Presented at the DLSU Research Congress 2017
De La Salle University, Manila, Philippines
June 20-22, 2017

SKIMusic: A Mobile Sheet Music Player

Art Edrick Choi, Theodore Fernandez, Bryan Reyes, and Joel Ilao

College of Computer Studies
De La Salle University-Manila
2401 Taft Avenue, Manila City,
1004 Metro Manila, Philippines
(+632) 524 0402
{art_choi, theodore_fernandez, bryan_dale_reyes, joel.ilao}@dlsu.edu.ph

Abstract: In this paper, we describe the design and development of an Android-based mobile application, called SKIMusic, for reading beginner grades 1 and 2 piano sheet music, with corresponding music playback. SKIMusic can capture an image of a printed music sheet using the mobile device's camera, and use a rule-based Optical Music Recognition (OMR) algorithm to encode it into MusicXML format. It uses a MusicXML reader for music rendering and audio playback. The developed prototype is benchmarked against existing mobile OMR applications such as SnapNplay and PlayScore Lite, which are both downloadable from Google Play Store. The implementation of the Object Recognition of SKIMusic is shown to be faster compared to SnapNplay's and PlayScore's. The overall OMR performance of SKIMusic has an average accuracy of 77.62%, for captured and ideal test images. Using the same test image set, SnapNplay was not able to perform since images taken from the mobile device's camera using the app are blurry, whereas PlayScore yielded an average accuracy of 74.17%. Future work may focus on expanding the coverage of musical elements that can be recognized, and explore the use of advanced machine learning and data-driven approaches in implementing the object recognition module.

Keywords: Optical music recognition, mobile application, sheet music, Music Theory

1. INTRODUCTION

Traditional ways of learning how to play music are via enlisting the aid of a music teacher, and using music transcriptions such as tablatures or scores. The best way to learn to play the piano is to take piano lessons under a qualified instructor. The music instructor can continually guide and provide feedback on the performance of the student that no book, video or any amount of self-study can provide. However, a teacher is not always accessible, and beginner musicians encounter tremendous difficulty learning how to play musical pieces due to lack of guidance and timely feedback. Suppose there is a tool where beginner musicians can simply take a picture of the sheet music they wish to hear and the software will play the piece to them? This solution is achievable with the use of Optical Music Recognition (OMR).

There already exist software systems that use OMR, such as SharpEye, Optical Music Easy Recognition, and Gamera; these applications, however, are only accessible through a computer desktop. Another OMR system is Audiveris; it is an open-source software which processes the image of a music sheet to automatically provide symbolic music information in MusicXML standard (Bitteur, 2013). MusicXML is a standard open format for exchanging digital sheet music. It was designed from the ground up for sharing sheet music files between applications, and for archiving sheet music files for future use (Prairie, 2015).

With regards to OMR, desktop applications outnumber mobile applications. Mobile devices and applications provide the advantage of mobility, and with conveniently accessible built-in cameras. One such mobile application is iSeeNotes (iSeeNotes, 2013). Using iSeeNotes, the user can take a picture of a printed piece of sheet music using his/her smartphone or tablet, and play the corresponding music. However, the ability of iSeeNotes to read and process the sheet music is only limited since it only supports a limited number musical elements, and occasionally does not process the sheet music image despite the clarity of the captured image. In our project, we have developed software, similar to

iSeeNotes, using the OpenCV Framework (OpenCV, 2015 for image processing and OMR. The software can also recognize musical elements amd encode them using MusicXML and SeeScore for Musical Instrument Digital Interface (MIDI) audio playback

2. SYSTEM DESIGN

The main process flow of the application is shown in Figure 1.1.

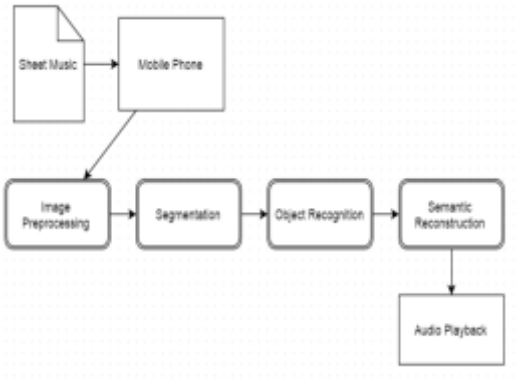

Figure 1.1: Main Process Flow

The software application's platform are smartphones using Android operating systems specifically those with Android version 4.4, also known as KitKat or higher, and Java as the programming language. In this project, a smartphone that has a camera resolution of at least 8 megapixels is recommended, to ensure comprehensibility and clarity of the image.

