character of the music, e.g. a 3/4 is used for waltzes while the 2/4 is used for military marches.

The time signature can change in the midst of a score, usually (but not always) after the occurrence of a double bar.

Measure

The measure is the basic unit of rhythm, equivalent to the number of beats indicated by the time signature. If a measure is incomplete at the beginning of a score or phrase, it is called anacrusis.

A.2.3. Key signature

The key signature is written a the beginning of a score, between the clef and the time signature. It represents the tonality of a score and indicates which pitch must be played higher or lower than its default sound.

The key signature must be composed by the same kind of symbols, in a fixed order. The sharps order is: F, C, G, D, A, E, B; and the flats order is the opposite one: B, E, A, D, G, C, F.

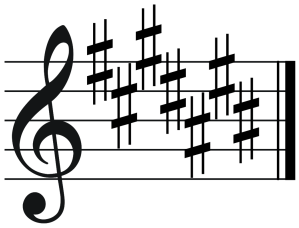


Figure A.6: Order of sharps

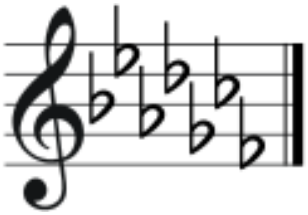


Figure A.7: Order of flats

The key signature can also change along a score, but always after a bar. If the composers want to change a pitch sound during a measure, they use accidentals.

A.2.4. Clef

A clef is a music symbol written at the beginning of each stave that indicates the pitch of a note. It is used to avoid ledger lines for instruments that use higher of lower pitches. There are three symbols (G-clef, F-clef and C-clef) that can be used as different clefs depending on their situation on the stave. Nowadays, there are only four common-used clefts:

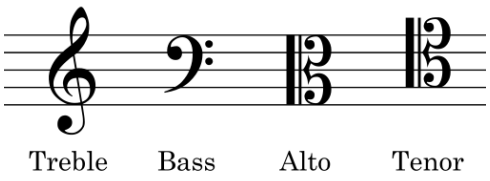


Figure A.8: Used clefs

A.3. Melodic line

The symbols that conforms the melodic line of a piece of music, the sounds, or the lack of them are notes and rest.

A.3.1. Notes

Notes conforms the sound in a score. They appear inside measures and have two features, its pitch and its duration.

The pitch can be described physically as the frequency of the sound. Today a A note situated in the second space using the G-clef must be at a frequency of 440-442 Hz. In the score, the pitch is decided by the position of the head in the stave and the clef.

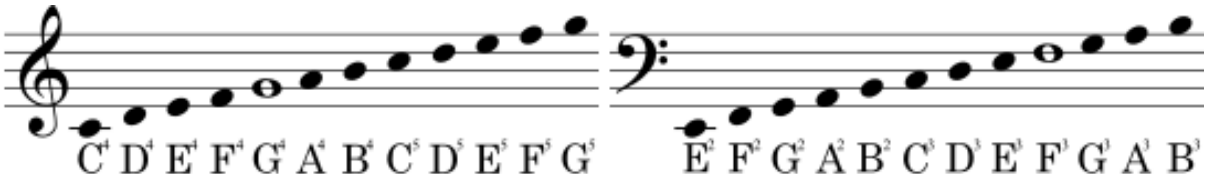


Figure A.9: Pitch depending clef

The duration is relative to the tempo and the time signature. The different duration are name after the fraction of a beat that last: whole, half, quarter, 8^{th}, 16^{th}, etc.

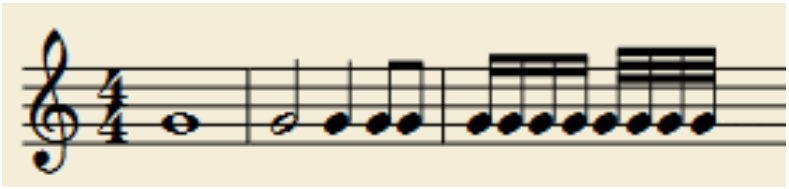


Figure A.10: Note duration

The rhythms so called complex during the report are really called tuplets or irrational rhythms. They are groups of nots that have a different value that their supposed one. An example of this is a triplet, that is three notes with the same duration that have to fit into a the duration that would have two of them. The notes affected by this special rhythm are grouped with a bracket and signed by a number.

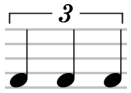


Figure A.11: Triplet example

A.3.2. Rests

The rests are similar except that they don not have a pitch, since they are sound absences.

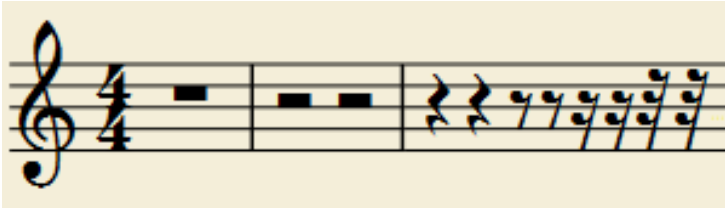


Figure A.12: Rest duration

A.3.3. Modifications

Both notes and rest features can be modified. In case of notes, it can be altered the pitch and the duration whether for rests, the only variation possible is the duration.

Dots

Dots are duration modifications, which can affect notes and rests alike. They add the equivalent value of half the previous value. For example, if there is a half note, and a dot is added, the final value of the note is three beats, two from the half note plus 1/2 x 2 = 1 from the dot.

The dots can appear chained. In this same example, if the first dot had a value of 1 beat, the second one have a value of 1/2 x 1 = 0.5

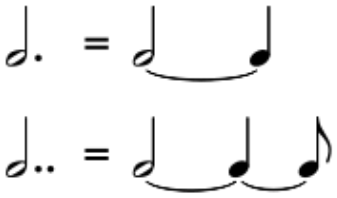


Figure A.13: Dots equivalence

Accidentals

Accidentals are pitch modifications, which can affect notes but not rests. There are three main types of accidentals. Sharps rise the pitch half a tone, flats lower the pitch half a tone and naturals erase any pitch alteration, including the key signature.

Figure A.14: Dots equivalence

Its effects applies to any note with the same pitch within the measure unless cancelled by another accidental.

A.4. Annotations

Although the explained symbols are enough to get sounds, the real music need annotations that tell the musician how to play those sounds. Here are a brief description of some of them.

