Optical Music Recognition and Playback on Mobile Devices

Steve Dai, Ye Tian, Chun-Wei Lee
Department of Electrical Engineering
Stanford University
Stanford, CA 94305, USA.
stevedai@stanford.edu, yetian@stanford.edu, chunweil@stanford.edu

***Abstract*—The recognition of printed sheet music has been studied since the late 1960s. However, current applications often require careful scanning of musical scores, making it inconvenient for average music enthusiasts. With the advances in mobile devices and network connectivity, we develop an optical music recognition system to remotely interpret printed sheet music captured by an Android phone and present the music through MIDI playback on the device. By applying various image segmentation, filtering, morphological processing, and feature detection techniques, the application is able to identify the pitch of the notes by spatial position, form chords based on spatial alignment, and generate corresponding music file through MIDI table lookup. This application can be used by music learners in practicing unfamiliar pieces of music. It can also be deployed as a software extension to the control unit of player pianos currently in the market.**

Keywords-optical music recognition; mobile image processing;

I. INTRODUCTION

Optical music recognition consists of the interpretation of sheet music into playable form. It has been a field of interest since the late 1960s. Currently, there are a number of developed applications on the market for interpreting scanned printed sheet music.

In the past few years, mobile phones have quickly evolved into mini-computers with high-quality cameras and broadband data connectivity. As a result, we propose to integrate the optical music recognition technology into the mobile environment. Improvements of mobile cameras enable us to capture music notations directly with a mobile device, and a smartphone's media capability allows it to process and play the generated music file. Compared to its expensive non-portable counterparts, a portable mobile music recognition system would serve as a convenient and cost-efficient alternative for the average music enthusiasts.

Our system prototype consists of an Android phone and a sever with MATLAB installed. (Figure 1) First, an image of a piece of music is captured by the phone's camera and then sent to the server. Then, the image is processed by MATLAB, in which our functions identify the note heads, stave lines, and clefs. Finally, a MIDI file is generated by MATLAB based on the previous analysis and sent back to the Android phone for playback.

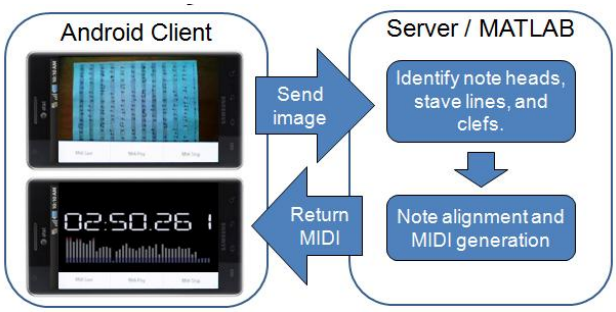

Figure 1: System Overview

II. ALGORITHM AND PROCESS FLOW

A. *Pre-Processing*

The original RGB image captured by the mobile camera (Figure 2) is first resized to a resolution appropriate for the subsequent processing steps. The resizing is necessary because the structuring elements and image templates used later in the flow are fixed in size. Therefore, any significant deviation from the desired scale may cause a breakdown of the entire flow. In addition, stave detection relies on the counting of the number of pixels, a method that is also largely scale-invariant.

After converting the RGB image into grayscale (Figure 3), we binarize the grayscale image by either applying a locally adaptive thresholding algorithm or using rank filtering followed by global thresholding. For locally adaptive thresholding approach, we use a window size of 128 pixels by 128 pixels and a step size of 64 pixels, and employ Otsu's Method within each window at each step to obtain the local optimal threshold. For the rank filtering approach, we use a order-statistic filter to calculate the background of the image by replacing each pixel in the image by the 100th brightest pixel in a 200 pixels by 200 pixels neighborhood. The resulting background is then subtracted from the grayscale for global thresholding using Otsu's Method. From experiment, the rank filtering approach tends to create a cleaner binary image compared to the locally adaptive thresholding method, which generates undesirable black and white tiles for the uniform regions of the image. (Figure 4)