For this project, printed classical sheet music of grades 1, 2, and 3 are used. The list of elements recognizable by SKIMusic are: staffs or staves, the bass and the treble clef, accidentals such as sharps, flats, and naturals, key signatures, note durations, beams and flagged notes, tied notes, and rests. Furthermore, SKIMusic can recognize additional musical elements such as time signatures 2/4, 3/4, 4/4. Moreover, the tempo for the sheet music will be manually input by the user.

It is recommended that printed sheet music images be captured in landscape mode, since the staves are drawn horizontally. Landscape would be more appropriate since it can capture the details of the notes more comprehensively, compared to the portrait layout.

2.1. Image Preprocessing

The application will first prompt the user to crop the image that was previously taken, and use the cropped region as the input for the proposed system using the Android Image Cropper library (ArthurHub, 2016). The cropped region will then be processed and be converted into a grayscale image and then converted to a binary image using adaptive thresholding. As shown in Figure 2.1, since the input image has strong illumination, using Otsu's threshold will result in a poor binarized image as shown in Figure 2.2. Also, median filtering is used to remove salt and pepper noise on the captured image. The advantage of using Otsu's thresholding is that it automatically calculates the best threshold value in a global scale of the image. It is only best used in bimodal histograms in which there are two peaks in the image histogram and has good bimodal distribution. Since there are varying lighting conditions in different areas in the image, the bimodal distribution will be destroyed and it is not good to select a single global threshold that will neatly segment the object from its background, therefore adaptive thresholding is used as the thresholding operation, since this approach will get different thresholds for different regions of the given image as shown in Figure 2.3.

Afterwards, the system will perform skew detection using Hough line transform to detect the angle of skew. The skew angle is defined by the degree of angle in which the lines are not horizontal or parallel to the x axis. The skew angle is equal to the average skew angle of each line detected by the Hough line transform algorithm. The angle of the line is calculated by getting the slope of the line based on the two points taken from Hough line transform and the angle of the line can be computed by calculating the inverse tangent of the slope. Lines that have angles greater than 75 degrees but less than 105 degrees are filtered out in this process to maintain a good skew angle. The skew angle will then be used to rotate the image accordingly using affine transformation of OpenCV, in which it is used to fix distortions of the image and also the set of parallel lines remains parallel after transformation, such that the staff lines are perfectly horizontal in order to properly detect and remove the staff lines. The way affine transformation works here is that it will be fed by a 2D rotation matrix in which it contains the center of rotation of the image and the rotation angle as

its parameters. The rotation angle will be the skew angle calculated by the program.