A.4.1. Tempo

Tempo are Italian words that suggest the speed of the score. It can vary within the same piece of music and it is usually placed above the stave. It can also appear as a equality between a duration and a number.

Some examples of Italian words with its equivalent in beats per minute (BPM) are:

- Larghissimo: 24 BPM and under
- Largo: 40-60 BPM
- Adagio: 66-76 BPM
- Andante: 76-108 BPM
- Allegro: 120-168 BPM
- Vivace: 168-176 BPM
- Prestissimo: 200 BPM and over

A.4.2. Ornaments

As the own name indicates, these annotations are not part of the main melodic line, but exist to decorate and make the melodic line more interesting.

There are a number of these symbols that are a shorter way of writing a fixed rhythmic-melodic scheme.

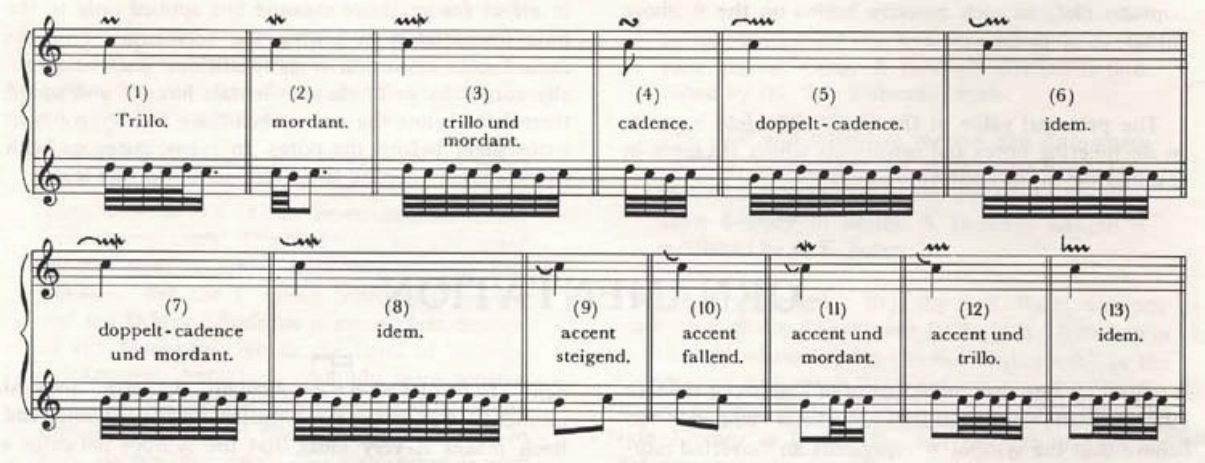


Figure A.15: Ornaments

A.4.3. Dynamics

Dynamics are symbols or Italian words that refer to the loudness of the sound. There are diverse kinds of dynamics, the absolute value of the sound, gradual changes, and attacks on a note. They appear below the stave or below the note they want to modify.

The absolute value tells how loud a note must be play. The main annotations are pianissimo (pp), piano (p), mezzo-piano (mp), mezzo-forte (mf), forte (f) and fortissimo (ff), from softer to louder.

The gradual changes can be from softer to louder, called crescendo (cres.) and represented by a "<" or the other way around, called diminuendo (dim.) and represented by a ">".

Finally, there are symbols that describe how to attack a note, e.g. sforzando (sfz).

APPENDIX B

COMPARATIVE OUTPUTS

B.1. Introduction

The aim of this annex is the discharge of images from the body of the report, to help the reading and provide a secluded section in which the reader can found the content preciously referred.

B.2. State of the art - Software output

This section is destined to those symbols that parametrize the score and its possible variations.

B.2.1. PhotoScore

1. Simple score
   The example shows the digitization of a simple score, the minimum that the application should read, in the view of the author. The chosen score is a musical arrangement of an original music score of the 2004 film *The Chorus* called *Caresse sur l'Ocean*.

   The outcome, as it can be appreciated in the figure B.2, is almost perfect. It also recognizes many of the pencil annotations. Nevertheless, these are precisely the origin of the few flaws that can be seen.

