Optical Music Recognition
CS 194-26 Final Project Report

Andy Zeng
23148163
andyzeng@berkeley.edu

Abstract

Optical Music Recognition (OMR), or alternatively sometimes referred to as Music Optical Character Recognition, is a system for music score recognition. Given a music sheet, usually in the form of an image, the goal of an OMR system is to use various vision algorithms to interpret the corresponding music symbols into digital form, typically playable in the form of a MIDI file. For this project, our objective is to try our hand at designing an OMR system for digital sheet music. Following pre-processing and conventional methods for staff line removal, we propose an observation based approach to musical object location, symbol identification, and melody generation. Our results show that our OMR pipeline works quite nicely for many simple high quality music sheets.

1. Useful Links

Matlab Code: https://bitbucket.org/andyzeng/omr/overview
The code comes with a folder of digital music sheets and a few sample audio files generated from the script.

2. Introduction

With the growing circulation of high quality scanned/printed sheet music across the web, as well as the availability of digital music score creating tools for artists, OMR is becoming increasingly in demand by people like me, who are unfortunately too lazy these days to spend the time and effort to learn new songs and record himself on the keyboard piano. In all seriousness however, OMR has many practical uses in the real world. For example, a digital musician (i.e. electronic/dubstep/techno artist, DJ) can use an OMR system to quickly generate editable audio tracks, in the absence of expensive musical instruments and/or extravagant recording technologies.

Optical Music Recognition is not a trivial task. It is known to be difficult due to the unavailability of robust algorithms for musical symbol location and identification within the presence of superimposed stave lines. Furthermore, due to variations in digital music score editing software and artist preferences, the expected expressions and spatial relationships between musical symbols of interest (notes, marks, dynamics, annotations) are not always consistent. It is also important to note that although the term OMR is closely linked with Optical Character Recognition (OCR), the two systems rarely share universal algorithms for the purpose of recognition (aside from maybe certain data-driven techniques, which have not been explored yet). An important difference is that textual information for OCR is strictly one-dimensional, while musical objects information for OMR is typically two-dimensional (pitch/notehead location vs. time/note length). Furthermore musical objects are typically made of multiple components which can be combined and mixed in many different ways, at the artist's discretion.

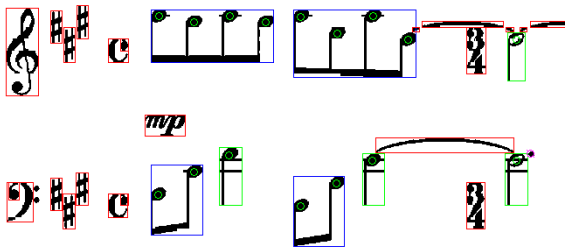


Figure 1: Sample bounding boxes of detected musical objects, categorized into beamed notes (blue), individual notes (green), and other symbols (red).

3. Approach

Following image pre-processing, our core OMR pipeline follows a 4 stage process. In the first stage, we locate and remove staff lines using horizontal projection. We then use this information to divide the sheet music into corresponding stanzas. In the second stage, we locate the musical ob-

jects of interests using flood fill and bounding box analysis. In the third stage, we use a combination of template matching and signature analysis to identify the location of noteheads as well as their associated beat duration. And finally, in the fourth stage, we associate each detected notehead with pitch information, and proceed to reconstruct the melody by inserting them into a time sequence stack. Propagating beat error is reduced using alignment analysis. The melody is then reconstructed into a MIDI file. The entire OMR processing was coded in Matlab.

3.1. Pre-Processing

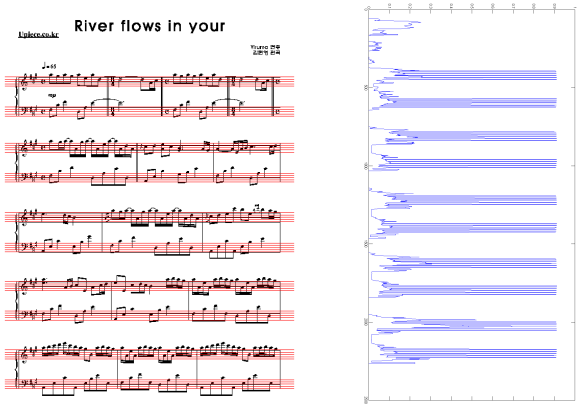
Prior to recognizing musical symbols, we want to reduce as much random noise in the given image as possible. Some digital scores, especially high quality PDF scans of printed scores are quite susceptible to salt and pepper noise. We first resize the music sheet pages to a consistent resolution and convolve the image with a Gaussian filter. We then convert the image into a binary image based on a relatively high threshold. The parameters for the pixels resolution and the Gaussian kernel sizes are chosen empirically to work well with our testing set of music scores.

