Computer Vision Library for Western Music Notations

Project Report

Sanka Rasnayaka
110465M
September - 2014

Table of Contents

Contents

Introduction

Abstract

The main target of the development was to develop a computer vision system to detect western music notations. The input to the system would be scanned or digital images of western music notations. These notations will undergo multiple image processing algorithms in order to get them in shape for the computer vision algorithms.
Then the computer vision algorithms will run to detect the vital parts of the western music notation scheme in order to gather enough information to understand the notations.

The library developed then can be used to many applications. The main application demonstrated here is to convert the detected western music notation into the eastern music notation scheme.

Western Music Notation Scheme

The western music notation scheme is very methodical therefore a computer vision system can identify its symbols through various means.

All the notes in a western music notation scheme will be on the staff. This consists of five straight horizontal lines.
All other notations are positioned relative to this staff. The relative position is the method of uniquely identifying which note each symbol represents.

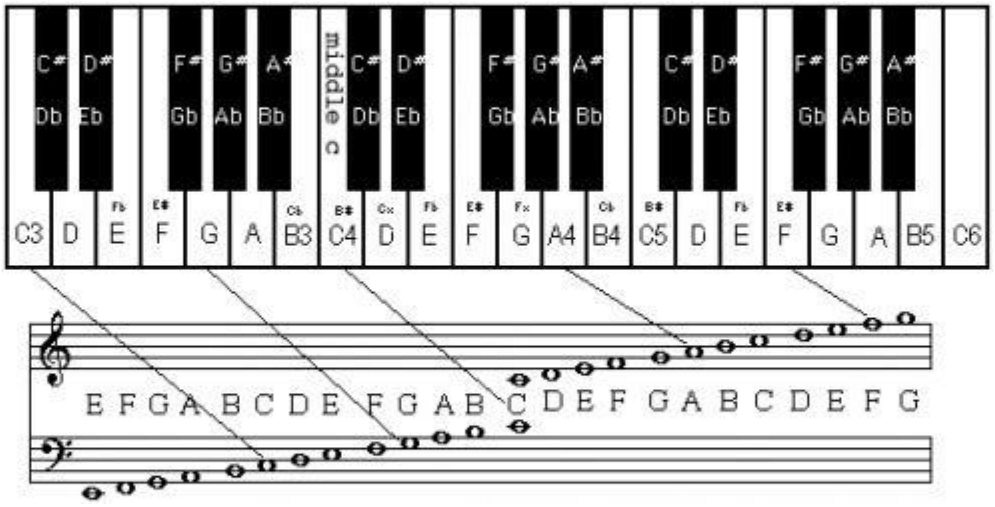

C# Db
D# Eb
F# Gb
G# Ab
A# Bb
middle c
C3 D E F G A B3 C4 D E F G A4 B4 C5 D E F G A B5 C6
Fb E# Cb B# Cx Fb E# Fx Cb B# Fb E#
E F G A B C D E F G A B C D E F G A B C D E F G

Figure 1: Western Music Notation Scheme

The different within the notation scheme is used for the distinction of timing of each notes, this is shown in the below figure

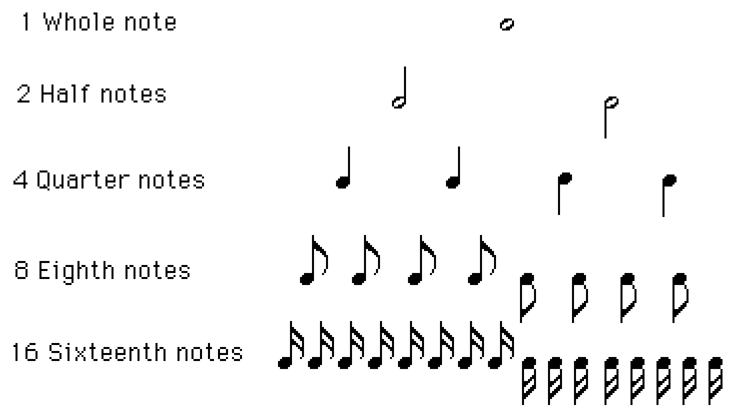
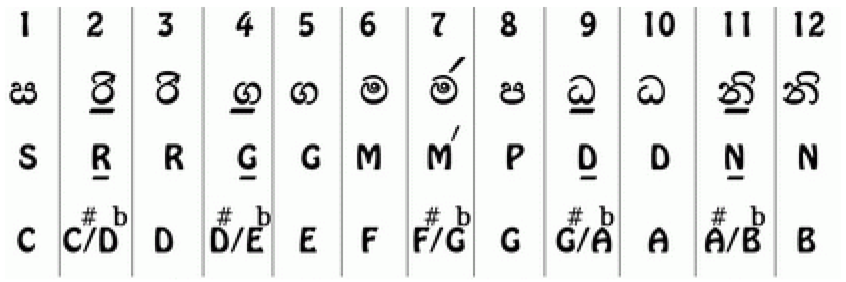


Figure 2: Notation Tree

However for the purpose of detecting western music note and mapping the individual notes to the eastern music notation all the different notes can be taken as same. Only the relative position of the notes head with respect to the staff would be needed for the conversion process.

Eastern Music Notation Scheme

The eastern music notation scheme uses simple characters to denote different notes in an octave. To show higher octaves a dot on top of the note is used. To denote lower octaves a dot at the bottom of the note is used.