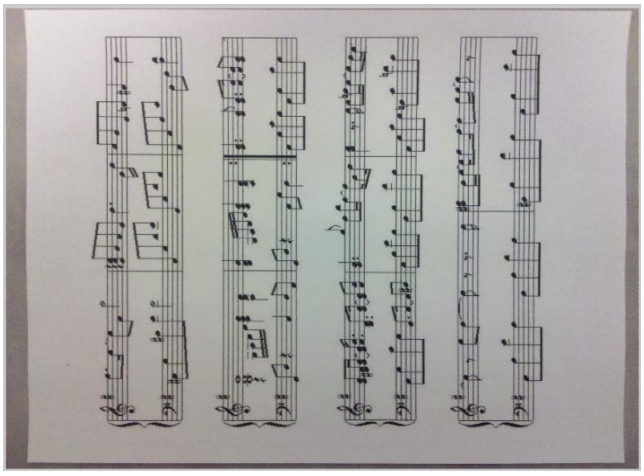

Figure 2: Original Captured RGB Image

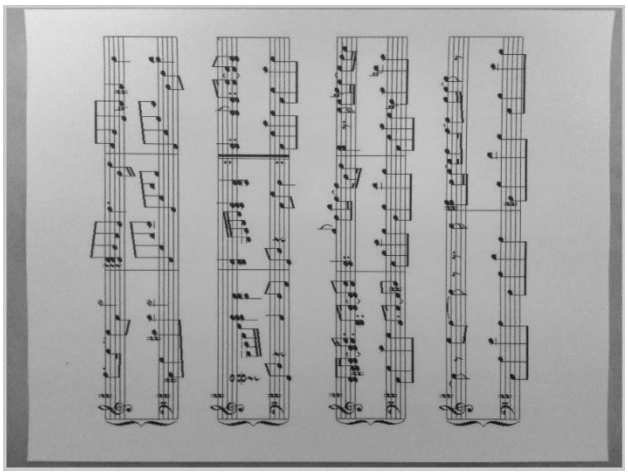

Figure 3: Grayscale Image

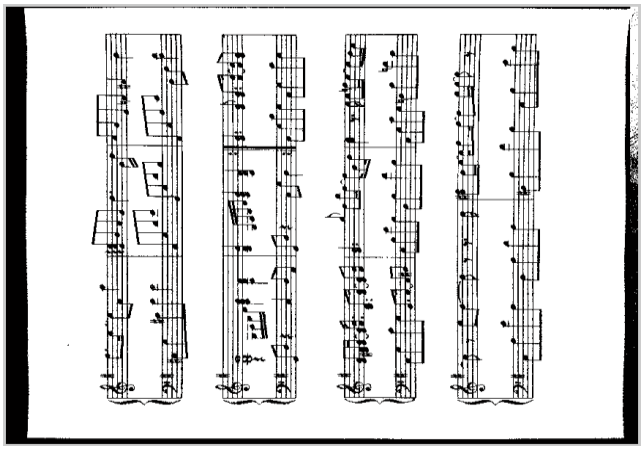

Figure 4: Binary Image with White Background

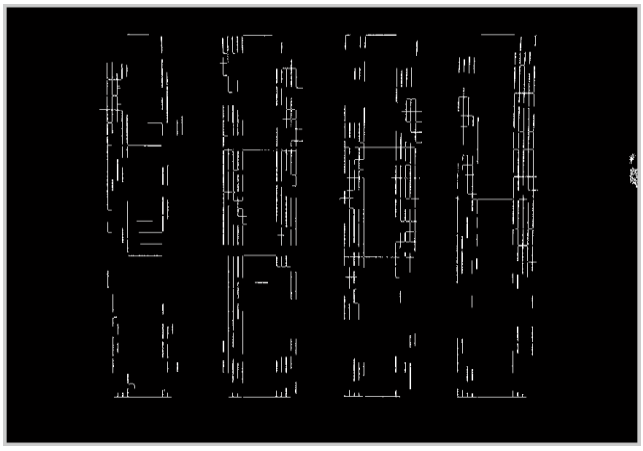

Figure 5: Improved Binary Image with Black Background

To improve image quality for line detection, we perform morphological opening to estimate the background, and create a copy of the binary image with the detected background subtracted. Then, regions whose areas span less than 100 pixels are removed from the background-subtracted binary image to generate the improved binary image. (Figure 5)

B. Rotate Image

Before the actual note recognition task, it is essential to rotate the sheet of music to an upright position. For this purpose, we perform a standard Hough transform on the original binary image (Figure 4) and identify the peak in the Hough transform matrix. Because there are significantly more horizontal lines on a sheet of music than vertical lines, the peak in the Hough transform matrix reveals the tilt angle of the musical notations. Rotating the image based on this angle would result in staves that are close to horizontal.