3.2. Remove Stave Lines

The first core step in our approach is to locate the stave lines. Not only does this help us determine the pitch of the notes, but it also helps us estimate the diameter of the notes (which will be useful later). After locating these lines, we can then proceed to remove them, which can help us locate musical objects of interest.

To locate the stave lines, I used a common method known as Horizontal Projection, as described by Scott Sheridan and Susan E. George. The conceptual idea is relatively straightforward. Given a music sheet image, we sum up the black pixels horizontally along each row. The peaks in the histogram of the projection are the locations of a stave lines. We save these locations to help us determine pitches later during melody generation.

To isolate each stanza segment to divide each page of the sheet music respectively, we first threshold the histogram of the horizontal projection by a value two standard deviations above the mean. This returns a binary vector where each consecutive group of 1s represent a staff line, and each consecutive group of 0s represent a white gap. We then construct a second vector where the widths of the white gaps (as determined from the binary vector) are listed in sequential order from the top to the bottom of the page. A stanza segment is typically defined to be two staves (each consisting of 4 small white gaps in between each staff line) with a large white gap in between. To locate the center of a stanza segment with two staves (a number that is typical for most music sheets, one for the treble clef and one for the bass clef), we convolve the second vector with [-1 -1 -1 -1 0 1 1



(a) (b)

Figure 2: The horizontal projection (b) of a music sheet page (a).

1 1] (can be changed depending on the number of staves per stanza). If a value in the convolved vector is near 0, then the index of that value is the corresponding index of the white gap that is in the middle of two staves. The center of that white gap is then defined to be the center of a stanza. Based on all the centers of the stanzas for each page of a score, the music piece is divided respectively into stanza segments. The measure lines for each stanza are located by using vertical projection, in a way similar to how the stave lines were localized.

For each page of the music score, we can subtract each column of pixels by the binary projection vector to remove stave lines. Directly removing these stave lines however, can lead to fragmented and/or deformed musical symbols. In order to fix this, for each gap between black pixels caused by staff line removal, we project the black pixels directly above and below the gap towards the center of the gap.

3.3. Locate Musical Objects

Prior to performing observation based bounding box analysis and notehead template matching, it is nice to be able to estimate the average diameter of each note in the music sheets. Recall that the distance between two staff lines in the same stave is typically the height of a notehead. Using a radius range based on half of this value as an initial parameter, we can perform Hough circle transform over all sheet images. We then take the average over the radii of all detected circles to get a better estimate for the height of each notehead .

To locate musical objects, we run the flood fill algorithm, or alternatively known as seed fill. The idea is simple: to

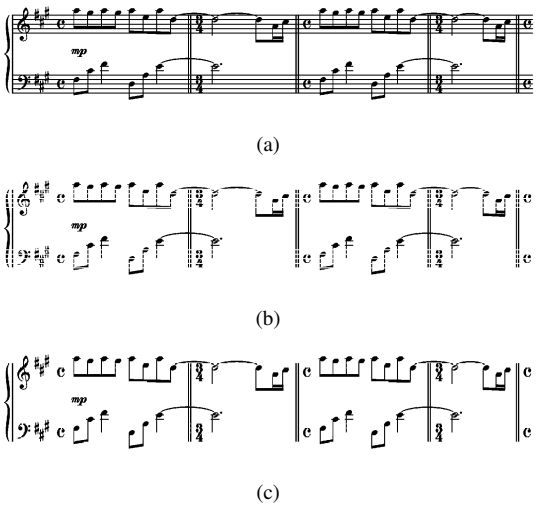


Figure 3: Staff line removal (b) of a stave segment (a), followed by gap stitching (c).

