Optical Music Recognition

Michael Guo, Daozhen Lu, Anirudh Jhina

1. Introduction

Optical character recognition is a popular subject area and its applications are found in many places such as in language translation apps that allows you to take a picture of some text and will translate that text for you. With the wide array of applications that uses optical character recognition, it seems that this area has been mostly solved. Aside from recognizing texts, optical music sheet detection appears to pose quite a challenge still as it hasn't received as much exposure due to the seemingly lack of demand for such applications. The complexity of how music is written and structured posses a great challenge.

In this project we will attempt to tackle the task of recognizing notes from an image of a sheet music and also transform the recognized notes into a midi file which can play back the music. We take this opportunity to apply our knowledge so far obtained in computer vision to explore the intricacies of sheet music and discover limitations as well as solutions to optical music recognition.

2. Background

For this project we initially begun by looking at outside papers regarding the subject, however, we did not gain much in terms of actual implementation other than getting an idea on image pre-processing and so we did not invest further time into reading research papers.

3. Methods

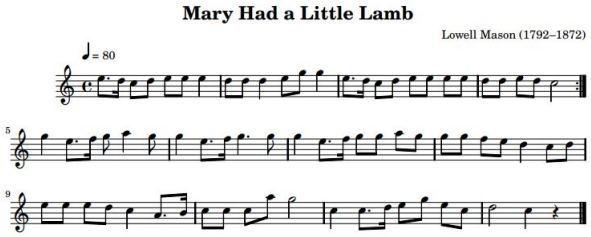


Figure 1: *Original sheet music image.*

3.1 Image Pre-Processing

Before being able to begin the recognition process, the image [figure 1] of the sheet music must be prepared so that information from it will be easier to extract and analyzed with minimal noise. The first step is to invert the image and apply a threshold to gray value pixels which sets their value to either black or white so that the image is binary [Figure 1a].

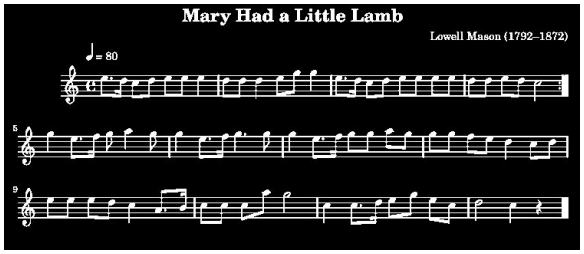


Figure 1a: *Binary version of the original image.*

This helps with reducing noise on the image. Next we apply a morphological operation called erosion on this binary

image using a 3 by 3 box kernel to erode away the horizontal staff lines as well as some noise that are still present [figure 2].

Figure 2: *Eroded image.*

Due to erosion, signals such as notes also become less visible. To remedy this, we apply an operation that is polar opposite called dilation [figure 3] which dilates or enhances the signals on the current image. This newly altered image is ready to be used in later steps for note detection.

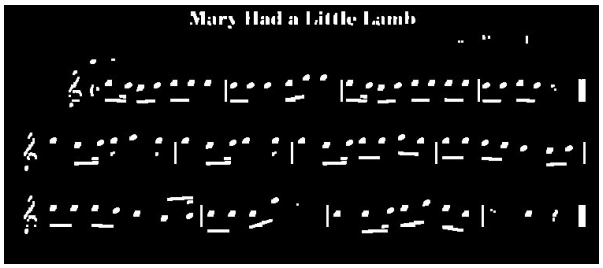


Figure 3: *Erosion followed by dilation.*

3.2 Line detection

Next, we must find the staff lines in the sheet music. This is the most crucial step as the staff lines are the underlying structure of sheet music and dictates the pitch of the notes. To do this, we apply the canny algorithm on the original image to obtain a binary image [figure 4] in which all the edges are clearly isolated. In the next step, we implemented the Hough Transform [figure 5] on the canny edge image.

However, there were many lines that do not belong to the music staff which were picked up.