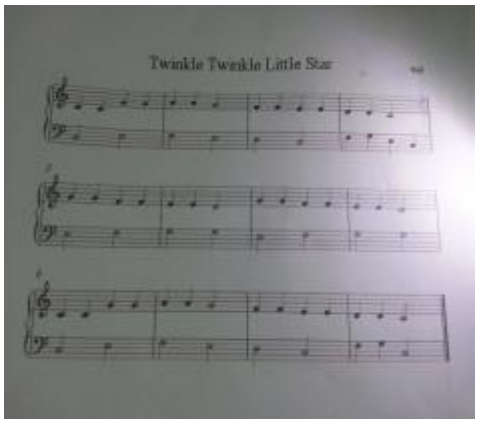

Figure 2.1: Input Image where Regions have Varying Illuminations

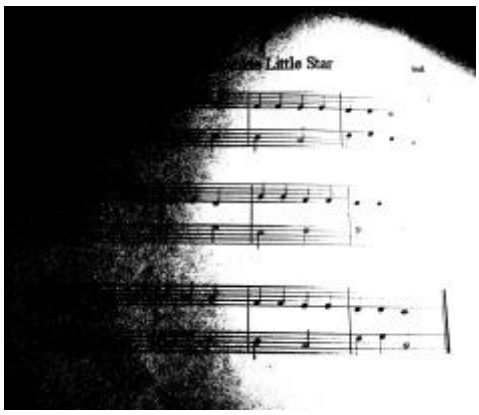

Figure 2.2: Result Using Otsu Thresholding

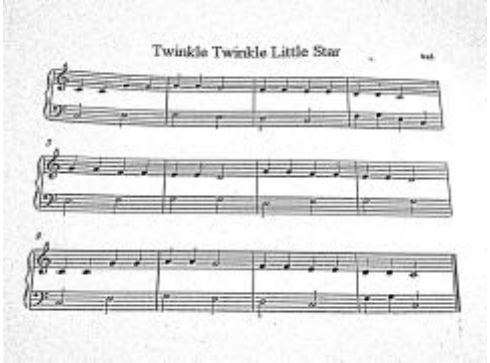

Figure 2.3: Result Using Adaptive Thresholding

Empirical values are used after testing the windows size that is used in the median blurring and adaptive thresholding algorithm and also for the threshold used for the Hough line transform algorithm. The values that the proponents used were tested through 30 sheet music, and these are the corresponding values that are best fit for the captured image. For the median blur algorithm, the window size value is 5, since having a higher window size will also blur the details that are needed like notes and the staff line are being fragmented. On the other hand, for the adaptive thresholding algorithm, the window size value is 25 since having a smaller window size makes the note head pixel value to white, which turns a quarter note into a half note. For cases of higher windows size value, the stem and the note head are being separated. Also, in some cases for the object recognition phase, an ellipse will not be detected if the note head of a quarter note/eighth note is fragmented. Lastly, for the Hough line transform algorithm, the threshold value used is 200, since it is more likely to detect a longer line, which is the staff line, rather than having a lot of shorter lines or line segments, since it will give a much more unreliable angle of the line.

2.2. Segmentation

The processed image would be segmented into staff lines and musical elements. Shown in Figure 2.4 is a data tree representation of the output for this module. The staff lines and the musical elements are stored, along with their x and y coordinates in order to be used in the next modules. Firstly, the staff lines are detected and removed using morphological operations, particularly erosion and dilation. Before the system can use erosion and dilation, a structuring element needed as a parameter in order to properly remove the staff lines. By using getStructuringElement() function of OpenCV, the system is able to create a rectangular structuring element with an anchor point int the middle of the element, and with a height of 1 and a width of the initial width of the input image and divided by values starting from 10 up to 100 to ensure that the best output is chosen by the user. Erode and dilate functions of OpenCV can now be used with the structuring element previously mentioned. When erosion is performed, the structuring element is scanned over the input image and the function computes the minimal pixel value that are overlapped by the structuring

element, the pixel value is then replaced with the minimal value. Thus, in the case of binary images, black pixels and objects are enlarge. On the other hand, dilation uses the maximal pixel value, thus white pixels and objects are enlarged. As the outputs of erosion and dilation vary depending on the size of the structuring element, the system prompts the user to choose an output with the least amount of fragmentation and noise from several output images of erode and dilate. Using the input image shown in Figure 2.5, the result of using erosion and dilation, given the structuring element mentioned above as a parameter is shown in Figure 2.6.

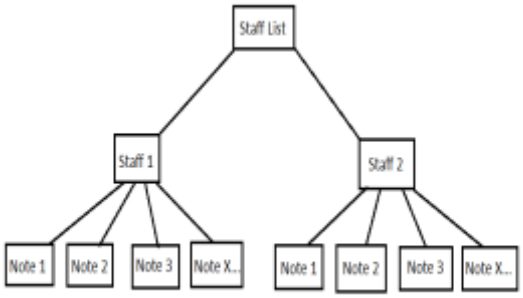

Figure 2.4: Data Tree Representation of the Output for the Segmentation Module

Figure 2.5: Input Image

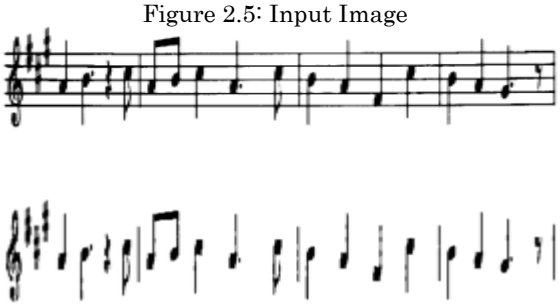

Figure 2.6: Result After Undegoing Erosion and Dilation

After the removal of the staff lines, the musical elements that are left will then be extracted using findContours() and boundingRect() functions of OpenCV. The system checks for contours that are fragmented which are caused by the removal of the staff lines; these contours are merged when they intersect each other as shown in Figure 2.7 and Figure 2.8. A disadvantage of merging contours is when noise that survived the noise suppression process, is merged with the actual musical elements, which may affect the accuracy of the Object Recognition module. Finally, the contours are further noise suppressed using area of the contour, the height and the width of the bounding box, as criteria. The remaining contours are the extracted musical elements, which is the final output of this module.