1	2	3	4	5	6	7	8	9	10	11	12
ස	$\underline{\text{රි}}$	රි	$\underline{\text{ග}}$	ග	ම	$\acute{\text{ම}}$	ප	$\underline{\text{ධ}}$	ධ	$\underline{\text{නි}}$	නි
S	$\underline{R}$	R	$\underline{G}$	G	M	$\acute{M}$	P	$\underline{D}$	D	$\underline{N}$	N
C	$C^{\#}/D^{b}$	D	$D^{\#}/E^{b}$	E	F	$F^{\#}/G^{b}$	G	$G^{\#}/A^{b}$	A	$A^{\#}/B^{b}$	B

Figure 3: Eastern Music Notation Scheme

A baring system of dividing the notes in to columns and using + and - marks is used to denote the tempo and length of each note. However for the current application that depth of the notation scheme is not required.

Steps in Conversion and Detection

The process followed in detecting and converting the western music notations can be shown using the following diagram.

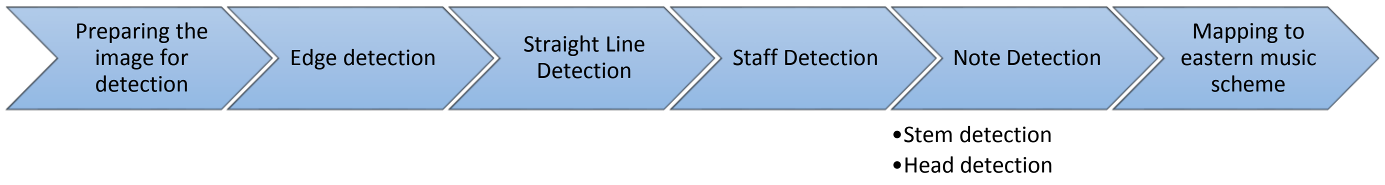


Figure 4: Steps to be followed in application

The output of each process combined with few other parameters are passed as the input to the next process. Therefore each of the processes are done sequentially finally resulting in the mapped notes in eastern music notation scheme.

Literature review

Current Implementations for Western Music Notations Schemes

The first target of the literature review was to identify already implemented computer vision libraries for western music notation detection. However it was evident that there were no computer vision libraries specially catering that requirement. There were many generic image processing and computer vision libraries which are discussed later.

There were many formats and standard schemes which were followed to digitally represent western music notations. Some of them were studied to get an understanding of what the current available technologies are capable of.
Also this review was to ensure that the developed computer vision is capable of generating a digital western music notation from the pictures of notations.

musicXML

MusicXML is a digital sheet music interchange and distribution format. The goal is to create a universal format for common Western music notation, similar to the role that the MP3 format serves for recorded music. The musical information is designed to be usable by notation programs, sequencers and other performance programs, music education programs, and music databases. [1]

This is a potential target for the music notation detection algorithm to finally store the notation sheet after the detection algorithm is run.

The review also revealed some open source software which allow the users to create sheet music with easy interfaces of dragging and dropping notations and combining them together to get the desired sheet music. MuseScore is one such product.

MuseScore

This software allows the user to create and compose western music notation through easy user interfaces. MuseScore is built on the cross-platform QT library and is also available for both Linux and the Mac.[2]

All the existing developments focus on creating western music notations and there were no solutions for extracting data from an existing western music notation in a computer understandable way.
Therefore that was the main target problem which this project tries to address.

Image Processing Libraries

The next phase of the literature review was to try and identify existing libraries for image processing and computer vision. There were many generic libraries and out of them some were more suitable for this application.
When choosing OpenCV with JavaCV the functionality the library provides as well as the customizability of the algorithms to fit the context also was considered. Even with the limited documentation and support available for JavaCV as it is in its early stages, the higher flexibility java provided was the main reason to choose this library for the development.

OpenCV

OpenCV stands for “Open Computer Vision”. As the name implies it is an open source computer vision library with many features. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications.[3] , [4]

JavaCV

JavaCV is a wrapper developed to give OpenCV support for Java programs. JavaCV uses wrappers from the JavaCPP Presets of commonly used libraries by researchers in the field of computer vision and provides utility classes to make their functionality easier to use on the Java platform, including Android. [5]

Edge Detection Algorithms

After careful consideration of the available algorithms some algorithms like edge detection were chosen for self-implementation to have a greater control over its actions. Since the generic algorithms did not suite the intended application.

There were many edge detection algorithms with various pros and cons. The main focus was with first order edge detection algorithms and second order edge detection algorithms.

Since all the western music notations were converted into binary images before processing, the amount of deviations were lower, also it could be safely assumed that edges will have a great colour difference. Therefore the first order edge detection algorithms were sufficient to properly detect edges in this application. [6]

Canny Edge Detection

This is a first order edge detection method, which was easily customizable to fit the needs of this application.
Usual implementation of the canny edge detection algorithm follows the following steps

Step1: Gaussian Mask

A mask matrix is used to average the values of neighbouring pixels before the edge detection is done.
However in our application of the algorithm since the edges are sharp and only two colours are in the image this operation is not required and will adversely affect the outcome if performed. Therefore Gaussian masking was omitted in the implementation to keep the sharp edges as they are.

Step2: Sobel Operator

The Sobel operator is a method to calculate gradients of each pixel in the image using an $n\ x\ n$ matrix. Two values for the two directions x and y should be calculated using appropriate matrices for each.
For the current application a $3x3$ matrix was used as the edges in the images are sharp in their color changes. The $3x3$ matrix values were chosen such that effect of the gradient will reduce as the distance increases. (More details in the implementation)

Step3: Calculating Edge Direction

The gradient of each pixel is calculated using the calculated gradients (Gx and Gy), the following equation is used.

$$\theta = \tan^{-1}\frac{Gx}{Gy}$$

Step4: Tracing directions

The adjacent pixels for each pixel would be traced along the directions and edges will be detected when directions match.
For the current application, angles were reduced to 4 ranges and the tracing were done only using the immediate adjacent pixels.
[7]

Line Detection Algorithms

There are many line detection algorithms which could be used for the application, after researching on several, the Hough transform method was selected as the most appropriate method to detect the straight lines in the western music notations.