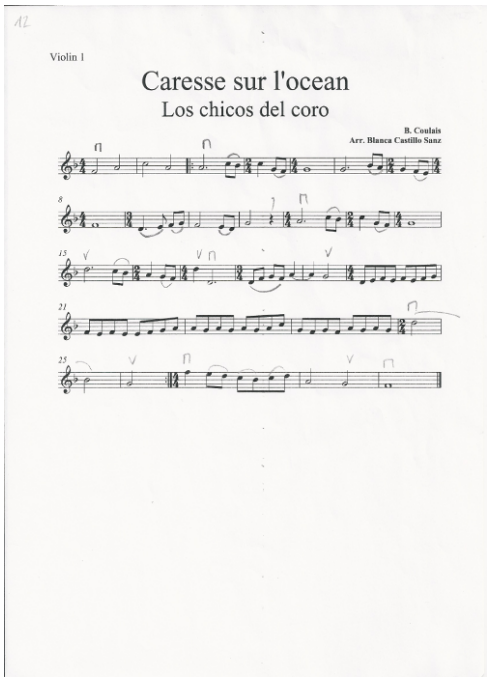


Figure B.1: Input Caresse sur l'Ocean

Figure B.2: Output Caresse sur l'Ocean - PhotoScore

2. Complex score

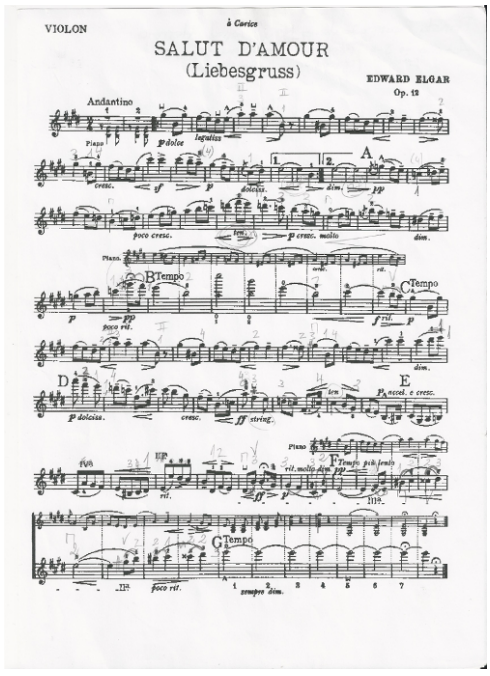


Figure B.3: Input Salut d'Amour

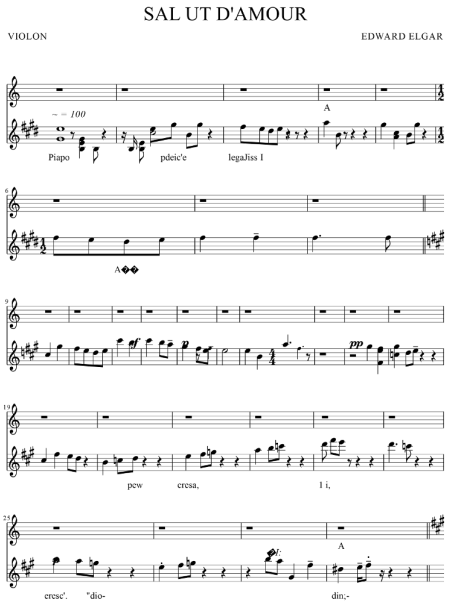


Figure B.4: Output Salut d'Amour - PhotoScore

The second example is a scanned version of *Salut d'Amour* by Edward Elgar. This score is

more complex not only because of its extension but for the types of notations that includes: piano defects in a violin particella, ornaments, and strings notations, for instance.

The software detects multiple errors and warns if the printed score option is wrong and needed the handwritten one. Nevertheless, despite the flaws found, the program does digitize.

The recognition accuracy is deficient in general. Nevertheless, there are some passages in which the melodic line can be followed [see figure B.4].

B.2.2. SharpEye

1. Simple score
   The recognition is perfect. As PhotoScore, it recognizes the pencil symbols. Furthermore, it does not introduce foreign elements. This can be due to the lack of letters in the digitization with this software (title, composer...) [see figure B.6].

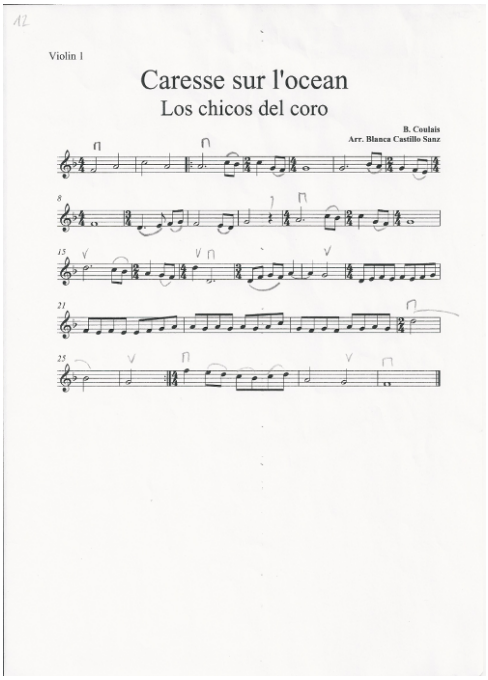


Figure B.5: Input Caresse sur l'Ocean

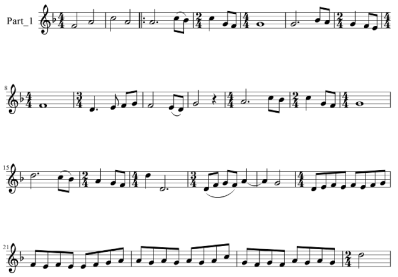


Figure B.6: Output Caresse sur l'Ocean - SharpEye

2. Complex score
   The errors made in this score are quite similar to the ones made by PhotoScore. The overall output is unsatisfactory [see figure B.8] and even worse than with PhotoScore.