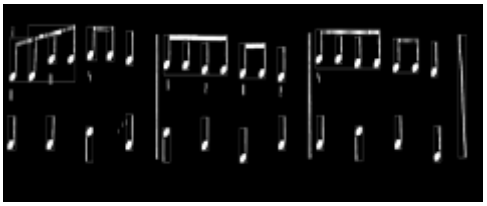

Figure 2.7: Before Merging Broken Contours

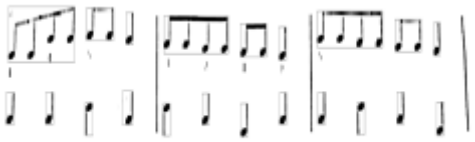

Figure 2.8: After Merging Broken Contours

2.3. Object Recognition

The features and primitive elements of the musical elements are extracted and are used to determine which particular musical elements are detected. The elements that are encapsulated and boxed shown in Figure 2.9, are the input for this module. Originally for this module, it was implemented using SURF of OpenCV, but was later changed because of resulting long processing time. The alternative that was used is using morphological operations such as erosion and dilation similar to the approach used in the Segmentation module. The features of the musical elements used to recognize each element consists of the x and y coordinates, the height and width of the bounding boxes, note heads, and lines.

The height and width of the bounding boxes are extracted using the boundingRect() function of OpenCV. Note heads and lines are extracted using morphological operations such as erosion and dilation on the contour input to isolate note heads and lines. Before the morphological operations are used, the system first generates a structuring element (e.g. an ellipse for note heads and a

rectangle with a width of 1 for vertical lines). A rectangular structuring element is implemented for vertical lines to have a width of 1 and a height that is 30% of the input contour. This is done to filter out objects that are less than the required height, therefore minimizing falsely recognized vertical lines. If the percentage is increased, some lines are not retained and therefore removed by erosion and dilation. Shown in Figure 2.9 is the input contour, and the result of using the aforementioned structuring element is shown in Figure 2.10.

Figure 2.9: Input Contour

Figure 2.10: Output Using 30% Height Structuring Element

Similarly, the elliptical structuring element behaves the same way, in order to filter out objects that can be falsely detected as ellipse. The elliptical structuring element's width and height are adjusted according to the number of vertical lines detected and the size of the input contour. The adjustment is done because of the different sizes of the input contours whereas the note heads found in each contour are similar in sizes relative to the whole input image. For example, contoured beam notes have larger width and height compared to when a single quarter note's, hence, the width and height percentage for the size of the structuring element to detect ellipse on such contour inputs should differ from each other.

Furthermore, the number of vertical lines found can further suggest that the input contour is not a single note, but instead are beam notes. Therefore, the size of the elliptical structuring element is based on the number of vertical lines found. Table 2.1 shows the rules for implementing the elliptical structuring element.

Table 2.1: Rules on Elliptical Structuring Element

Number of Vertical Lines	Size of Elliptical Structuring Element
1	40% width and 15% height of the input contour
2	10% width and 15% height of the input contour
4	5% width and 10% height of the input contour

The accuracy of recognizing elliptical structures depends heavily on the lines detected. As the number of lines detected increases, the percentage used for the size of the elliptical structuring element decreases. Avoiding miscount of lines is essential for a higher accuracy. After including the structuring element as a parameter in the "erode and dilate" processing step, the system may now isolate structures that are based on the structuring elements. The output of erode and dilate is used as an input for findContours() of OpenCV to count the number of ellipses and lines via the size of the output matrix of findContours(). In some cases, beam notes were also counted as ellipses due to fragmentation, erosion and dilation. Therefore, for each detected ellipse, circularity is calculated to determine whether it is indeed a note head. Similarly, the number of lines that are detected is determined by the size of the output matrix of findContours(). The system then checks each bounding box of the line contours if they intersect with each other, and merge them into one contour if they intersect. The checking is done to ensure that number of lines is correctly counted, since fragmentation on lines often occurs because of the previous processing of the image (e.g. Image Preprocessing and Staff Line Removal, etc.). Thus, some lines are counted as multiple lines. The ellipses and lines detected are then removed from the original input contour, to determine if there are features that do not fall under the structuring elements. Finally, after obtaining all the features needed, the system recognizes each input contours as a musical element based on the features detected. A set of rules is created for each musical element, shown in Table 2.2.

