Technical University of Cluj-Napoca
Faculty of Automation and Computer Science
Department of Computes Science

# Sheet music reader

Student: **Stefan Ciuprina**, gr. 30434

30/05/2020

## 1. Introduction

The purpose of this project is to design and develop an application capable of reading sheet music, playing sounds based on an input score. The application is implemented in C++ using OpenCV, and requires an input image, of which it will play its notes and rests. By default, it has 5 scores for demonstration purposes, but the input data set may be enlarged.

## 2. State of the art

There are already various implementations to this subject. Most of them rely on identifying each characteristic element of a music score. For example, removing the staff lines using image processing algorithms, identifying each note using a structural element, then putting them all together in an audio file. My approach to this project isn't far from those techniques, being a simpler and faster, but having some drawbacks such as fixed image size input for correctly recognition of elements. In the following section, all the steps for sheet music recognition will be presented.

## 3. Proposed method

To better understand my approach on this project, each step will be exemplified using a simplified version of the classic Jingle Bells song.

First of all, the image is read, converted to grayscale, and then to binary (containing only black and white colors), removing all the tints of gray.

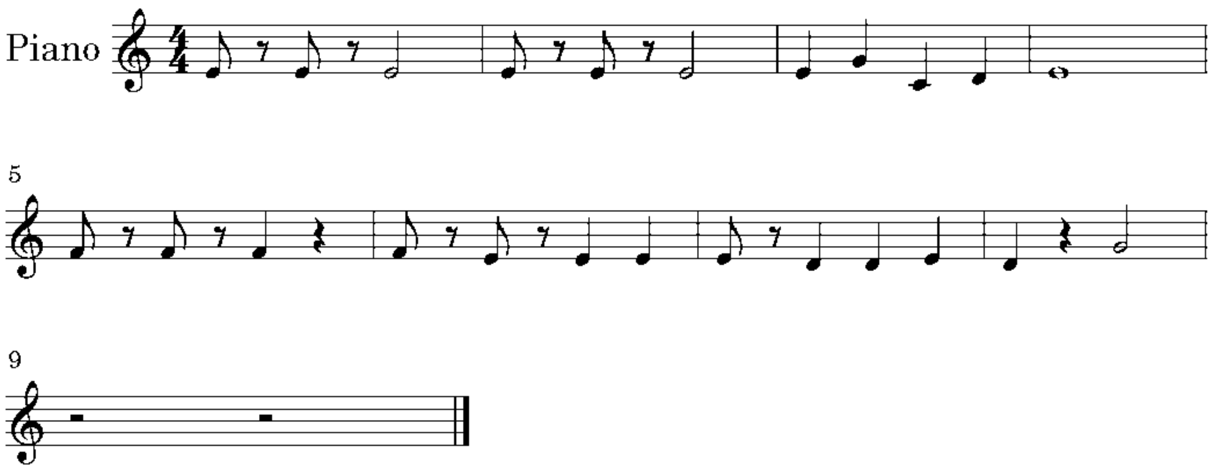


Figure 1: Image converted to binary

For the binary conversion, a threshold of 100 is being used, making all tints of gray less than 100 black, and all the others white.

Next, to the image is applied a horizontal projection, helping us identify where in the image we have notes and staff lines.