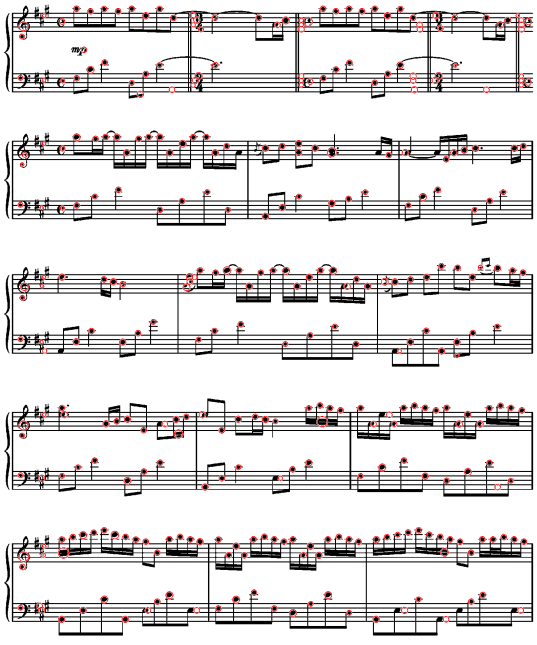


Figure 4: Hough circle transform visualization over all stave segments of a music sheet page.

isolate each chunk of black pixels based on connectivity. For each detected blob, we compute a respective bounding box. Despite gap stitching, some musical symbols can still be fragmented in certain parts (usually the tips) after staff line removal. To cope with this, we combine fragmented musical objects with an adjacency heuristic based on Euclidian distance that merges the bounding boxes together. Dots, defined to have both a bounding box width and height

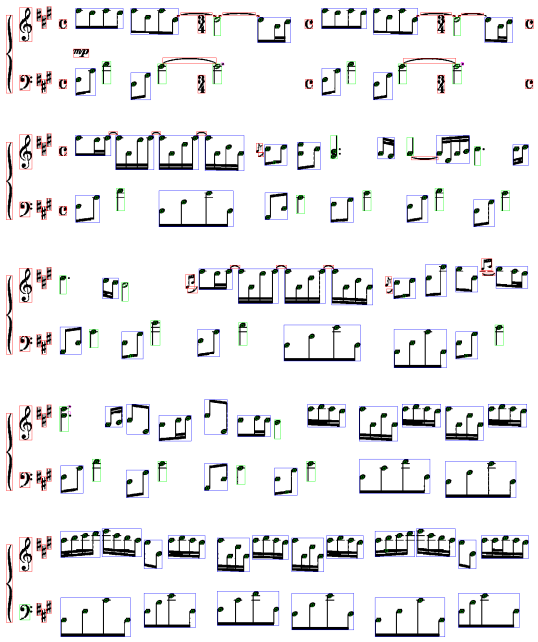


Figure 5: Sample bounding boxes of detected musical objects, categorized into beamed notes (blue), individual notes (green), and other symbols (red).

that is significantly less than the estimated notehead height, are detected and isolated for later in the detection of dotted notes.

3.4. Identify Musical Symbols

For each detected musical object, we generate a blank pixel canvas and copy the corresponding pixels of the musical object into the canvas. We resize three notehead templates (filled note, half note, whole note) such that their height is equal to the estimate notehead height found above. We then compute the matching score response of the template within the canvas using Normalized Cross Correlation (NCC), and isolate peak responses by thresholding by a relatively high value. The centroids of these peaks are then defined to be the location of the notes, where the type of the notehead corresponds to the template that returned the highest response in the peak.

We can associate each musical object (detect using floodfill) into one of three particular categories. First, we can isolate the musical objects that that do not return a peak from notehead template matching into their own category. This category is set aside for future work, in which non-notehead symbol identification can be performed using template matching or kNN. For the rest of the musical objects that do return a peak from notehead template matching, we categorize them based on bounding box size. If the width of a musical objects bounding box is larger than 3 times the

estimated average notehead height (computed from above), it is placed into the first category, group notes. Otherwise, it is placed into the remaining category, single notes.

For each group note object and single note object, given the locations of the detected noteheads in that object, we perform a variation of musical signature analysis in order to determine the length of the notes. For each notehead location, we project a superimposed vertical scan line in the musical object canvas. Using the frequency of white and black gaps, we can determine the length of the note. More specifically, for group note objects, if the notehead scan line returns one white gap in between two black gaps, the note is an eighth note. Likewise if the scan line returns two white gaps in between three black gaps, the note is a sixteenth note. If the notehead only returns white gaps, then the type of note is determined by contextual information (as an example, see figure). Notes that belong to the same chord are defined similarly, except that the region in between the upper and lower notes of the scan line are treated as a single black gap.

Figure 6: A set of beam notes and their respective scan lines (red) used for note length identification.