Despite the convenience of the Hough transform, there are several issues inherent to its application in our particular situation. On one hand, we can simply tell whether the staves are horizontal, but not whether the image is upright or upside down based on the Hough peak. We will consider this issue later in the report. On the other hand, because of the discrete nature of the angle in the implementation of the Hough transform as well as the inherent distortion of an image captured by a camera, rotated staves would not necessarily turn out completely horizontal. Often times, we observe a deviation of up to 5 pixels from one end of a stave to the other end.

C. Locate Treble and Bass Clefs

Using appropriately-sized treble and bass clef templates, we employ convolution, a basic template matching method, to locate the clefs on the binary image. The location where the convolution output is highest should correspond to the centered locations of the clefs. Subsequently, we can use the detected locations of the clefs to determine whether a note belongs to a higher or a lower octave.

An interesting use of clef matching lies in determining the correct rotation of the image. For a sheet of music that is upside down, clefs would be located on the right-hand side of the page as opposed to the left-hand side. If the clef matching algorithm determines that the peaks of the convolution are on the right-hand side of the page, then it can alert the system to rotate the page by 180 degrees before further analysis. We find from experiment that it is acceptable to use the same clef template even if the clefs are on the right-hand side of the page. The treble clef, especially, is shaped in a way that matches well in either orientation. In our implementation, we take the mode of

the top 25 detected peak locations individually for the treble clef and the bass clef. We determine that the page is upright if the mode is located on the left one-third of the page. Otherwise, it is upside down and requires rotation.

D. Stave Detection

There are several methods to detect the line positions of the staves. One possible algorithm is to iterate through the pixels of the improved binary image. (Figure 5) Whenever a "1" is detected, check the pixels under it within a range of, for example, 100 pixels. If these pixels meet the pattern "1…0…1…0…1…0…1…0…1", we can use the starting pixel and the ending pixel as the positions of the top and bottom line of the stave, respectively. Using this method, the positions of eight sets of top and bottom stave lines are obtained. (Figure 6)

Another possible algorithm is to perform a Hough transform on the improved binary image and remove detected lines that are too short and those that are not close to horizontal. From the lines that are left, we use the first one as the position of the top of the first stave, then search for the next line down that is beyond a certain number of pixels vertically from the first line. This number is selected so the next line selected is the bottom line of the first stave. We continue this process with the same threshold of vertical pixel counts, and the next line selected would be the first line of the second stave. This process ends when the bottom line of the last stave is detected, and there is no more line to follow.

Regardless which algorithm is used, we have to take into consideration the slight slant in the staves. As a result, the positions of the sets of top and bottom stave lines become a function of the horizontal position of the page. The vertical positions of the staves are shifted from left to right of the page. Our source code currently does not consider such vertical shift of approximately 3 to 5 pixels, thus sometimes causing a deviation of one note.