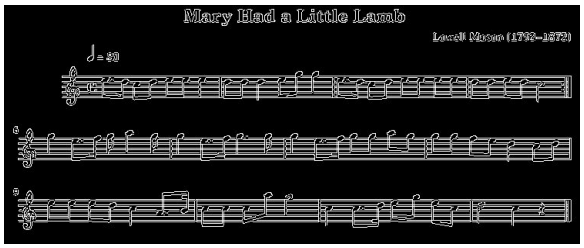


Figure 4: *Canny edge detection.*

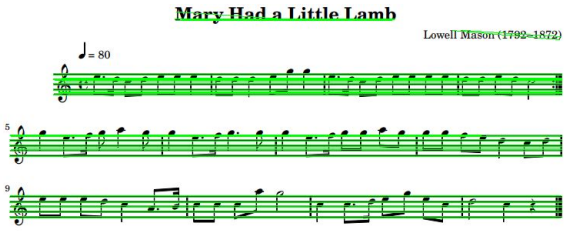


Figure 5: *Hough transform.*

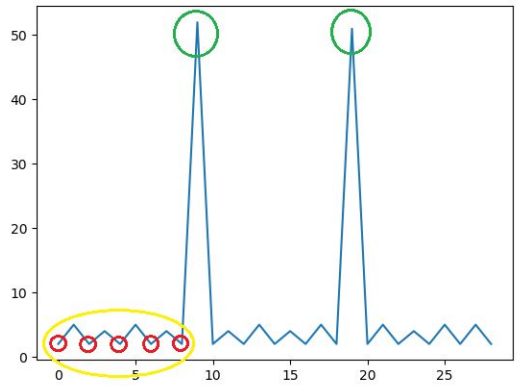


Figure 6: *Graph of row pixel values vs row position index.*

To clear the noisy lines, we plot a graph [figure 6] of the sum of the values of all the pixels in each row against the index of that row by using the original image. In this graph, we can see clear symmetry. Our sheet music contains 3 staffs total. The highest peaks circled in green on the graph indicate the mid distance between individual

staffs. The region circled in yellow indicates a staff. The valleys circled in red indicates individual staff lines and we can see that there are 5 staff lines registered. By using the information from this graph, we can trim out all the noisy lines that are not part of the staff resulting in well defined lines [figure 7].

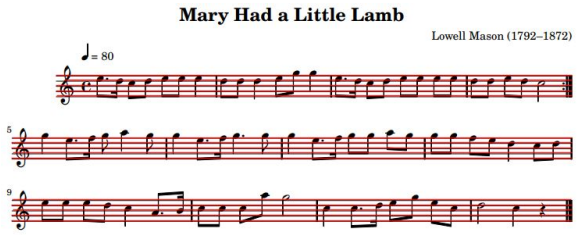


Figure 7: *Denoised Hough lines*

Alternatively, we found a more simple method in detecting the staff lines. To do so, we simply take the canny edge image and scan over each row of pixels as was done for the graph in figure 6, but this time, we add up all the pixel values along each row and apply a threshold. It turns out that the the length of the staff lines make up at least 55% of the width of the image which is marginally greater than other objects on the sheet. So the threshold we apply checks to see if the the sum of the pixels values along a row is at least 55% or so the value of the width of a single row of white pixels. This method produces consistent and accurate detection of the staff lines [figure 8].

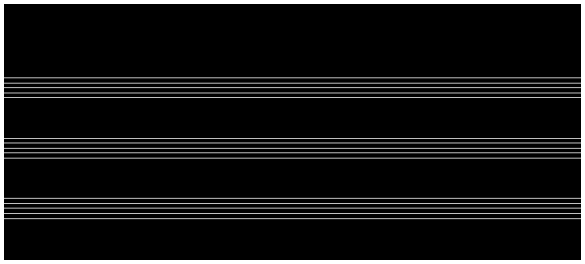


Figure 8: *Alternate line detection results*

3.3 Note Detection

Note detection is the most difficult task of this project. To begin, we define a few key parameters and include some approximations that generalizes well among different sheet musics. We set the note diameter which is the horizontal length of the note as the width of the gap between staff lines multiplied by 1.2. We also define the area of the note as the diameter of the note multiplied by the gap length between staff lines.