Hough Transform for Straight Line Detection

The common problem solved in this method is fitting line segments into few discreet points. However for the given discreet points can be fitted into many line segments as shown in the image below.

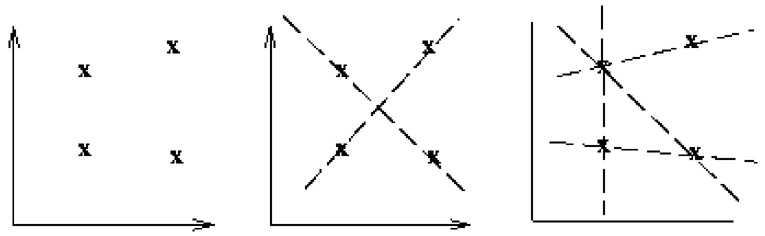


Figure 5: Possible line segments for several points

The Hough transform method will evaluate the possible straight lines and maximise the best fitting lines and give them. The Hough transform implemented in the OpenCV uses this method. [9]

OpenCV implementation of Hough Transform

The OpenCV implementation uses the polar coordinate system in representing the points, this system is illustrated in the image below.

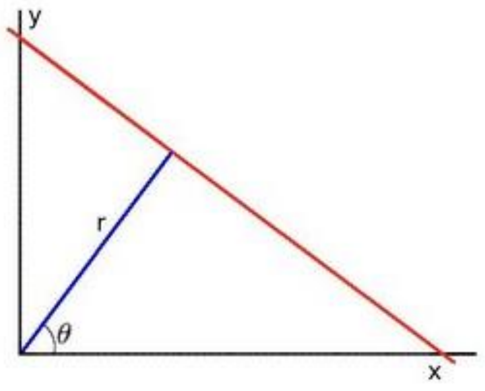


Figure 6: Polar Coordinates

In the given image this is the polar coordinates,

$$y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{r}{\sin\theta}\right)$$

Therefore with this system a general point x_0 , y_0 can be seen as a family of lines through that point which is represented below,

$$r_\theta = x_0\cos\theta + y_0\sin\theta$$

Using this system the Hough Transform in OpenCV keeps track of the intersection between curves of every point in the image. If the number of intersections is above some threshold, then it declares it as a line with the parameters (θ, r_θ) of the intersection point. [10]

System design

Use-Case Diagram

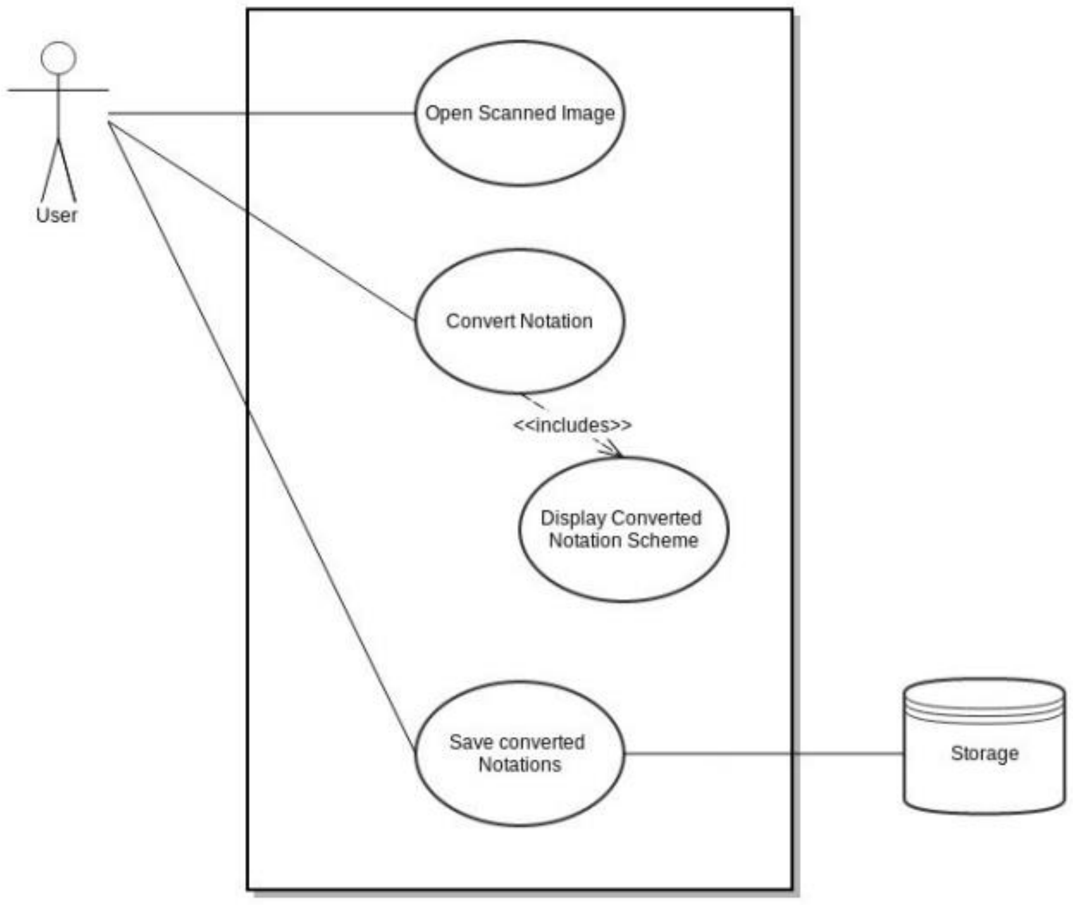


Figure 7: Use-Case Diagram

The basic functionality of the application is described in the above use-case diagram. The actual implementation will be heavily concentrated on the Note Conversation use-case since that is the main focus area.
The other use-cases is used to demonstrate a practical application of the developed library.