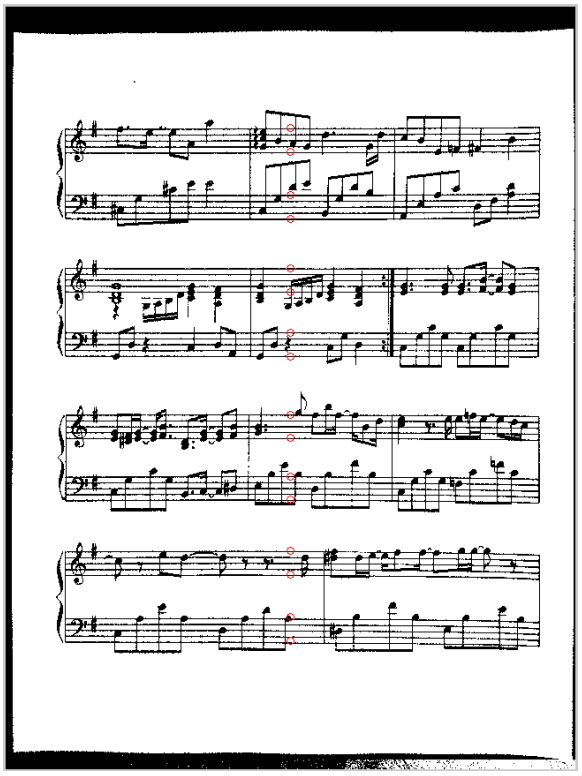

Figure 6: Rotated Sheet with Top and Bottom Stave Lines Marked

E. Solid Note Recognition and Identification

Starting with an inverted binary image (Figure 7) in which the note heads are white and the background is black, a horizontal open filter `(SE1 = [1 1 1 1 1 1 1 1 1 1 1])` is used to remove the vertical lines (Figure 8) followed by a vertical open filter `(SE2 = [1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1])` used to remove the horizontal lines (Figure 9). Some regions whose areas are either smaller than a lower limit or larger than an upper limit are removed because they are highly unlikely to be solid notes. The resulting sheet music are shown in Figure 10.

Once we are able to single out the solid note heads, we perform region labeling and centroid detection on the note heads. Based on the y-positions of the centroids in relation to the positions of the stave and the positions of the clefs, we can determine the pitch of each detected note.

F. Align Simultaneous Notes

Based on the horizontal positions of the detected note heads, we align notes for corresponding staves. In this project, we assume that all detected notes are of the same duration and do not consider the differences between, for example, whole

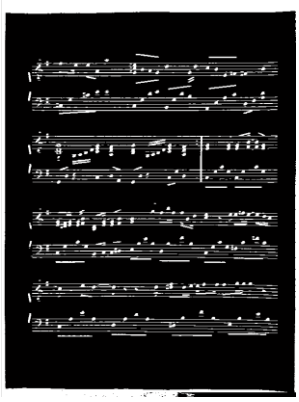

Figure 7: Inverted Binary Image

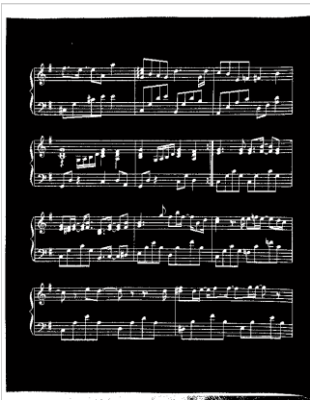

Figure 8: Vertical Lines Filtered

Figure 9: Horizontal Lines Filtered

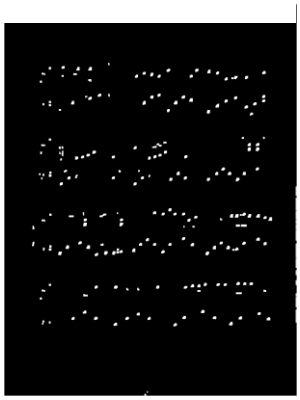

Figure 10: Image of Solid Notes

Octave #	Note Numbers											
	C	C#	D	D#	E	F	F#	G	G#	A	A	B
0	0	1	2	3	4	5	6	7	8	9	10	11
1	12	13	14	15	16	17	18	19	20	21	22	23
2	24	25	26	27	28	29	30	31	32	33	34	35
3	36	37	38	39	40	41	42	43	44	45	46	47
4	48	49	50	51	52	53	54	55	56	57	58	59
5	60	61	62	63	64	65	66	67	68	69	70	71
6	72	73	74	75	76	77	78	79	80	81	82	83
7	84	85	86	87	88	89	90	91	92	93	94	95
8	96	97	98	99	100	101	102	103	104	105	106	107
9	108	109	110	111	112	113	114	115	116	117	118	119
10	120	121	222	123	124	125	126	127				

Figure 11: MIDI Table

notes, half notes, and quarter notes. Therefore, the beat in the generated music is mapped from the horizontal position of the notes. If notes of corresponding staves are in close proximity to one another horizontally, they would be played at the same time as a chord in the generated music. In our implementation, close proximity means that the centroids of the notes are within one and a half times the width of a note.

G. Generate MIDI File

With the help of a third-party API for MIDI processing, we generate the MIDI file by mapping the pitches of the detected notes to corresponding MIDI numbers based on a table (Figure 11). Each note in the MIDI format requires a number identifying its pitch, a start time and end time, as well as its volume.