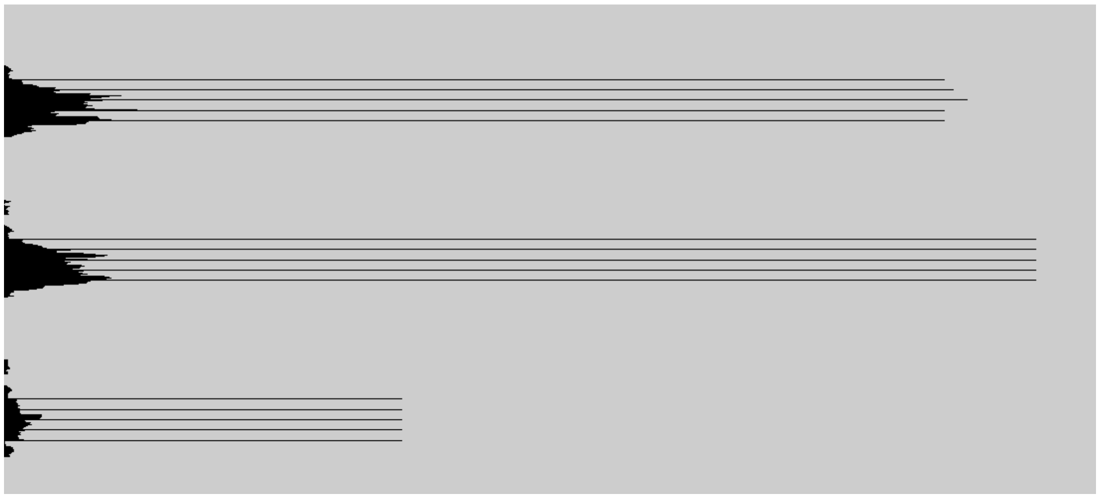


Figure 2: Horizontal projection

Using this horizontal projection, we can easily separate the staves and identify the staff lines. The image is scanned up to down, on the first column (for staves separation) and on the first quarter from the whole projection for the staff lines. The image below exemplifies the algorithm.

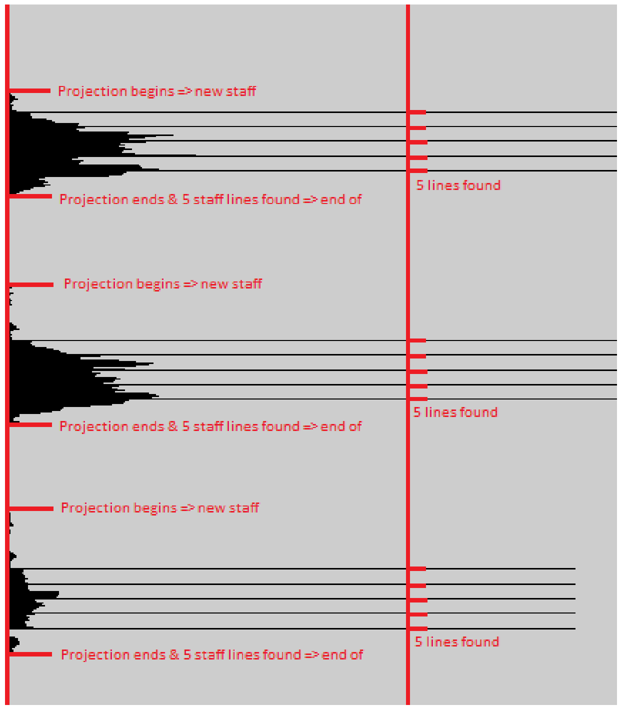

Projection begins => new staff
5 lines found
Projection ends & 5 staff lines found => end of
Projection begins => new staff
5 lines found
Projection ends & 5 staff lines found => end of
Projection begins => new staff
5 lines found
Projection ends & 5 staff lines found => end of

Figure 3: Staff identification

Next, we start scanning the binary image column by column, for each staff. The elements searched for are notes (whole, half, quarter and eighth) and rests (half, quarter and eighth). The image is scanned in blocks (structural elements), searching in each of them for the characteristics for a note or for a rest. The algorithm is presented below:

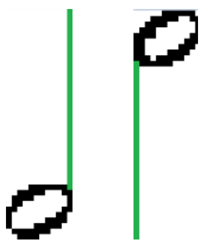
```
For pixel (x, y) in image:

    if(structural element not inside image)
        return INVALID NOTE
    line-position = getLinePosition(x, y)
    if(line not found)
        if(is whole note)
            return WHOLE NOTE
        else return INVALID NOTE
    if line-position = RIGHT => ellipseLocation = DOWN
    if line-position = LEFT => ellipseLocation = UP
    if(hasEllipse(ellipseLocation)
        if(isEllipseFilled)
            if(hasTail)
                return EIGHTH NOTE
            return QUARTER NOTE
        return HALF NOTE
    return INVALID NOTE
```

## *Explanation:*

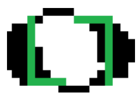
The first check is that the structural element to be searched into is inside the boundaries of the image, since we go in the right-down of the current (x,y) pixel and we might get out of the picture. Then, we search for a *note line*, which is the line which all notes besides the whole note have.