Logical View

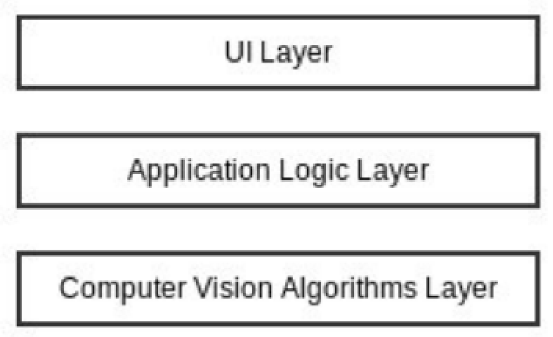


Figure 8: Logical View

As shown above the system can be seen as three layers. The top two layers UI layer and the application logic layer will be for the Music Notation Conversion. However the Computer Vision Algorithms layer will be loosely coupled so that it can be used anywhere as a separately importable library when western music notation detection will be required.

- UI layer will hold all the user interfaces which will give an easy to use and pleasant experience to the users of the Music Notation Converter
- The Application Logic Layer will use the algorithms in the computer vision layer to convert the notations and display them for the user using the UI layer.
- The Computer Vision Algorithms layer will provide all the image processing algorithms required to detect western music notations.

Process View

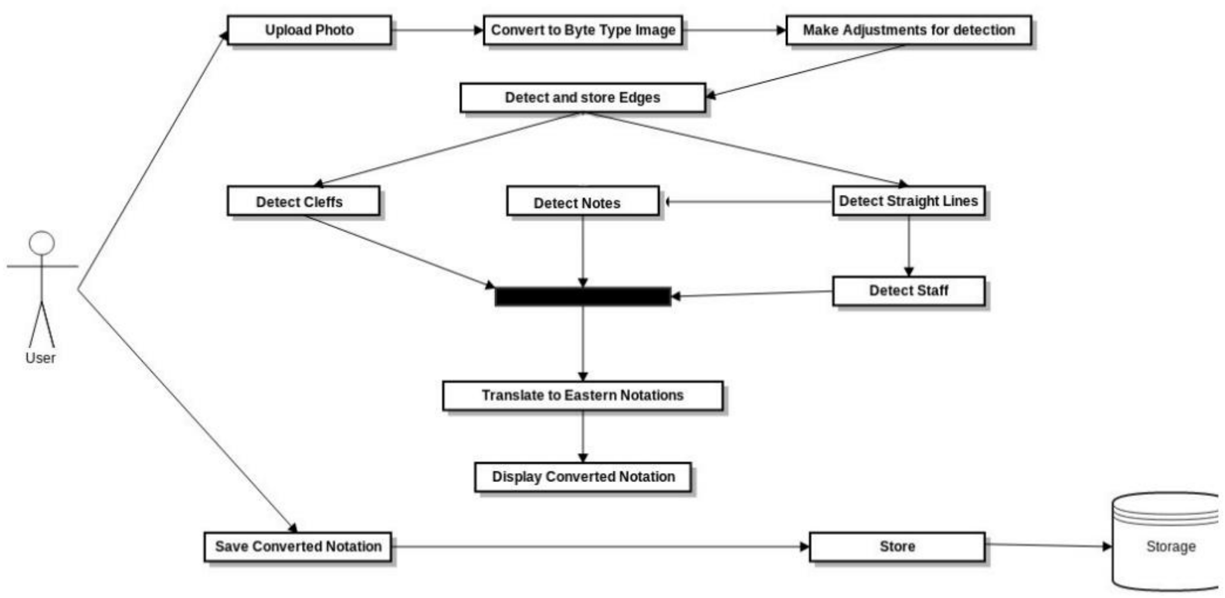


Figure 9: Process View

The initial process diagram was changed slightly as the development progressed. The main change was the note detection being able to use a part of the straight line detection, mainly for detecting the stem of the note.
The detection of the clefs have been omitted from the final implementation, a choice could be given for the user to choose the clef for each staff, it a user does not specify it will be taken as the treble clef for the conversion purposes.

[8]

Implementation details

Algorithm Design and Justification

Getting the image ready

Before the actual computer vision algorithms are run on the image some adjustments might be needed to make sure the image is in proper standard. In order to get this done some editing algorithms have been implemented.

Scale up / down

Scale down is a simple algorithm which is capable of only scaling down by powers of 2, by removing every other pixel row and column, similarly two scale up algorithms have been implemented, first one duplicates pixel values and other one linearly interpolates the pixels.

Smooth

The smooth function is taken from the OpenCV library function cvSmooth.

Convert to Binary Image

All images are required to be converted to binary images before being sent into the computer vision algorithms. This is to ensure there are minimal noise and other adverse effects. This is accomplished by the BufferedImage available in java.

Edge Detection

The edge detection algorithm was implemented separately to make sure it is customized properly to fit the needs of detecting the thin edges which are present in western music notations.
The method followed for edge detection was to use the gradients through x and y axises.
First two masked arrays for x direction and y direction were created. Following two masks were used to calculate each pixel value in arrayGx and arrayGy.

```
int[][] xMask = {{-1, 0, 1},
                 {-2, 0, 2},
                 {-1, 0, 1}};

int[][] yMask = {{1, 2, 1},
                 {0, 0, 0},
                 {-1, -2, -1}};
```

The rationale for using only a 3 x 3 mask was the fact that western music notations consists of very thin lines which need to be detected as edges.