Now, we begin the note detection process by iterating through each staff column wise, pixel by pixel, on the binary image and sum up the pixel values of each column that are within the boundaries of each staff in order to detect the tails of notes which allows us to locate where notes are [figure 9] on the staff column wise. Of course, this also picks up other vertical lines that are not note tails such as bar lines.

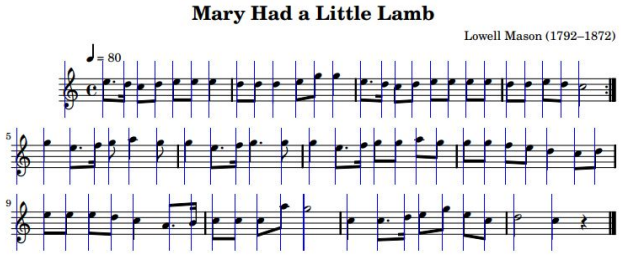


Figure 9: *Blue lines designating where notes are located.*

Next, for each of the possible note locations marked by the blue lines, we perform a vertical or up and down scan by incrementing over each line and gap within a staff. For each increment, a square window of the same area as the note area that we approximated earlier is used to detect whether a note exists at that specific increment by taking the pixel values within

the enclosed square window and applying a threshold. The results are depicted in [figure 10].

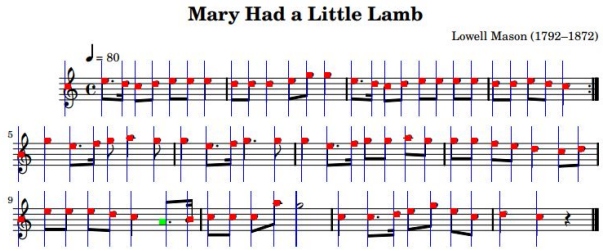


Figure 10: *The red and green squares indicates where notes are detected.*

3.4 Tempo Detection

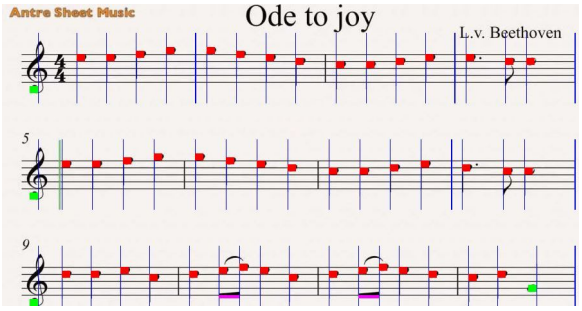
We have two version of the tempo detection.

The first one getNotesAndTmpo_lite() in notereader.py this version can work on any music sheet and can find the 1/8 note by finding if two notes are connected by horizontal bar. But this version is not perfect it have problem detecting the 1/16 and 3/16 notes.

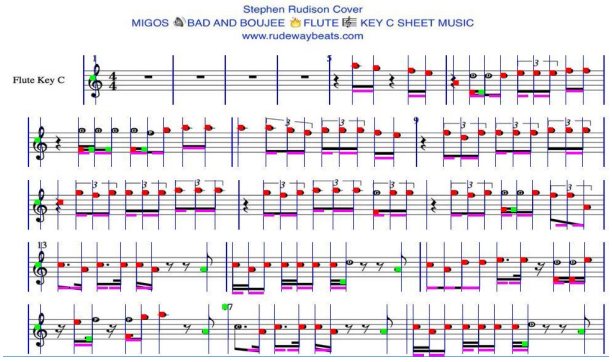
The second version getNotesAndTmpo() in project.py it work perfectly on Mary had a little Lamb song but have some problem with the other songs. It is based on finding the line bar beside each note and detect if they are connection based on whether there is a horizontal bar connected two notes. If yes the length of the note is 1/8. It can also detect if there is a shorter bar between two connected notes to find out if they are 1/16 and 3/16 length by comparing the sum of pixels in the whole area(a square box) near the connection bar with a threshold. However this method failed when it use to scan other music sheet because of different brightness.