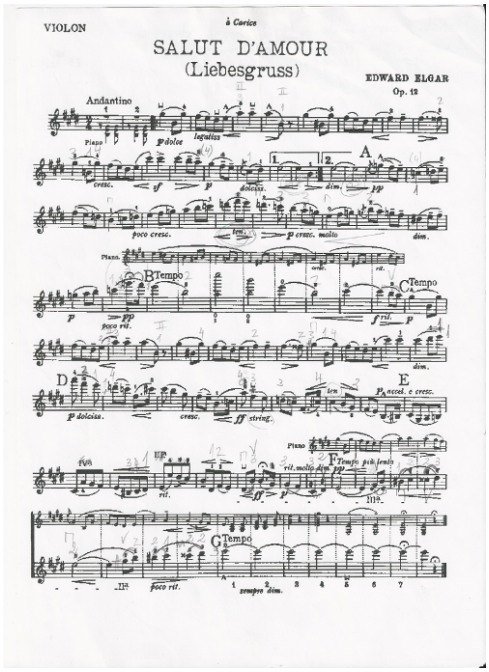


Figure B.7: Input Salut d'Amour

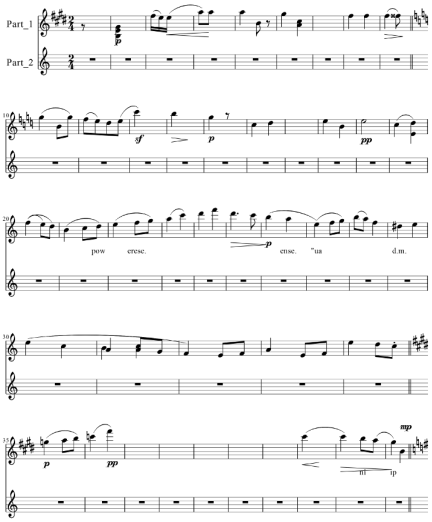


Figure B.8: Output Salut d'Amour - SharpEye

B.2.3. Audiveris

1. Proposed sample

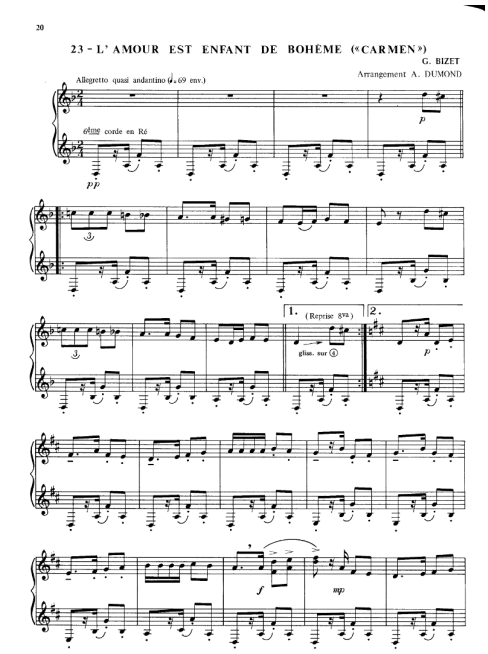


Figure B.9: Input Carmen

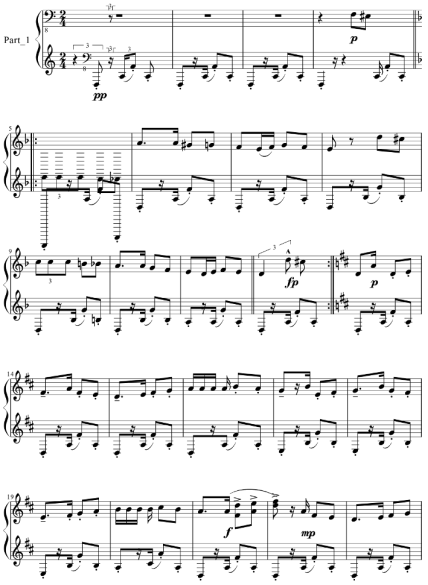


Figure B.10: Output Carmen

The score of the opera *Carmen* by Bizet, *L'amour est enfant de bohème*, was included in a collection of samples in the software distribution. However, the results are not completely satisfactory, as can be seen from figures B.9 and B.10.

The errors appear mainly in the first two staves. The ones in the following staves are related with annotations, glissandi and 8^{va}, for instance.

2. Simple score
   The pitch digitization is acceptable but not entirely satisfactory [see figure B.12]. The time signature does not match and there are multiple overlapping symbols, as well as modifiers that do not appear in the original score.

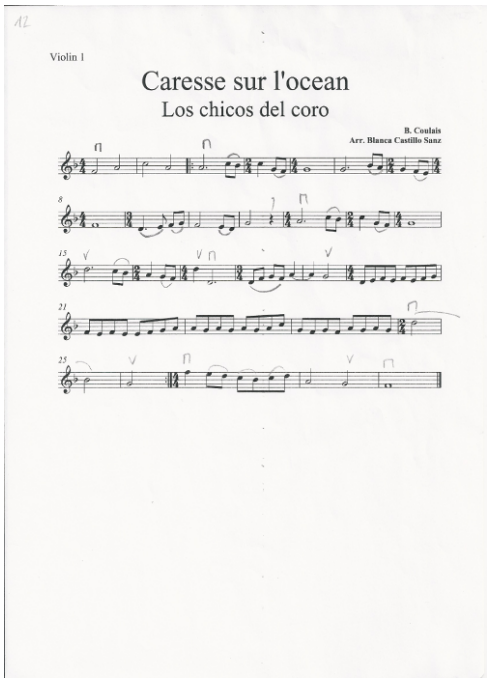


Figure B.11: Input Caresse sur l'Ocean

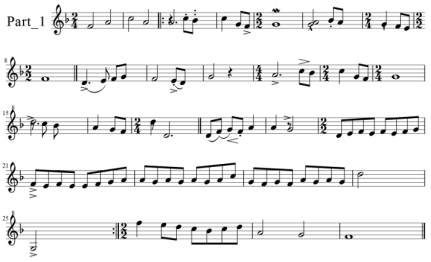


Figure B.12: Output Caresse sur l'Ocean - Audiveris

B.2.4. OpenOMR

1. Simple score
   Several mistakes can be appreciated in the analyzed score [see figure B.14]. An extra stave is recognized, and the program misidentify the time signature for note heads. It also seems to have problems recognizing white headed notes.

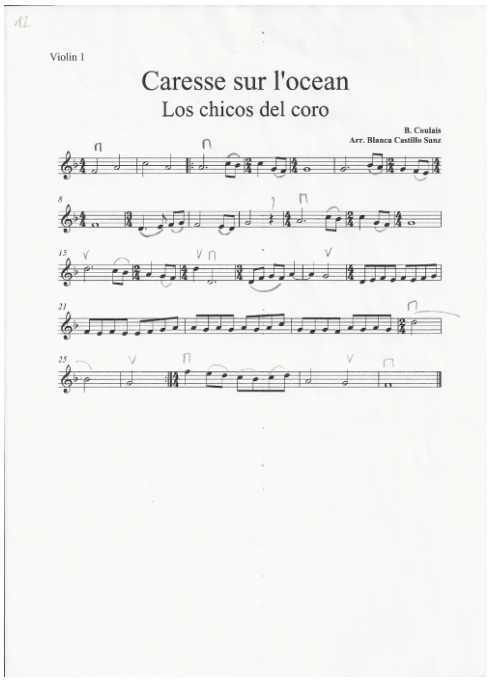


Figure B.13: Input Caresse sur l'Ocean

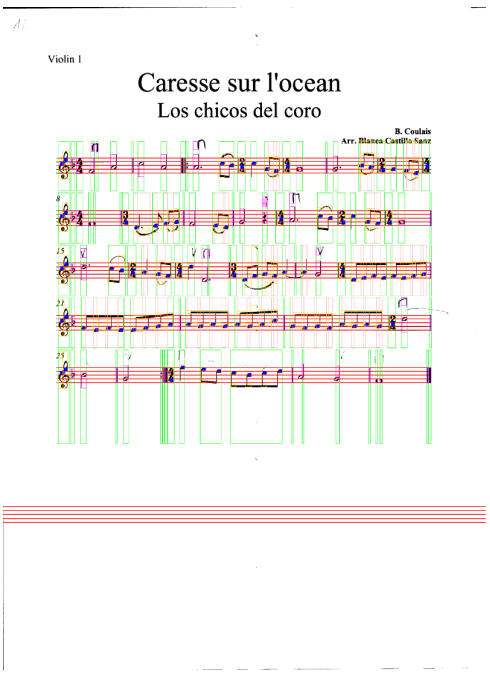


Figure B.14: Output Caresse sur l'Ocean - OpenOMR

APPENDIX C

SURVEYS

C.1. Introduction

The importance of retrieving data as both requirements and feedback is crucial for developing a software that will be use for musicians exclusively. This fact force the application to be extremely easy to learn and use if the program wants to be useful.

In the present situation, the recovery of information is difficult because of the limited public for the software. Therefore, the followed strategy it has been the design of surveys and their distribution among conservatory students and teachers, and university students and professors of music.