For each pixel in the image the following code segment calculated its Gx and Gy values.

```
for(int i = -1; i <= 1; i++){
      for(int j = -1; j <= 1; j++){
            Gx += (img.getRGB(row + i, col + j) * xMask[i + 1][j + 1]);
            Gy += (img.getRGB(row + i, col + j) * yMask[i + 1][j + 1]);
      }
}
```

Next the strength of the gradient and the angle of the gradient was calculated as follows.

```
edgeGradient[row][col] = Math.sqrt(Math.pow(Gx, 2) + Math.pow(Gy, 2));

if(edgeGradient[row][col] > 25000){
      if (Gx != 0 && Gy != 0){
            k = (float)Gy / (float)Gx;
            myAng = Math.atan(k) * 180;
      }else { myAng = 90; }
}
```

The gradient strength corresponds to the contrast level change, since the edges in western music notations will be moving from white to black or black to white the edge gradient value should be a greater value. The if condition will therefore filter out noise elements,

The calculated myAng value can then be used to mark the image reducing the number of angles as needed.
For the purpose of western music detection, I have selected to identify only four angle lines which are shown in the following figure.

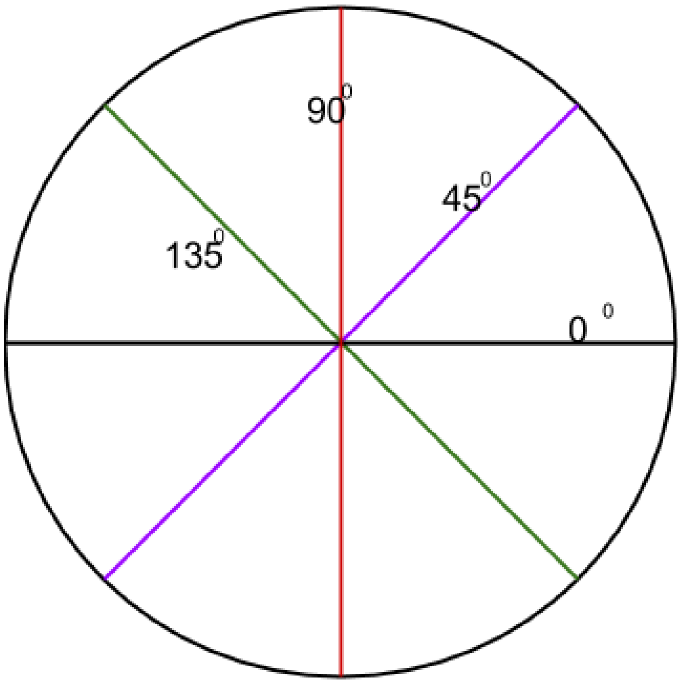


Figure 10: Selected angles for Edge Detection

These angles were filtered from the calculated myAng values, using the following code segment,

```
if(( myAng > -22.5 && myAng < 22.5 ) || (myAng < -157.5 || myAng > 157.5))
      edgeDirections[row][col] = 0;
else if(( myAng > 22.5 && myAng < 67.5 ) || (myAng > -157.5 && myAng < -
112.5))
      edgeDirections[row][col] = 45;
else if(( myAng > 67.5 && myAng < 112.5 ) || (myAng > -112.5 && myAng < -
67.5))
      edgeDirections[row][col] = 90;
else if(( myAng > 112.5 && myAng < 157.5 ) || (myAng > -67.5 && myAng < -
22.5))
      edgeDirections[row][col] = 135;
```

After the gradients are marked using the above method, this 2D array is used to trace through the edges and mark them in the image. At each pixel according to the direction of it the findEdge method will be called to it adjacent pixel in that direction.

```
//Trace along edges in the image
for(int row = 1; row < img.getWidth() - 1; row++){
  for(int col = 1; col < img.getHeight() - 1; col++){
    if(edgeGradient[row][col] > upperB){
      switch (edgeDirections[row][col]){
        case 0:
          findEdge(0,1, row, col, 0, lowerB, image.getWidth(),
image.getHeight());
          break;
        case 45:
          findEdge(1,1, row, col, 45, lowerB, image.getWidth(),
image.getHeight());
          break;
        case 90:
          findEdge(1,0, row, col, 90, lowerB, image.getWidth(),
image.getHeight());
          break;
        case 135:
          findEdge(1,-1, row, col, 135, lowerB, image.getWidth(),
image.getHeight());
          break;
        default:
          temp.setRGB(row, col, Color.YELLOW.getRGB());
          break;
      }
    }
  }
}
```

The findEdge function will simply check to see if the new pixel is in the same direction and if so colour it in a way which is easy to identify visually. Following two images demonstrate the effect of the edge detection algorithm implemented.