Our code separated this two method. Such that one works fine for any music sheet and the other will return a relative better version of mary have a little lamb and the other works fine for any other music sheet.

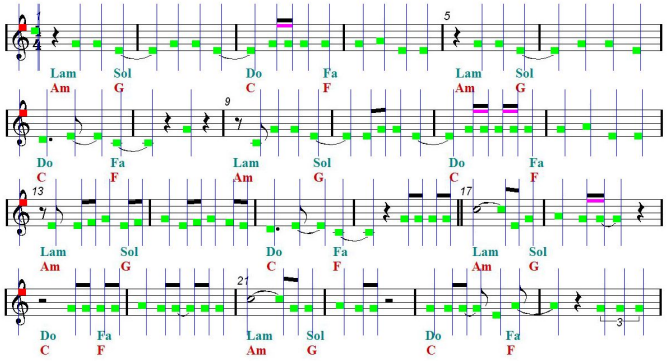
4 Results



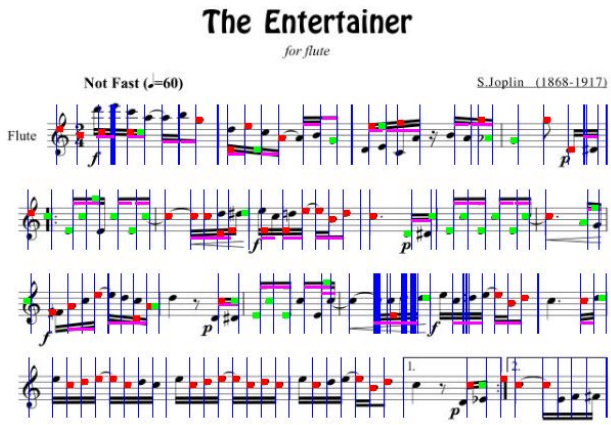
Sample 1



Sample 2



Sample 3



Sample 4

The initial implementation of our detection algorithm was done using the “project.py” file. It was specifically written to work on the “Mary Had A Little Lamb” music sheet in which we were able to achieve a perfect read including being able to detect the sixteenth notes. However, it did not generalize or work on any other sheet music. The final implementation that this report mostly covers is in the “notereader.py” file which generalizes fairly well, but is not able to distinguish 1/8 notes from 1/16 notes.

Our music recognition method works fairly well in terms of detecting where the notes are and the value of the notes as shown in samples 1 through 3. When it comes to detecting the tempo of the notes, it becomes inconsistent as seen in sample 3. Of all the images that we have fed into the algorithm, the treble clef symbol is nearly always picked up. While we can just simply erase the first note detected for each staff, it doesn’t solve the underlying issue which is detecting the notes based of their vertical tails as it is too simple or general of a heuristic. When it comes a symbol such as the whole note which don’t have a tail, it will be missed when we scan through the image. In the case of more complicated sheet music such as the one in sample 4, the algorithm will break down.

5 Possible Improvements

As depicted in the figures and samples, we have false detections which are the result of generalizations in our implementation. A solution that we can explore further on is through the use of classification algorithms in machine learning. For example, at each vertical blue line in which we have detected a note, we can create a bounding box around that detection region and extract the image just within that box and save that image to a folder in which then it can be manually labeled and learned using a neural network. Once the network has been trained we can iterate through each possible note position and use the trained network to predict or classify the image within the bounding box. We have attempted to implement this method by trying to just training a neural network using the “FNN.py” file. The code to generate these mini bounding box images are commented out in the “notereader.py” and we have tried learning the network on the 1/16, 1/8, 1/4, and 1/2 notes with a sample of 7 labeled images for each note. Naturally, we were obtaining nearly random predictions the the network due the the sample size and the number of distinct note labels we are trying to classify. With more time to label more sample images, this could be a promising option.