In the next sections the reader can find the questions proposed for each survey and it results. The images represent the answers at the moment of the writing of this report, but the link contains the real time outcomes. The open answers question are not included in the figures to avoid its extension, but can be consulted through the provided link.

C.2. Requirements survey

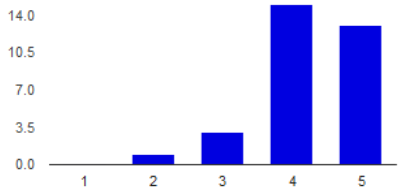
This form was released on July 28 as a first approach of potential users, their interest and their needs.

The results show that there are people interested in the project that use music editors but nearly half of them does not use any OMR system. Their preferred platform is a computer and they are quite demanding with the functionality.

Most of the surveyed are indifferent to pattern choosing and they do not really understand the input/output idea, since they asked for both PDF format.

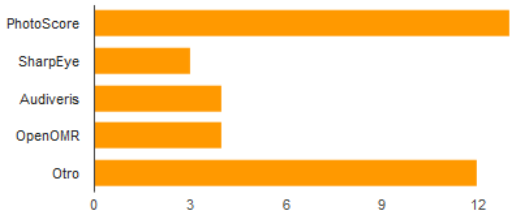
The complete results are accessible on: `goo.gl/gxl91e`

¿Cuán útil crees que puede ser un programa de estas características?



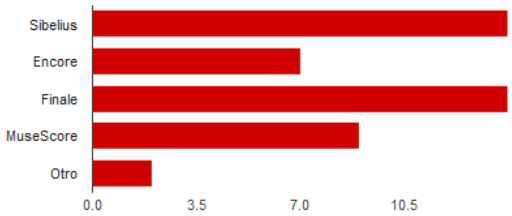
Nada: 1	**0**	0%
2	**1**	3.1%
3	**3**	9.4%
4	**15**	46.9%
Mucho: 5	**13**	40.6%

¿Conoces o utilizas alguno de los siguientes programas?



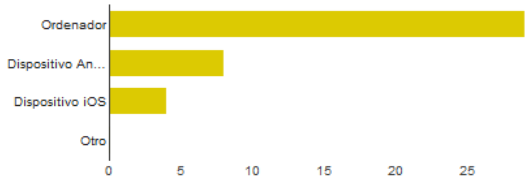
PhotoScore	**13**	40.6%
SharpEye	**3**	9.4%
Audiveris	**4**	12.5%
OpenOMR	**4**	12.5%
Otro	**12**	37.5%

¿Utilizas alguno de los siguientes programas?



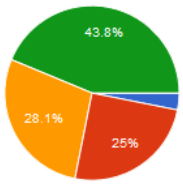
Sibelius	**14**	43.8%
Encore	**7**	21.9%
Finale	**14**	43.8%
MuseScore	**9**	28.1%
Otro	**2**	6.3%

¿En qué plataforma te sería más útil la aplicación?



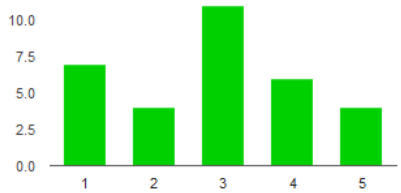
Ordenador	**29**	90.6%
Dispositivo Android	**8**	25%
Dispositivo iOS	**4**	12.5%
Otro	**0**	0%

¿Qué mínima funcionalidad aceptarías para un programa de este tipo?



Notas	**1**	3.1%
Notas y ritmo	**8**	25%
Notas, ritmo y matices	**9**	28.1%
La partitura exacta	**14**	43.8%
Otro	**0**	0%

¿Te molestaría que la aplicación te hiciera señalar patrones antes de la digitalización?



Nada: 1	**7**	21.9%
2	**4**	12.5%
3	**11**	34.4%
4	**6**	18.8%
Mucho: 5	**4**	12.5%