Table 2.2: Rules on Each Musical Element

Musical Element	Rules
Whole Note	2 contours adjacent to each other, with similar heights and widths. Height is also similar to distance between 2 lines in a staff.
Half Note	1 line, no ellipse, others >1, height>width
Quarter Note	1 line, 1 ellipse, 0 others, height>width
Eighth Note	2 lines, 1 ellipse, >=1 others, height>width
Beamed Eighth Note	>= 2 lines, >= 2 ellipse, >= 1 others, width > height

Using erosion and dilation from OpenCV, combined with the implemented set of rules, the system can recognize each contour as a particular musical element. An advantage of this approach compared to using SURF is that the processing time is reduced significantly. Furthermore, APK installation time of the application is also reduced due to the fact that the template images use for SURF is removed. Disadvantages of this approach include the lack of detection of features for some musical elements such as clefs and time signatures.

2.4. Semantic Reconstruction

The input of this module is an array list of staffs where all the detected staff and their corresponding notes are stored. There are two clefs that can be recognized by the system, particularly the treble clef and the bass clef, and each clef affects the pitch and octave of the notes differently, as shown in Figure 2.11. This module considers the type of corresponding clef and determines the pitch of the notes. The G or treble clef is applied to the first staff while the bass or C clef is applied to the second staff.

Depending on the type of clef detected, the exact pitch and octave of the notes are determined by comparing the y coordinates of the notes with the y-coordinates of the staff lines and spaces previously gathered from the Segmentation module. The output of the module is an array list of staffs where the pitch and octaves of each notes are reconstructed with regards to the clefs.

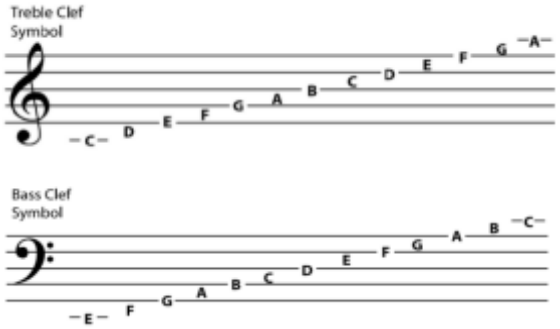

Figure 2.11: Pitch of Each Staff Line and Space of Treble Cleff and Bass Cleff Respectively

2.5. MusicXML Writer

This module will receive the array list of staffs from the Semantic Reconstruction module, traverses all the notes per staff, and convert them to their corresponding MusicXML equivalent. On writing the notes to MusicXML, it first checks for the contents of the first staff. Every note that is detected has its value, 0.5 for eight note, 1 for quarter note, 2 for half note and 4 for whole note, It will be added up and be stored to noteCount, in which if it reaches the number of duration per measure, it will stop writing and proceed to the next staff if there is. It will repeat the process and afterwards it will end the measure and repeat the process until all notes have been written. It was decided to include the bar, which is a symbol of the end of the measure, as a "note", since using the note counter only is vulnerable to some errors in the detection part; this entails testing a condition for when a bar is encountered and if notes have been detected and counted, a measure has ended. The resulting MusicXML file will then be passed on to next module for audio playback purposes.

2.6. Audio Playback

The MusicXML file will be played using the SeeScore library (Dolphin Computing Ltd., 2014). The MusicXML file will be saved and all information from the file will be used to create an SScore Java object. Furthermore, the SScore object will be further segmented into bars or measures using the BarIterator() method found in

Playdata, and further segmented into individual notes using the NoteIterator(). The note returns a midi pitch and start time and duration in millisecond. The MIDI file can then be played via the Android media player.