The line can be either in the right or the left of the structural element, since if the notes are higher in the stave, they will have it on the left side.

The images below represent in green what is searched.

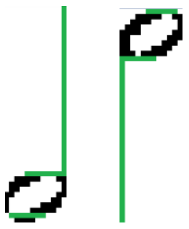


If no line is found, it means there may be a whole note (the only one which doesn't have a line). So it is searched for the lines as in the picture below.

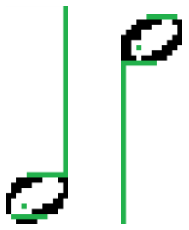


If it’s not a whole not either, then it’s not a note there. It may be a rest though, but that is checked outside of this function.

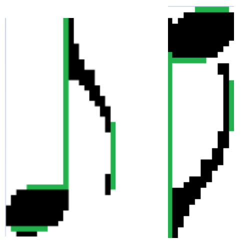
If we do have a line, we check for an ellipse at the bottom of the structural element (if the line is on the right) or at the top (if the line is on the left), with the lines presented as in the pictures below.



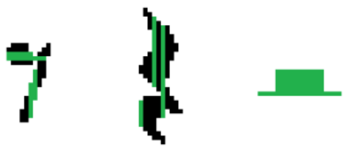
If those lines are found, it means there we certainly have a note here. Now to find its type, we check first for a pixel inside the ellipse. It the pixel is white, it means the ellipse is not filled, and therefore it is a half note.



If the pixel is black though, it means the ellipse is filled. One more check is needed to separate the quarter note from the eighth note, and that is the tail, which is checked with using the line illustrated in the picture below.



If no note is found, there still have to be made scans for rests. The lines searched for each rest are presented below.



Once we have found a note type, we can also determine its location on the stave lines. The pixel found at the center of the ellipse is compared to the staff lines found earlier using the horizontal projection.

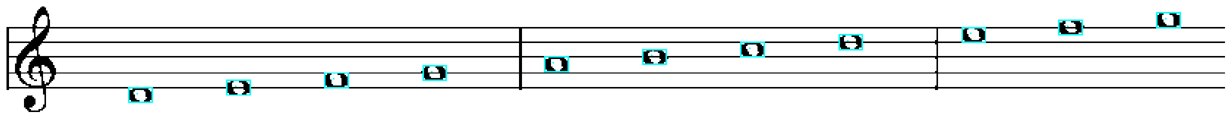
An array is used for saving all the row indexes for the staff lines. For each staff line of a stave, we check, if there's a note point in the nearby of it, or above it. If yes, the note is labeled with its value accordingly.

Lastly, each note is played using *Beep*, and each rest using *Sleep,* from the *windows.h* library. Also, each ellipse and rest found are surrounded in colored boxes, according to their duration.

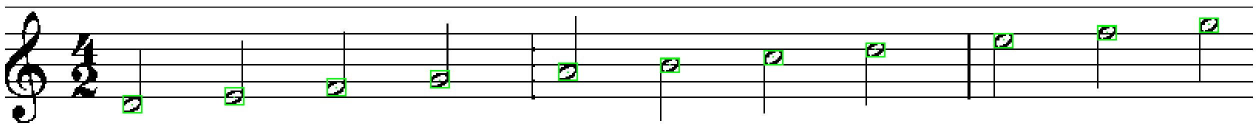
## 4. Experimental results

The workings of the project have been tested with 5 input scores represented as *png* files. The audio sounds are as expected, comparing those result with the website flat.io. The visual output of those images is presented below.