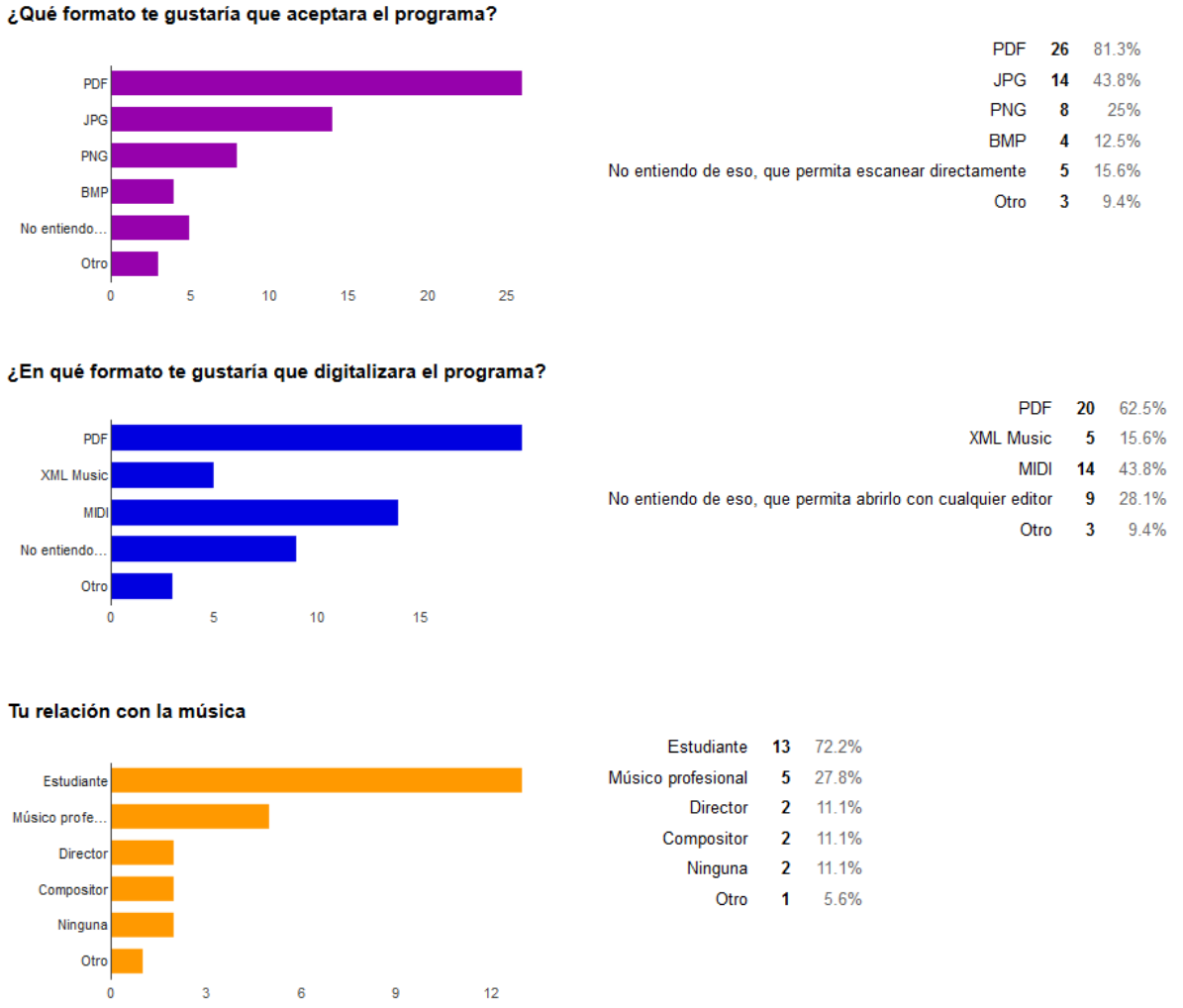


Figure C.1: Presurvey

C.3. Software survey

This form was released on April 20 as a way of receiving feedback from the workshops attendant.

Even though the quantity of results is low, the main idea is clear. The participants are users of editors but not all of them are familiar with OMR systems. They are interested in the idea and understood the functioning of the logic of the software.

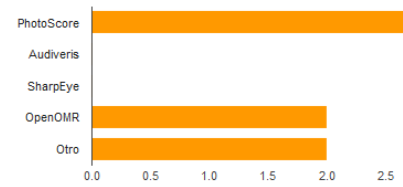
Regarding the application itself, they think it is user-friendly and depending on their future objective, found the pattern choosing annoying or not.

The complete results are accessible on: `goo.gl/HxJs4K`

¿A qué taller has asistido?

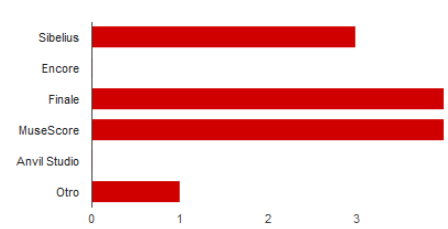
UAM - 22/04/2015	5	71.4%
Conservatorio Arturo Soria - 27/04/2015	2	28.6%
Ninguno	0	0%

¿Conoces o utilizas alguno de los siguientes programas?



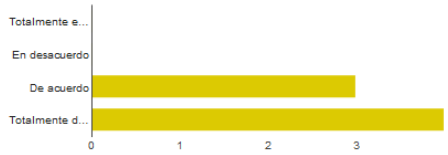
PhotoScore	3	42.9%
Audiveris	0	0%
SharpEye	0	0%
OpenOMR	2	28.6%
Otro	2	28.6%

¿Utilizas alguno de los siguientes programas?



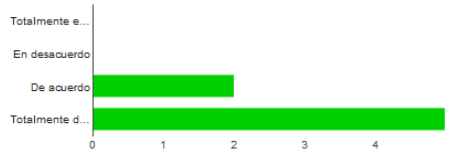
Sibelius	3	42.9%
Encore	0	0%
Finale	4	57.1%
MuseScore	4	57.1%
Anvil Studio	0	0%
Otro	1	14.3%

Las posibilidades de la herramienta me parecen atractivas [A continuación se te preguntará acerca de la parte teórica del taller]



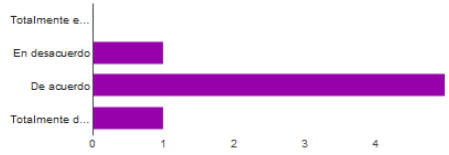
Totalmente en desacuerdo	0	0%
En desacuerdo	0	0%
De acuerdo	3	42.9%
Totalmente de acuerdo	4	57.1%

He entendido el proceso de reconocimiento llevado a cabo por la aplicación [A continuación se te preguntará acerca de la parte teórica del taller]



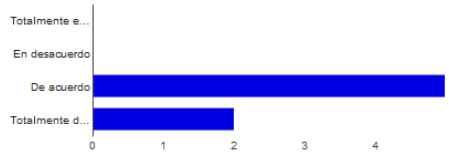
Totalmente en desacuerdo	0	0%
En desacuerdo	0	0%
De acuerdo	2	28.6%
Totalmente de acuerdo	5	71.4%

Los ejemplos son representativos del alcance del software [A continuación se te preguntará acerca de la parte teórica del taller]



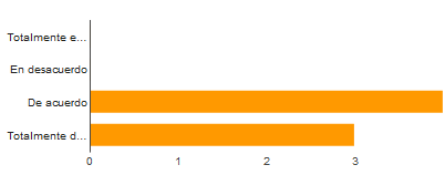
Totalmente en desacuerdo	0	0%
En desacuerdo	1	14.3%
De acuerdo	5	71.4%
Totalmente de acuerdo	1	14.3%

La aplicación es sencilla de instalar [A continuación se te preguntará acerca de la parte práctica del taller]



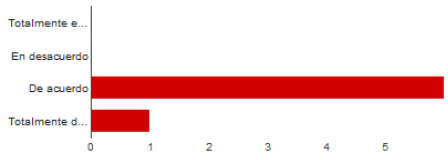
Totalmente en desacuerdo	0	0%
En desacuerdo	0	0%
De acuerdo	5	71.4%
Totalmente de acuerdo	2	28.6%

La aplicación es sencilla de ejecutar [A continuación se te preguntará acerca de la parte práctica del taller]



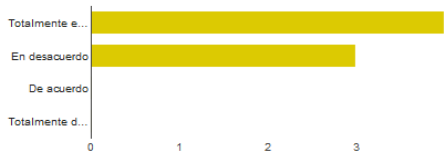
Totalmente en desacuerdo	0	0%
En desacuerdo	0	0%
De acuerdo	4	57.1%
Totalmente de acuerdo	3	42.9%