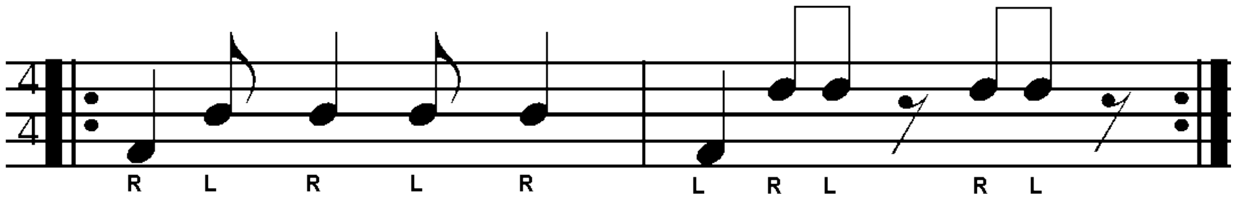


Figure 11: Sample Western Music Notation

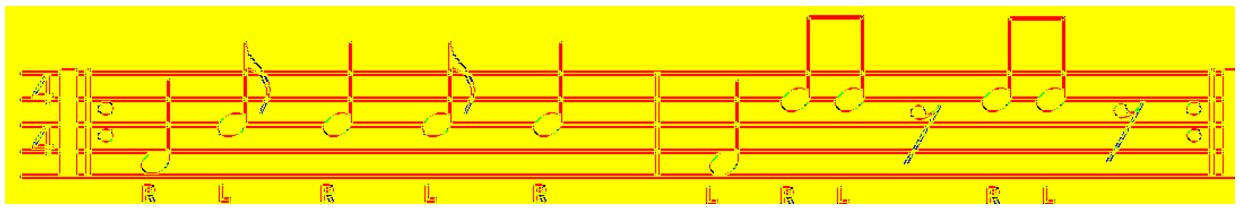


Figure 12: Sample after Edge Detection

Staff Detection

The edge detected image was then passed to a staff detection algorithm. In this algorithm the straight line detection algorithm provided by OpenCV was used. The following code snippet gets all the straight line segments in the edge detected image and stores them in the lines array.

```
lines = cvHoughLines2(dst, storage, CV_HOUGH_PROBABILISTIC, 1, Math.PI / 180,
40, 50, 10);
```

Each line segments x axis projection length is calculated as follows,

```
for(int i = 0; i < lines.total(); i ++){
      Pointer line = cvGetSeqElem(lines, i);
      CvPoint pt1  = new CvPoint(line).position(0);
      CvPoint pt2  = new CvPoint(line).position(1);

      if(pt1.y() > tempY + tol){
            lengths[i] = (pt1.x() - pt2.x()) * (pt1.x() - pt2.x());
      }
      else
            lengths[i] = -1;
            tempY = pt1.y();
}
```

Here the if-condition is to filter out the multiple lines which might get detected for one single line by the OpenCV's line detection algorithm. This is mainly due to the thickness of the lines, some lines are identified more than one time. These lines are filtered by a tolerance value (tol)

There will be lines in the list with various angles, out of these the staff can be detected by taking the horizontal lines. From all the horizontal lines evenly spaced and longest 5 lines were used to detect the staff from the image, the following code segment shows the implementation of the said algorithm,

```
for (int i = 0; i < lines.total(); i++) {
  Pointer line = cvGetSeqElem(lines, i);
  CvPoint pt1  = new CvPoint(line).position(0);
  CvPoint pt2  = new CvPoint(line).position(1);
  boolean okay= true;
  if(pt1.y() > tol && pt1.y()<(in.getHeight() - tol) && pt1.y() == pt2.y()){
  okay = true;
  for(int k = 0; k <= 5; k ++){
      if((pt1.y() - lineY[k]) * (pt1.y() - lineY[k]) < 75){
            okay = false;
            break;
      }
  }
  if(okay){
    cnt ++;
    // draw the segment on the image
    cvLine(colorDst, pt1, pt2, CV_RGB(255, 0, 0), 3, CV_AA, 0);
  }
}
```

Here each line which is accepted is again filtered to make sure 2 lines doesn't get detected for the same line position in the staff. This is achieved through the nested for loop which will iterate through the currently detected lines in the staff.

After the new line is detected a line is drawn on top of the image to visually display the identified 5 lines of the staff.

This method is used to return only the y coordinates of the staff lines since that is the only data required.

After detecting the staff this is the image displayed for the user.

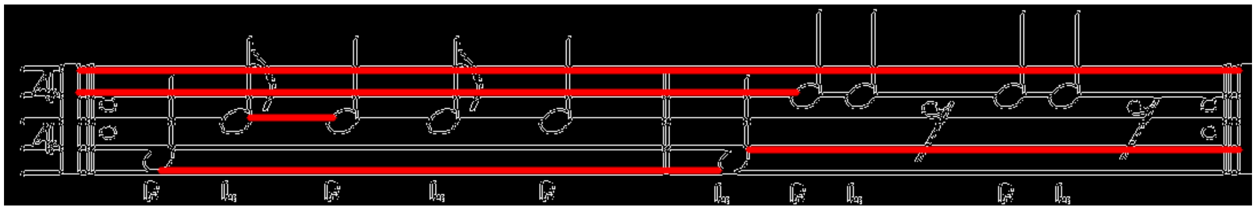


Figure 13: Sample after Staff Detection

Note Detection

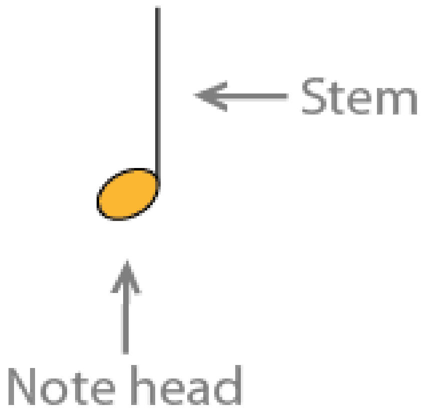


Figure 14: Parts in a Note

For detecting the note the stem of the note is first detected.

Similar to the staff detection the note detection algorithm also makes use of the straight line detection provided by the OpenCv library. The process is similar to staff detection, however here the x coordinates are equalled to select the vertical lines.
The same approach is used to remove the multiple vertical lines detected and other outliers.

The next part of the algorithm is to identify the note head.
For this the approach is to find which position has the most number of black coloured pixels relative to the identified staff, the following diagram illustrates this.