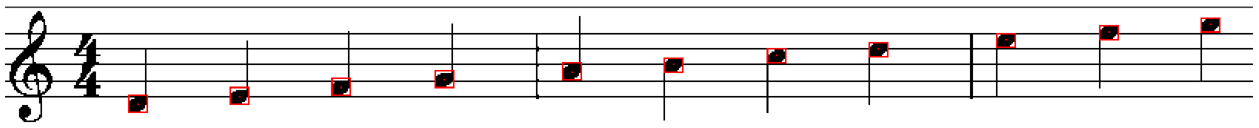
Whole notes:



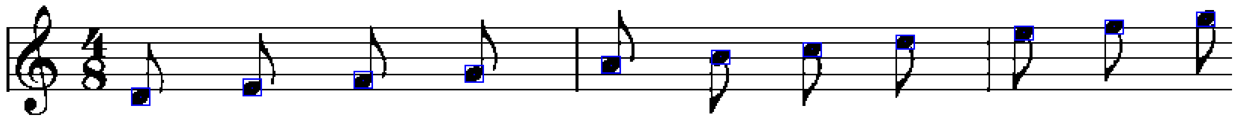
Half notes:



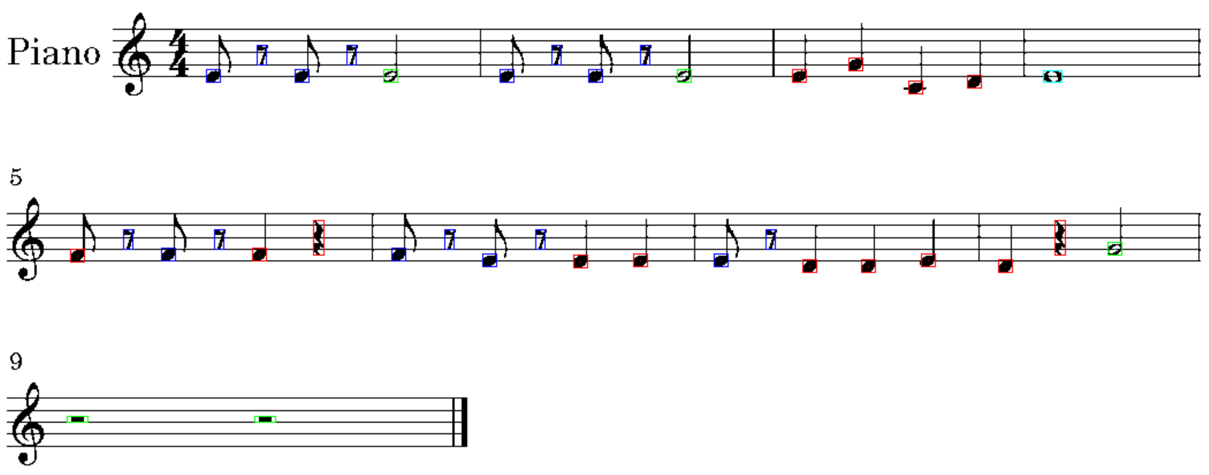
Quarter notes:



Eighth notes:



Jingle bells:



## 5. Conclusions

The goal of the project is reached, having a fully functional sheet music reader. The approach used is a fast one, since it doesn't search for all of the details of the note, but only of a few aspects that define it, but its limitation is that the input images must be of an exact size. It would still work in cases with greater or smaller size pictures, but it would require a resize at the beginning. Other future improvements include identification of the other types of notes (third, sixteenth and so on), identification of other clefs than G Clef (Romanian: cheia sol), support for accidentals, ornaments, dynamics, articulations, a GUI and even a use case for creating your own score. Nevertheless, those were out of scope of this project, the current application being completely functional for what is was designed to be.

## Bibliography

- Sheet music input generation: https://flat.io/
- OpenCV: https://opencv.org/
- Inspiration – cadenCV research paper: http://bit.ly/2lfe8Gv