An alternative approach to the note length identification (that may work and is worth mentioning) would be a data-driven approach where the vertical and horizontal projections of the musical object canvas are used as feature vectors. This is left for future work, since I was unable to acquire mass amounts of data in time.

3.5. Generate Melody

The final step in the OMR system is to generate a respective melody using the time and pitch information obtained from the notes. When generating a MIDI file, we have 7 parameters to consider: note start (in beats), note duration (in beats), channel, midi pitch, velocity, note start (in seconds), and note duration (in seconds). The first and second parameters are almost equivalent to the sixth and seventh, scaled by beats per second. For each stave in a stanza, we create a new channel and associate a clean time sequence (in the form of a stack) to insert the corresponding detected notes (within that stave) into. For two stave stanzas, this typically amounts to two channels, one representing the left hand, and the other representing the right hand.

To recap, we know the location in pixels of each note, as well as its corresponding length in time, deduced from its note type. For each note, we use the y coordinate of the notehead and its contextual relation to the stave lines, in order to determine the pitch of the notehead (using the distance to the stave line representing the E note divided by the average gap size between the stave line). Using the number of detected sharps and flats in the far left of each stanza, we can also determine the key of the song, to which particular notes in the scale that we work with is shifted respectively. The notes are then added to the time sequence stack one by one for the corresponding channel, spaced in between based on their note lengths. To deal with miscalculated note lengths, whose error can reciprocate through the time sequence, we can perform x axis alignment analysis over the sheet music and stretch/shrink note durations accordingly such that notes in separate channels which are visually aligned in the stanza, are aligned within the generated time sequences. Conceptually, we snap the note lengths such that the melodies are aligned at certain places.

Implementation wise, for each channel, we temporarily put on hold the notes to be added to the time sequence until x-axis aligned notes are found in at least two channels. If the total sum of the time lengths for the notes that are put on hold for one channel is the same as the total lengths for the notes of the other channel, all temporary notes are added to the time sequences as normal. However, if the total sum of the times length of the notes that are put on hold for first channel exceeds that of the other channel, then the lengths of all of these notes in the first channel are scaled a ratio such that the total sums match. Then the notes are added to the time sequence. The time sequence for each channel with all of the added notes are then loaded into a MIDI file.

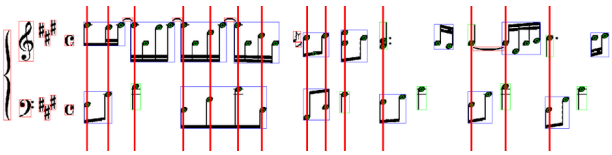


Figure 7: A sample stanza with its points of alignment between both of its channels highlighted in red.

I personally thought it was pretty interesting to be able to play around with the velocity parameter for the MIDI file generation to control the dynamics of the melody, as well as some cool time functions to speed up and/or slow down portions of the song. A great combination of both parameters could potentially elicit some fake emotions into the song.

4. Concluding Statements and Future Work

Within the scope of this project, in which we explore OMR using parametric observation based techniques, we make under two major assumptions about the music score input. The first aforementioned assumption is that our OMR system can only work properly with digital, high resolution

images of music scores. With that being said, by high resolution, we expect each staff line to have a height of at least 1 pixel. The second major assumption is that the staff lines for the sheet music should have minimal distortion (to be straight and very near horizontal).

There are a number of features we hope to implement in the near future. The project in its current state handles two staves per stanza (two channels), and assumes a treble clef stave orientation for the first channel and a bass clef stave orientation for the second channel. We are currently working to develop recognition for sharps, flats and naturals as well as a technique for using spatial information to relate these symbols to the noteheads. We also want to develop recognition for key changes and transitions in syncopation using measures and digit recognition. And finally, using textual information obtained from certain areas of the music sheets, as well as slurs, legatos, etc., we hope to be able to implement some interesting dynamics into generated melodies.

From this project, I learned that OMR is really hard. Current approaches are highly parametrized, causing systems to fail given any major fluctuations in musical symbol style and note-related spatial information. Furthermore, there are currently no reliable data-driven techniques out there that can substantially reduce the number of observation-based operations traditionally used in most OMR systems. Although for this project, I had proposed to find that magical data-driven method by the final project due date, I now realize just how difficult that is to find, and just how crazy I must have sounded when I introduced my proposal. :)

Thank you Professor Efros for this opportunity to explore optical music recognition!