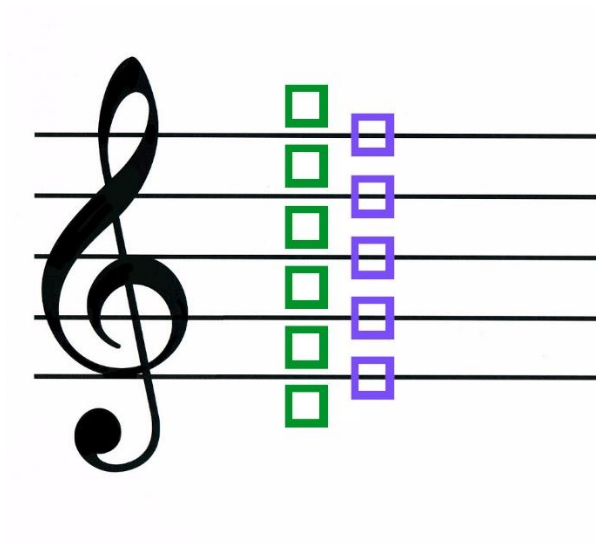


Figure 15: Square areas for note head detection

The green square represent the notes which come in between staff lines and the blue squares represent the notes that come on top of the staff lines.

The following algorithm would calculate the number of black pixels in each of the squares mentioned above, near a stem.
After that the midpoint of that square is taken as the notes position, also that point is marked in the image to make it visually clear for the user.

```
for(int j = 0; j < staffin.length; j ++){
      currY = staffin[j] - staffwidth / 4 ;
      value = 0;
      for(int m = currX; m < (currX + staffwidth / 2); m ++){
            for(int n = currY; n < (currY + staffwidth / 2); n ++){
                  if(m < in.getWidth() && n <in.getHeight() && m>0 && n > 0){
                        Color temp = new Color(backUp.getRGB(m, n));
                        if(temp.getRed() == 0 && temp.getGreen() == 0
                              && temp.getBlue() == 0){
                              value ++;
                        }
                  }
            }
            if(max <= value){
                  pos = (currY+staffwidth/4);
                  max = value;
            }
      }
}
```

This algorithm increases the variable value at each specified location and out of all the positions the max value is taken as the position.

However there can be vertical lines which are identified and which is not a note, to filter these kind of stems the value of the max can be made to be greater than a small value, which is shown in the following code segment.

```
if(max > staffwidth * staffwidth / 8){
    cvLine(colorDst, pt1, pt2, CV_RGB(0, 255, 0), 3, CV_AA, 0);
}
```

Here the max is made sure to be greater than a tolerance value which is proportional to the area of the selected square, that way out lying vertical lines will not cause notes to be detected.

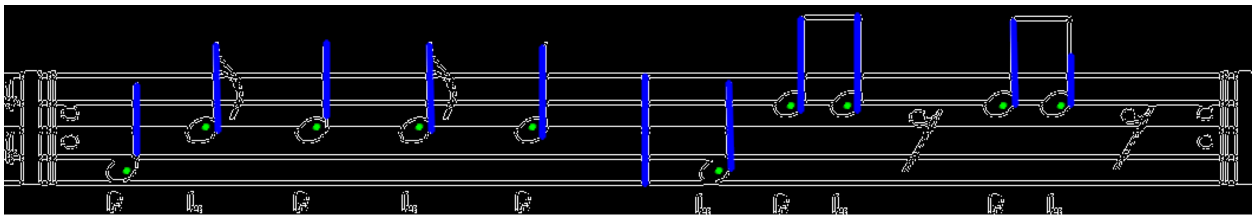


Figure 16: Sample after Note Detection

Here there is a vertical line which is not a note, and it has not been detected as a not due to the last explained filtering process.
The note stems identified are given in blue colour and the note heads are identified in green dots.

Mapping to eastern notes

The mapping to eastern music notation was achieved through a simple Y position comparing algorithm, which would compare the notes Y position with the staff.
First the notes were ordered in the X position order to make sure the mapping shows the notes in the correct order. The following code segment shows how the x position sorting was done.

```
Arrays.sort(notes, new Comparator<int[]>() {
      @Override
      public int compare(int[] o1, int[] o2) {
            return Integer.compare(o1[1], o2[1]);
      }
});
```

Here the variable notes is a 2D array, the first column denotes the y positions and the second column denotes the x positions.

After that the notes y positions are mapped using the EasternMapping.note() function as shown in the below code snippet.

```
for(int i = 0; i < notes.length; i ++){
      if(notes[i][0] != 0)
            ans += EasternMapping.note(staff, notes[i][0]);
}
```

Here the ans is an empty string which will accumulate the converted eastern music notations and later will be printed in a message box for the user to see.

The EasternMapping.note() function will simply check the relative position and according to the mapping scheme discussed in the introduction it will return a string containing the appropriate eastern music note.

Evaluation & Performance

The Western Music Notation detection algorithms were the main focus in evaluating the performance. The main identified evaluation criteria is the accuracy of the results.

The detection algorithm can give un-expected results due to many results such as effects of noise, pixel densities varying from image to image.

Also the time complexity of all the custom algorithms were evaluated to get a measure of performance of the developed computer vision system.

Accuracy of detection algorithm

When considering the accuracy of the algorithm, the variations of the images played a huge role. Therefore the images were broadly categorized into two areas in this application. They are

1. Scanned Images
2. Screen Captured Images

Scanned Images

The scanned images should be properly aligned in order for the vision algorithms to run. The main issue in the scanned images were that they could be in any colour and there is a higher percentage of noise in these images.