III. Implementation on Android Platform

Our user interface is built on the Android platform so to allow users of the program to easily and quickly capture images of sheet music with their smartphone. The image is then sent to a server for processing on MATLAB. Upon completion of the entire MATLAB flow, the server will return a MIDI file to the Android device for playback with a click of a button.

A JPEG image captured at 5.3-megapixel resolution has a size of approximately 1.5 megabytes and requires about 10 seconds to be transferred from the client to the server through an average 3G connection. After sending out the image, the client will await response from the server. The server runs the analysis of the image on MATLAB and generates a response to the Android client. The client then downloads the MIDI file from the server and save it onto its SD card. User can then play the MIDI file on the device, for which our app invokes the MediaPlayer Object.

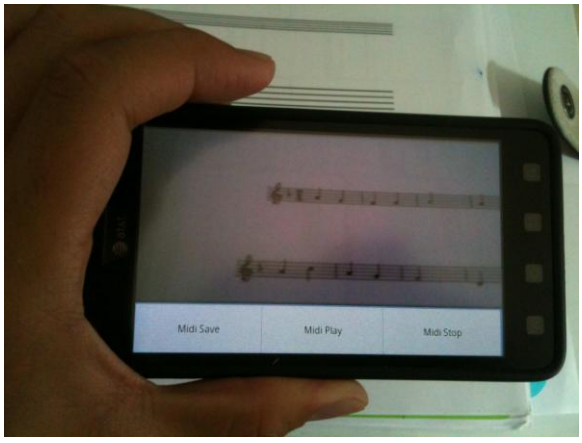

Figure 12: User Interface on Android

IV. Experimental Results

The system has been tested with more than four different pieces of printed sheet music. We will demonstrate two of them in this report.