3. Results and Discussions

3.1. Accuracy Testing

The purpose of this testing is to test the accuracy of the final output of the system, which is the audio playback of the captured sheet music. Furthermore, different test inputs with varying perspective, lighting, angle, and quality of the image are used. Testing the accuracy of each note detected and their corresponding pitch and octave is important to determine the capabilities and limitations of the system.

3.1.1. Music Element Recognition

Two kinds of test inputs were used for this testing, namely, the ideal image version which are in PDF format, and the corresponding captured image version taken using the mobile device's built-in camera. The captured images were taken with varying perspective, lighting, angle, and quality. A total of 30 music sheets were used for the testing. Shown in Table 3.1 and Table 3.2 are the results and accuracy for Musical Element Recognition.

Table 3.1: Using Ideal Images (PDF)

Musical Element	Total # of Elements	Correctly Recognized	Accuracy
Quarter	544	536	99%
Eighth	543	472	87%
Half	163	144	88%
Whole	44	30	68%

Table 3.2: Using Captured Images

Musical Element	Total # of Elements	Correctly Recognized	Accuracy
Quarter	544	463	85%
Eighth	543	386	71%
Half	163	115	71%
Whole	44	23	52%

3.1.2. Pitch Detection

The testing done for Pitch Detection is identical to Musical Elements Recognition testing seen above. The only difference between these two accuracy testing is that the results of the Musical Elements Recognition, particularly the number of elements recognized correctly, is used as the basis for this testing. Seen on Table 3.3 and Table 3.4 are the results and accuracy for Pitch Detection of both PDF inputs and captured image inputs.

Table 3.3: Using Ideal Images (PDF)

Musical Element	Total # of Elements	Correctly Recognized	Accuracy
Quarter	544	463	94%
Eighth	543	386	96%
Half	163	115	91%
Whole	44	23	90%

Table 3.4: Using Captured Images

Musical Element	Total # of Elements	Correctly Recognized	Accuracy
Quarter	544	463	84%
Eighth	543	386	80%
Half	163	115	82%
Whole	44	23	83%

As seen in Table 3.5, the average accuracy is computed by multiplying the accuracy of the Musical Elements Recognition testing and the accuracy Pitch Detection testing of each musical element to obtain the average accuracy for each musical element since the process of musical element recognition and pitch detection are cascaded.

Table 3.5: Average Accuracy of the System

Average Accuracy (recognition and pitch)	Ideal Image	Captured Image
Quarter Note	92%	71%
Eighth Note	83%	57%
Half Note	80%	58%
Whole Note	61%	43%

3.2. Comparison Testing

The purpose of this testing is to compare SKIMusic to similar downloadable applications in the Google Play Store. The applications included in this testing are PlayScore (Dolphin Computing, 2016) and SnapNPlay (SnapNPlay, 2015) since they are the apps with similar features as SKIMusic, which is being able to capture a sheet music and play it. PlayScore is currently around 350 PhP and SnapNPlay is currently around 200 PhP on the Google Play Store as of March 2017. Since iSeeNotes is no longer available in the Google Play Store in the Philippines, it is no longer included in the comparison.

SnapNPlay's camera produces blurry images, with no facility to allow the user to adjust the focus of the camera. Due to this fact, accuracy testing cannot be performed on the SnapNPlay app. PlayScore, on the other hand, was tested for both ideal images (PDF) and captured images of 30 piano sheet music. These images are the same images that is used for the SKIMusic accuracy testing. Only the recognition accuracy is tested for the PlayScore since the pitch accuracy is hard to determine. The results are shown in Table 3.6 and Table 3.7.

Table 3.6: PlayScore Tested Using Ideal Image (PDF)

Musical Element	Total # of Elements	Correctly Recognized	Accuracy
Quarter	544	490	90.1%
Eighth	543	477	87.9%
Half	163	153	93.9%
Whole	44	42	95.5%

Table 3.7.: PlayScore Tested Using Captured Image

Musical Element	Total # of Elements	Correctly Recognized	Accuracy
Quarter	544	463	57.4%
Eighth	543	386	61.7%
Half	163	115	57.1%
Whole	44	23	50%

The average accuracy of PlayScore on the ideal images and on the captured images are 92% and 57% respectively. There is a huge difference between the ideal images and captured images since PlayScore was not able to read some of the captured images of sheet music as seen on Figure 3.1, while majority of the ideal images were read.