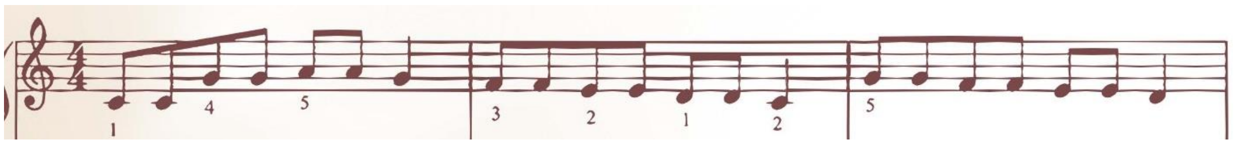


Figure 17: Scanned image of a Western Music Notation Book

The above image is an example for a scanned image from a Western Music notation book. This image has RGB values for each pixel. Therefore the accuracy is effected by the conversion to binary image as well.

Out of the test cases done with images of this nature the observations showed an accuracy of about 50 % ¬ 60 % notes.

Find several images of scanned western music notations that were converted in the appendix.

Screen Captured Images

The images which are captured through a digital image of a western music notation is categorized here. Since these images are created digitally the noise component does not affect the quality of the images. Also since the colours are properly identified it will be properly converted into the binary image.

Below is a sample for a screen captured image.

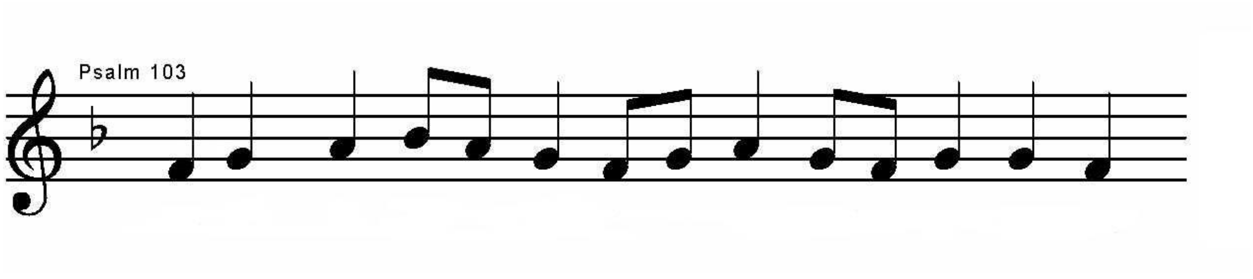


Figure 18: Digital Image of a Western Music Notation

The accuracy rate of such images are in the range of 95% - 100%.

The rate is calculated considering a high quality (pixel density) image of this nature. There are some other issues that would affect the success rate as well.

Find several images of converted digital images of western music notations in the appendix.

Other Factors Affecting Accuracy

There are many other factors which could affect the accuracy of images, some of them are described below.

Pixel Density of image

Images with a low pixel density will cause various errors resulting in wrong answers. One main issue in the low pixel images are the inability to identify the staff properly. When the pixel density is low even with a little amount of noise, the straight line detection will fail to connect a long line together. Therefore the staff will be detected as short line segments.
This will cause the algorithm to choose 5 wrong lines as the staff or to select less than 5 lines.

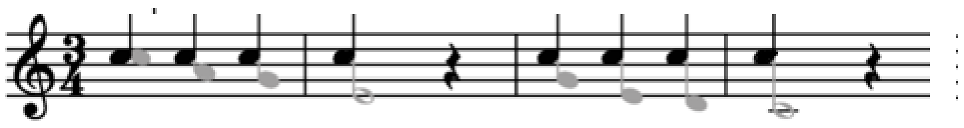


Figure 19: Low pixel density image

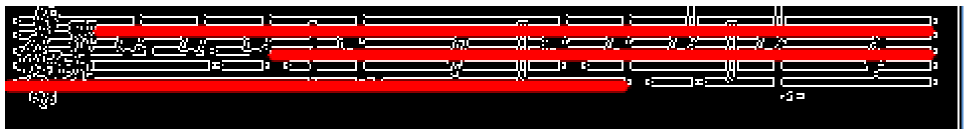


Figure 20: Faulty Staff Detection in low pixel density images

The above to images illustrates a low pixel density image and its result after running the staff detection. The images have been scaled up to make them visible, the original images are much smaller than what is represented here.

Width of stem and staff lines

When the width of the lines in the western music notation is very low it can lead to the line not being detected.
This especially effects the staff detection algorithm and the note detection algorithm (the stem detection part). Out of the many test cases tested there were only few which had very thin lines and caused this issue.

Image clarity and sharpness of edges

The sharpness of the edges also a factor. Some images which had blurry edges and smudged or smoothed edges would also cause the edge detection algorithm to come up with a rounded solution which will not give the proper input for the line detection algorithm.

Time complexity of algorithms

Edge Detection

Staff Detection

Note Detection

Conclusion

References (should be sited in order of reference)

[1] MusicXML Documentation:
http://www.musicxml.com/UserManuals/MusicXML/MusicXML.htm#Tutorial.htm

[2] MuseScore Documentation:
http://musescore.org/

[3] OpenCV Website:
http://opencv.org/

[4] OpenCV Documentation:
http://docs.opencv.org/modules/core/doc/intro.html

[5] JavaCV git hub repo:
https://github.com/bytedeco/javacv

[6] University of Nevada, Reno: Computer Science and Engineering. Edge Detection Notes,
http://www.cse.unr.edu/~bebis/CS791E/Notes/EdgeDetection.pdf

[7] Canny Edge Detection Tutorial: by Bill Green (2002)
http://dasl.mem.drexel.edu/alumni/bGreen/www.pages.drexel.edu/_weg22/can_tut.html

[8] Music Notation Converter Software Architecture Document

[9] Hough Transform: R. Fisher, S. Perkins, A. Walker and E. Wolfart.: 2003
http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm

[10] OpenCV Documentation: Hough Transform
http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html