A. Demonstration 1: For Elise

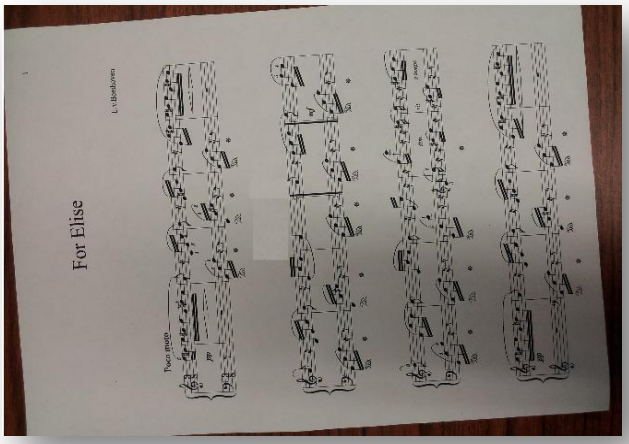

Figure 13: Demo 1 Original Tilted RGB Image

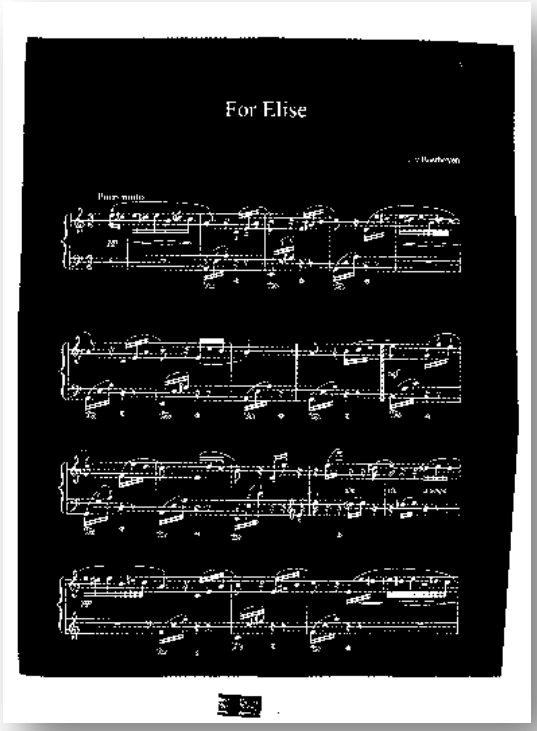

Figure 14: Demo 1 Rotated Binarized Image

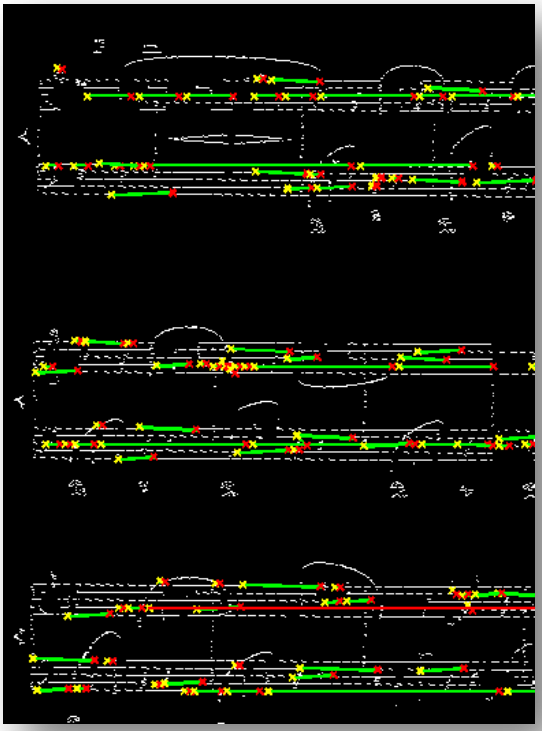

Figure 15: Demo 1 Detected Stave Lines

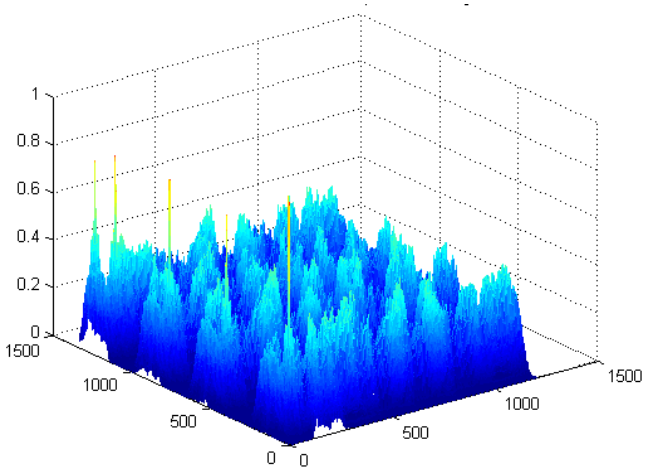

Figure 16: Demo 1 Treble Clef Peaks

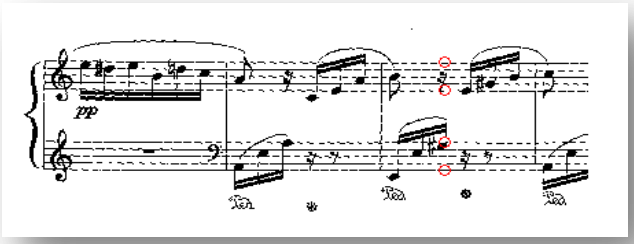

Figure 17: Demo 1 Top / Bottom Stave Positions

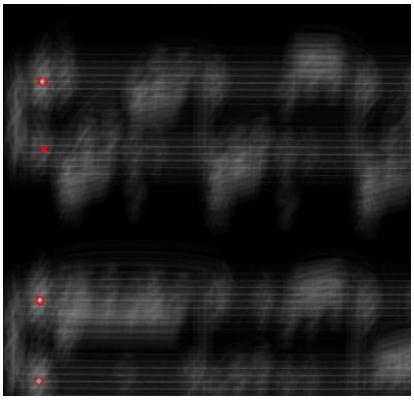

Figure 18: Demo 1 Template Matching Convolution Peaks

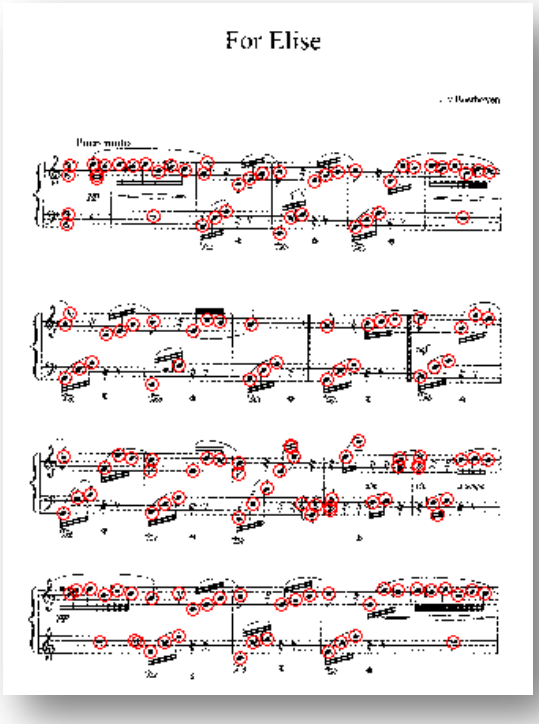

Figure 19: Demo 1 Detected Note Heads

B. *Demonstration 2: I Believe*

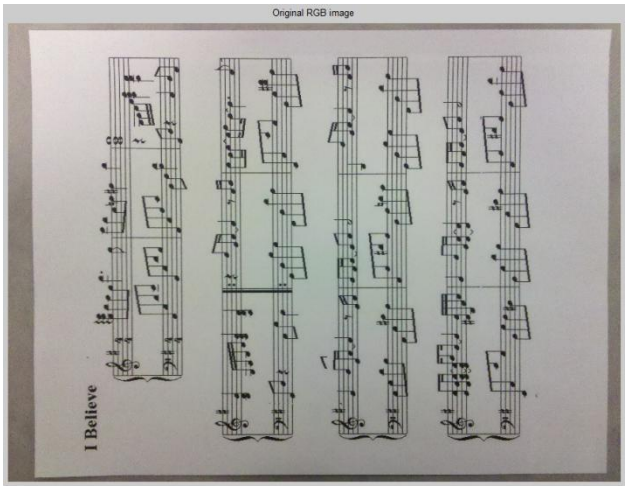

Figure 20: Demo 2 Original RGB Image

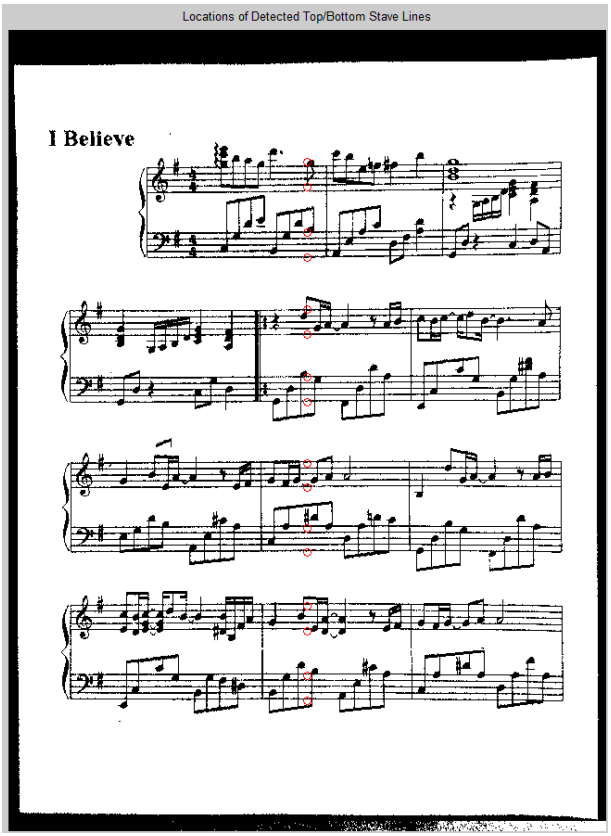

Figure 21: Demo 2 Rotated with Stave Detection

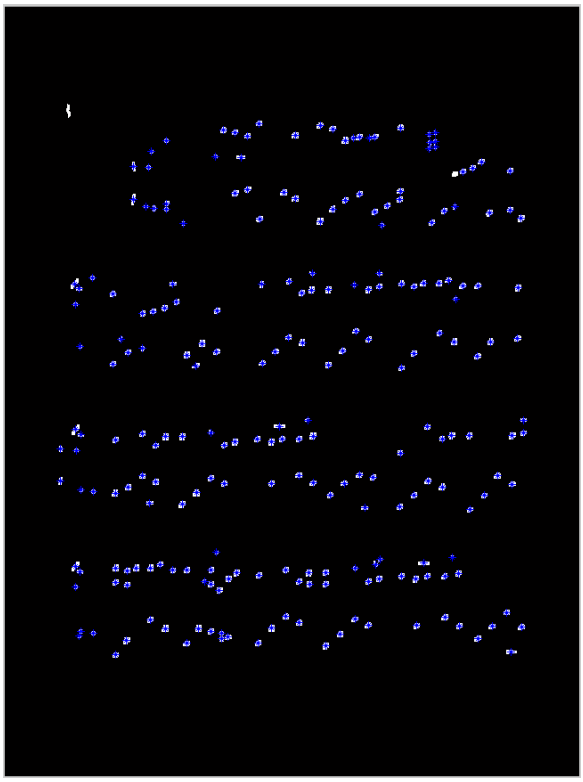

Figure 22: Demo 2 Detected Notes

V. Future Improvements

There are a few aspects in which this project can improve to obtain more accurate regeneration of the original sheet music. Although we don't have the chance to implement them due to time constraint, they would certainly be interesting to work on for the future.

A. *Fetch Music from a Database*

In our daily life, sometimes when a short piece of familiar song is played, we can immediately tell which song it is. It is possible to build a recognition system similar to that of our brain. By using the first few notes of a piece of music as reference, the system would be able to determine the title of the music and retrieve a well-performed version of that music.