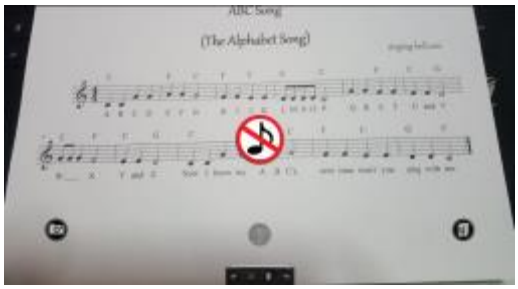

Figure 3.1: Captured Sheet Music Unrecognized by PlayScore

SKIMusic has an average recognition accuracy of 85.5% and 69.8% for ideal and captured images respectively, while PlayScore has an average of 92% and 57% for ideal and captured images respectively.

3.3. Observations

The quality, perspective distortions, and other factors of the captured sheet music greatly affect the accuracy of the musical element recognition as well as the accuracy of the pitch and octave for each musical element. The testing done with the PDF version of the sheet music yields a higher accuracy compared to the captured image further supporting the claim that the quality of the sheet music, perspective distortion, lighting, and other noise affects the accuracy of the system.

In addition, based on the observations, the reason for the decreased accuracy of the whole note compared to other musical elements is because that it has the least amount of features that can be extracted, such as the lines and the ellipses. The case for recognition of the whole note is for 2 contours that are near each other with a height and width of approximately the same as the line distance, which is the distance between two staff lines. This is due to the fact that whole notes are frequently fragmented as seen on Figure 3.2 by the staff line removal in the segmentation module, thus it is not detected by the system, although there are cases wherein the whole note is not fragmented as seen on Figure 3.3.

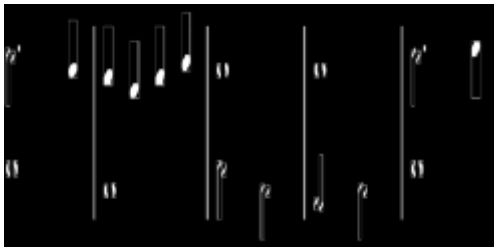

Figure 3.3: Output of Staff Line Removal from the Segmentation Module

Figure 3.4: Contour of a Whole Note Wherein It is Not Fragmented

Furthermore, there are cases wherein the lines of the elements are slanted due to the distortions of the image. This can add to the decrease in recognition accuracy for the quarter notes, eighth notes, and half notes.

Most of the fragmentations are caused by the staff line removal process in the segmentation module. This is due to the fact that erosion and dilation also affects the other musical elements other than the staff lines, which can be considered as a limitation of the algorithm. Additionally, the distortions in the image such as perspective distortion, lighting, and noise can contribute to the fragmentations caused by the staff line removal process. Since the Object Recognition module implements a rule based approach, the fragmentations and distortions caused by the previous modules can affect the recognition and pitch accuracy of the Object Recognition module.

For recognition accuracy on ideal images, PlayScore is 7% more accurate than SKIMusic. This is due to the fact that SKIMusic's whole note recognition accuracy is only 68%, meanwhile PlayScore has a recognition accuracy of 95%. This leads to a decrease in overall accuracy for SKIMusic. For recognition accuracy on captured images, SKIMusic is 13% more accurate than PlayScore. This is attributable to the fact that PlayScore was not able to detect some of the sheet music that SKIMusic detected. It may be concluded that SKIMusic can handle image distortions like lighting, perspective, image noise, and such better than PlayScore.

3.4. Limitations

A limitation of the system is that the user should capture the image at most 10 inches away from the sheet music which correlates with another limitation of the system, that it can only process at most 2 staff lines at a time, these limitations is set in order to maintain a standard size of the musical elements found in the sheet music for segmentation and recognition purposes.

Furthermore, to minimize perspective distortions, it is suggested that the mobile device is parallel to the sheet music. In addition, in order to maintain a higher accuracy of musical elements recognition and pitch detection, it is preferred that the user make use of higher quality version of sheet music, as blurry and noisy sheet music can affect the output of some modules, thus the accuracy of the whole OMR process is decreased.

The number of musical elements recognized by the system is limited only to whole notes, half notes, quarter notes, eighth notes, beam notes, and measure. This is due to the fact that other musical elements lack the features that can be extracted in order to be recognized by the implemented algorithm. Furthermore, some elements in the sheet music are fragmented and will deemed unrecognizable and considered as noise.