La aplicación es sencilla de usar [A continuación se te preguntará acerca de la parte práctica del taller]



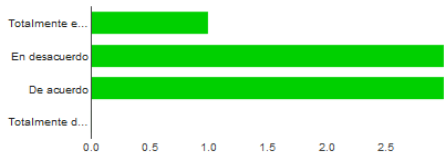
Totalmente en desacuerdo	0	0%
En desacuerdo	0	0%
De acuerdo	6	85.7%
Totalmente de acuerdo	1	14.3%

El idioma me dificulta su uso [A continuación se te preguntará acerca de la parte práctica del taller]



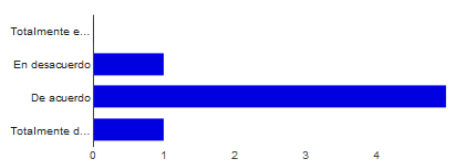
Totalmente en desacuerdo	4	57.1%
En desacuerdo	3	42.9%
De acuerdo	0	0%
Totalmente de acuerdo	0	0%

Me resulta inconveniente introducir parámetros [A continuación se te preguntará acerca de la parte práctica del taller]



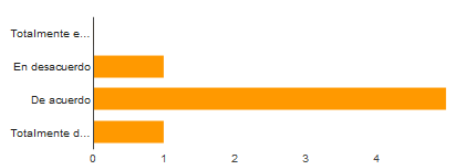
Totalmente en desacuerdo	1	14.3%
En desacuerdo	3	42.9%
De acuerdo	3	42.9%
Totalmente de acuerdo	0	0%

La aplicación cubre los objetivos planteados [A continuación se te preguntará acerca de la parte práctica del taller]



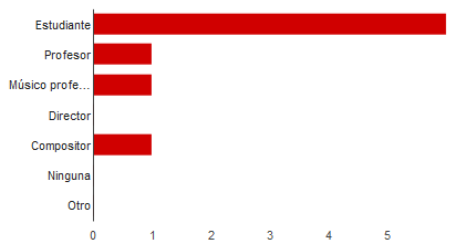
Totalmente en desacuerdo	0	0%
En desacuerdo	1	14.3%
De acuerdo	5	71.4%
Totalmente de acuerdo	1	14.3%

Usaré la aplicación el en futuro [A continuación se te preguntará acerca de la parte práctica del taller]



Totalmente en desacuerdo	0	0%
En desacuerdo	1	14.3%
De acuerdo	5	71.4%
Totalmente de acuerdo	1	14.3%

Tu relación con la música



Estudiante	6	85.7%
Profesor	1	14.3%
Músico profesional	1	14.3%
Director	0	0%
Compositor	1	14.3%
Ninguna	0	0%
Otro	0	0%

Figure C.2: Postsurvey

APPENDIX D

USERS MANUAL

D.1. Download

The software will be free of charge, and findable for the moment at a Dropbox account, so you can download it by clicking any of the links below.

The link will redirect you to a Dropbox download page in which you will have to choose between download it to your computer or save it into you Dropbox account. In around a minute, you will own your copy of the software.

Linux/Ubuntu

The Linux/Ubuntu version of the program is reachable from:
`https://www.dropbox.com/s/i9v8t4ivyhby9fv/MScrivener_Linux.zip?dl=0`

Windows

The Windows version of the program is reachable from:
`https://www.dropbox.com/s/9zvvmwil9a4sivs/MScrivener_Windows.rar?dl=0`

D.2. Installation

Once you have downloaded the compressed file, you will have to decompress it in a directory of your choice. The program is portable, so you do not have to do anything else: The software is ready to use.

To run the program, follow the steps detailed for you distribution.

Linux/Ubuntu

Open a terminal and access the directory in which you have unzip the compressed file. Reproduce the following commands:

```
cd MScrivener
./MScrivener
```

Windows

Open the directory in which you have decompressed the file and open the directory MScrivener inside it. To run the program, you only have to double click on the file MScrivener.exe.

D.3. Getting started

The program usage is the same for both operating systems. The recognition process includes selecting an image, inserting several parameters and choosing some patterns.

Input/output

The first step you have to take is select the score you want to digitize. The program accepts as input any image format, being the most common JPG, PNG and BMP. If you have a PDF you want to recognize, you can convert it using services as `http://pdf2jpg.net/`, for instance. Notice that you must choose the higher quality in order to obtain the best result.

To open an image, you have to click on the menu: File => Open...

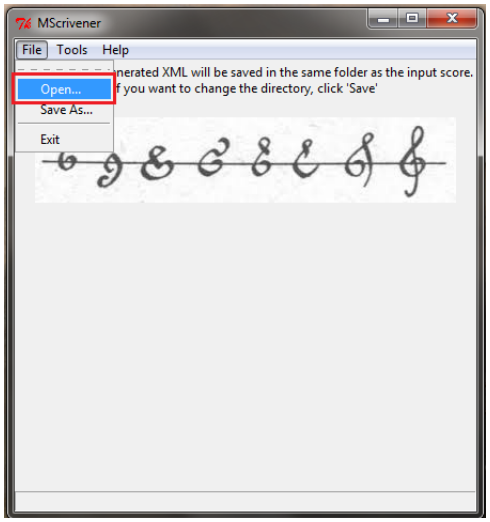


Figure D.1: Open a image

Then, navigate your computer and choose the image you want to recognize. Once you open it, it will be display automatically. The size of the image depends on the resolution and it can not be resized.

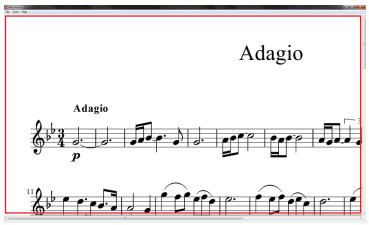


Figure D.2: Score display

By default, the output file will be save with the same name and in the same folder as the input file, but if you desire to change the location or file name, click on the menu: File => Save as...

After you have your score loaded into the program, you only have to run the recognizer. To do this, click on the menu: Tools => Start recognition.

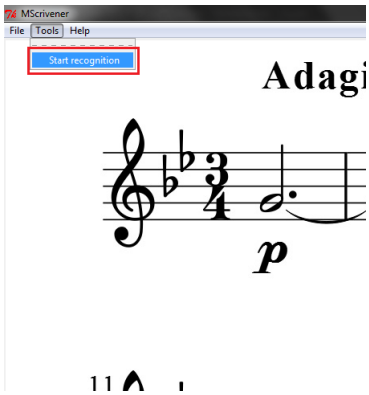


Figure D.3: Start the recognition

Parameters

The program will open a wizard that will help you during the recognition process. The first parameters you have to set are the time signature and the key signature. Both of them work with spinners so to fix the numbers, click the arrows to increase or decrease the amount inside the box.