First, we build a database that contain a large number of good-quality music. Then after the sheet music is analyzed, we should be able to match the first twenty notes with the corresponding music in the database. Once we find a high-probability match, that music file would similarly be streamed to the Android phone for playback. If no match is found, a MIDI file could be generated as what we have done.

B. *Add Timing*

So far in our project, timing has not been taken into consideration. As a result, the lengths of all notes in the MIDI file are uniform, failing to represent the rhythm of the original music. On top of the recognition of solid note heads, we need methods to recognize hollow note heads, tails, and dots representing notes of other durations.

C. Add Sharps and Flats

We have also not taken into account the sharps and flats for the notes, leading to another limitation in the MIDI playback. Other than playing the music in C-major, it is difficult to generate a euphonious MIDI playback without the sharps and flats. Template matching may be a feasible method for extracting the sharps and flats

VI. Conclusion

We have tried many different image pre-processing techniques to improve the image quality of sheet music before feeding it in for analysis. These techniques have been generally very successful in cleaning up the image before recognition. With a cleaner image, we were able to use mainly morphological processing and feature matching methods to extract the lines and the notes. More than 80 percents of the note heads on the original sheet music have been successfully detected. Further improvements in stave detection would help produce a better sounding music. Nonetheless, we are able to use an Android mobile phone to take a picture of music notations and get a simple MIDI file in return from the server within 1 minute under normal circumstances. With our prototype, we hope that this powerful concept of mobile optical music recognition would one day be introduced in the mainstream consumer market.

VII. Work Breakdown

All work related to the MATLAB coding is finished by Steve Dai and Ye Tian. All work related to connecting the Android phone to the server is done by Chun-Wei Lee. The poster is done by Steve Dai and Ye Tian. Each member is responsible for writing his part of work in the project.

References

[1] Fujinaga, Ichiro. Adaptive Optical Music Recognition. Thesis. McGill University, 1996. Print.

[2] Raphael, Christopher, and Jingya Wang. "New Approaches to Optical Music Recognition."12th International Society for Music Information Retrieval Conference (ISMIR)2011: 305-10. Print.

[3] Milan Sonka, Vaclav, Hlavac, Roger Boyle, Image processing, analysis, and machine vision, 2008.

[4] Berne Jähne, Practical handbook on image processing for scientific and technical applications, 2004.

[5] Bainbridge, David, and Tim Bell. "The Challenge of Optical Music Recognition." Computers and the Humanities 35 (2001): 95-121. Web.

[6] Gonzalez, R. C., R. E. Woods, and S. L. Eddins, Digital Image Processing Using MATLAB, Gatesmark Publishing, 2009.

[7] Soille, P., Morphological Image Analysis: Principles and Applications, Springer-Verlag, 1999.

[8] MacMillan, Karl, Michael Droettboom, and Ichiro Fujinaga. "Gamera: Optical Music Recognition in a New Shell." Web. <http://www.music.mcgill.ca/~ich/research/icmc02/icmc2002.gamera.pdf>.