4. Conclusion

This project was able to utilize the algorithms of OpenCV Library and the functionalities of other libraries such as SeeScore and Android Image Cropper for the development of SKIMusic. The algorithms of OpenCV is used for a proper cleaning and image preprocessing of the captured sheet music is valuable and can be considered the most important stage of the system, as the input of the following modules and processes highly depends on the output of the image preprocessing stage. Moreover, a challenge to the study is when few of the proposed algorithms and functions of OpenCV consumes a significant amount of processing time, namely feature detection SURF of OpenCV and staff line removal algorithm.

SKIMusic is able to detect and extract the musical elements from a captured sheet music. boundingRect() and findContours() functions of OpenCV was used for the detection and extraction of the staff and the musical elements within the

staff. Limitations with the recent change in the proposed algorithms and functions is shown with the lack of musical objects recognize by the system. As some musical elements such as clefs and time signatures lack features to be recognize by SKIMusic's current recognition module. Positively, the alternative solutions' processing time is better when compared with the previous solutions.

The development of SKIMusic includes the construction and implementation of the rules for musical element recognition on extracted musical elements. Processing time plays a vital role in any mobile applications, thus minimizing processing time can maximize the essence of mobility and portability of such mobile devices. Thus, the implementation of rules to minimize the processing time is essential. Furthermore, the Object Recognition module of SKIMusic relies on rules for features detected is implemented by the proponents, these rules are unique to this research. In addition, other OMR systems and developers may look upon these rules for further improvements and enhancements for the entire OMR process.

SKIMusic converts the recognized musical elements to MusicXML format in order to output an audio playback. The audio playback of the system is dependent on the information found in the captured sheet music. Quality and other factors such as perspective distortions and lighting can affect the accuracy of the output audio.

The end result of this research is SKIMusic, a mobile OMR application on the android platform. SKIMusic has the advantage of portability since it is developed for android mobile devices with built in cameras and can be accessed easily compared to desktop OMR systems. A limitation of OMR applications on mobile devices is that the processor is weaker compared to desktop OMR systems. Thus, minimizing processing time helps the overall performance of the system. Furthermore, challenges tackled by the proponents includes several issues arises regarding the quality, lighting and illuminations, perspective distortions, and other factors of the captured sheet music. In addition, since SKIMusic is based on a mobile platform, these issues mentioned above cannot be overlooked, thus the accuracy of the system is dependent on the quality and other factors of the captured image input.

Multiple testing methods were performed to test the usability and performance of the SKIMusic. The accuracy testing evidently shows that when the system uses PDF versions of a sheet music as input, it yields higher accuracy compared to captured image versions. The results from the accuracy testing is further proof that lighting, perspective distortions, and etc. are factors to be highly considered when developing an OMR system that utilizes captured sheet music. The proponents were able to try, evaluate, and compare similar OMR based mobile applications such as, SnapNPlay(86) and PlayScore Lite(87). Although more steps are needed for SKIMusic to output audio playback and given that the factors such as, quality, lighting, etc. surrounding the sheet music are the same. SnapNPlay was not able to process any sheet music due to the blurriness of its camera's functionalities. On the other hand PlayScore is able to process the sheet music and yielded similar accuracy percentage compared to SKIMusic but takes significantly more time to process.

5. References

Bitteur, H. (2013). Audiveris. Retrieved from https://audiveris.kenai.com/

Prairie, E. (2015). Musicxml. Retrieved from http://www.musicxml.com/UserManuals/MusicXML/MusicXML.htm

iSeeNotes. (2013). iseenotes. Retrieved from http://www.iseenotes.com/

Dolphin Computing Ltd. (2014). Seescore - a musical score reader. Retrieved from http://www.seescore.co.uk/seescore/

OpenCV. (2015). Retrieved from http://opencv.org/

ArthurHub. (2016). Android image cropper. Retrieved from https://github.com/ArthurHub/Android-Image-Cropper

Dolphin Computing. (2016). Playscore lite. Retrieved from https://play.google.com/store/apps/details?id=uk.co.dolphin_com.camrascorelite&hl=en

SnapNPlay. (2015). Snapnplay music demo. Retrieved from https://play.google.com/store/apps/details?id=com.snapnplaydemo.android&hl=en