The time signature value is direct, as you can see in the image below.

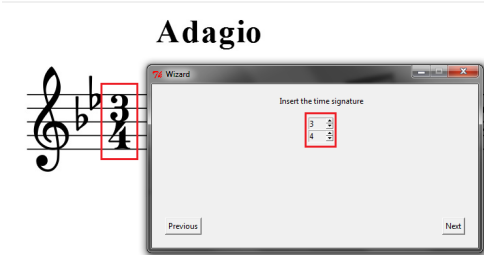


Figure D.4: Set the time signature

For the key signature though, you have to keep in mind that sharps are positive values and flats are negative values. That way, if you want to set the key signature for A major, you will have to choose +1, whether if you seek c minor, you will have to establish the parameter to -3.

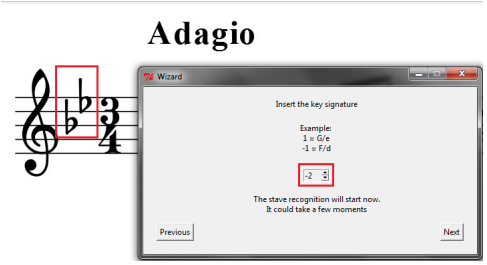


Figure D.5: Set the key signature

Immediately after setting these parameters, the program will recognize the staves in the score, which can take some time. Then you will be asked to select some patterns.

If the number of staves it is not what expected, the software will not be able to digitize your image. You can try to obtain a better quality image out of your score or wait until a new version of the program is release.

Patterns

The wizard will guide you now to the head pattern selection. It will ask you if there are whole, half and shorter notes in your score. In each case, if so, you will be allowed to choose the head pattern.

Let us see a practical example: If we are being asked about black heads (quarter notes or shorter value) and our score has one or more of this type of heads, click on the Yes button. The wizard will disappear and we will have to click in the center of the head, like this:

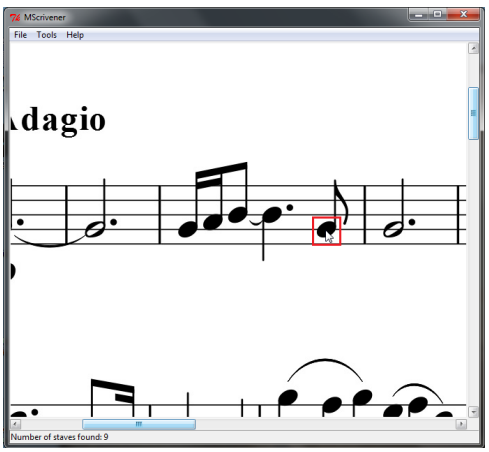


Figure D.6: Choose head pattern

The program will try to auto-adjust the area around the head. In any case, it will show you the area elected in case it is not what you wanted.

Figure D.7: Re-choose pattern

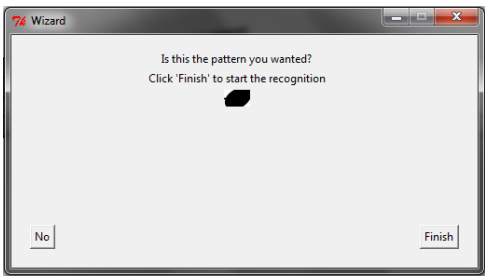


Figure D.8: Accept patter

The accepted pattern will have to be as close to the head as possible, without interfering with any other symbols. The pattern selection is really important, it will determine the quality of your output.

Having selected all necessary patterns for your score, you will click on the Finish button and wait for the recognition to end.

Recognition

The status area will inform you of the recognition process. The program analyze stave by stave the score, so you will know at any time the recognition state.

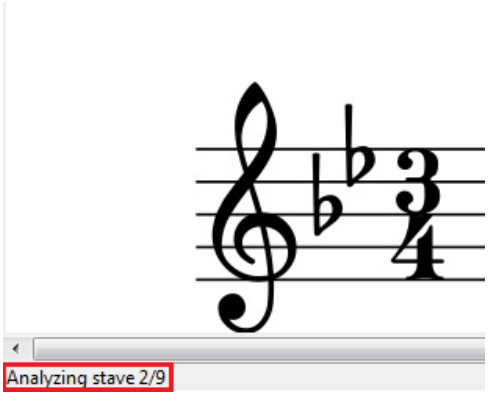


Figure D.9: Choose head pattern

The information sequence will be as followed:

1. Analyzing stave 1/X
2. Analyzing stave X/X
3. Writing XML
4. Done!

Once the process is completed, you can search for your digitized score and open it with any music editor, such as MuseScore, Sibelius, Finale, Encore...

D.4. Contact

Please, take a few minutes filling this survey:

`https://docs.google.com/forms/d/1kWY3NNiTbrf3HH5itAPjUbcz6K3oVBvngJ183Y5-QYE/viewform`

It is completely anonymous and it will help the author to get your feedback and fix any bug reported.

If you want to contact the author, do not hesitate and write to:

monica.villanueva@estudiante.uam.es

APPENDIX E

DOWNLOADS

E.1. Introduction

This section is dedicated to centralizing download links of additional material, such as the Application Programming Interface (API) of the developed system, or the application itself.

The links will redirect you to a Dropbox download page in which you will have to choose between download its content to your computer or save it into you Dropbox account.

E.2. API

The code is documented using Doxygen annotations, which helped creating a dynamic API in HTML that can be navigated in a browser in a local way.

This API is accessible on the following link: `https://www.dropbox.com/s/f73wpyf6ws14c86/Doxygen_html.zip?dl=0`

E.3. MScrivener

The software is distributed for two platforms. Choose the one that fits better with your operative system.

Linux/Ubuntu

The Linux/Ubuntu version of the program is reachable from:
`https://www.dropbox.com/s/i9v8t4ivyhby9fv/MScrivener_Linux.zip?dl=0`

Windows

The Windows version of the program is reachable from:
`https://www.dropbox.com/s/9zvvmwil9a4sivs/MScrivener_Windows.rar?dl=0`