Optical Music Recognition Application

Grady Low
Yung-Ho Chang
Group 3

30 April 2012

Abstract

The modern day smart phone processor is quickly approaching clock speeds of up to 2GHz while possessing up to 1Gb of RAM. By contrast, the Apollo Guidance Computer from the 1960s which had put man on the moon had a clock speed of a 2.048MHz and 2048 words of RAM [9]. The average person possesses in his/her pocket more processing power than all but the more recent computer desktops built within the past ten years or so. And while optical music recognition has been around since 1966 [2], it has never been able to take advantage of the massive technological resource that resides within the current smartphone. This project aims to develop an application within the iOS mobile operating system to convert written data on a sheet of music into digital data which may be interpreted and played by a computer or piano-playing robot. Computer vision is utilized to process the image and do the recognition process.

Procedure

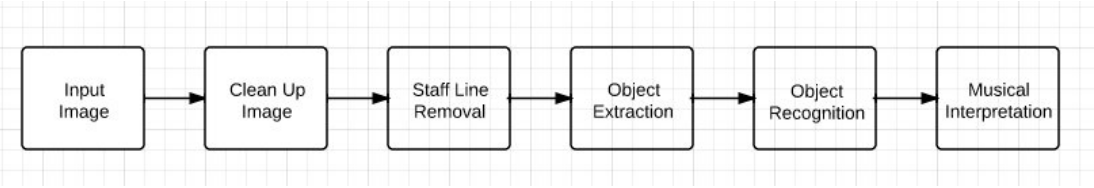

Figure 1: The flow chart shows graphically the process of optical music recognition. The image must go through these steps in order to be fully processed and yield meaningful results.

Almost all forms of optical music recognition since its inception follow the same general structure when it comes to realizing the objective. In short, optical

music recognition must have the following to be considered optical music recognition: Staff line detection and removal, musical object extraction, musical object classification or recognition, and musical object interpretation.

Staff Line Detection

A staff is a set of five horizontal lines which determine the pitch of a note on the sheet music. These lines cover the majority of the sheet music and are absolutely crucial to the interpretation of music. However, staff lines pose a problem to OMR systems because they connect and overlap with almost all the musical objects on a given sheet of music. Therefore to simplify the recognition process the staff lines need to be recognized and removed. To realize this task, the OMR system creates a histogram of the number of black pixels which exist in any given row. Because staff lines are always horizontal (given a perfect image), the locations of staff lines are always given by the highest peaks of the histogram.

It would then seem fairly trivial to simply remove the staff lines after discovering their locations, however this would separate objects into many different pieces and cause far too much segmentation and create many avoidable problems. An algorithm that properly removes the staff lines would scan across the rows of the staff lines, removing the existing black pixel if there are no black pixels above nor below it. The reasoning behind this is that if a specific point has a black pixel either above or below it, it likely is part of a bigger symbol. However, if a specific point does not have a black pixel surrounding it, it can be assumed to just be part of the staff line in empty space.

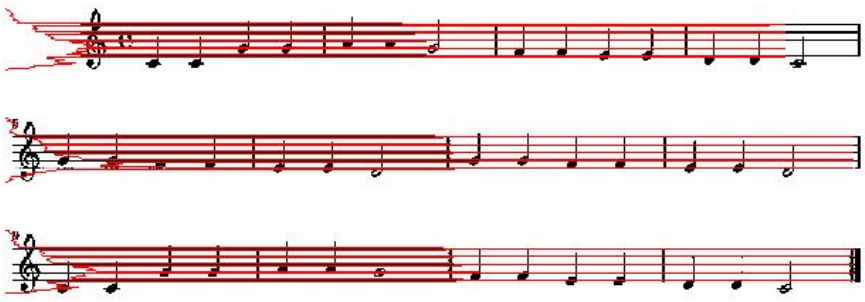

Figure 2: The histogram of black pixels is shown in red. Note that the highest peaks of the histogram correspond to the location of the staff lines and show where the staff lines need to be removed.

Musical Object Extraction

With the staff lines removed, the image then becomes a collection of disconnected symbols and objects. A flood fill algorithm, much like the popular paint bucket function present in most graphic editing software, could then be utilized to extract each object. Every time the algorithm comes across a black pixel, it will run the flood fill algorithm on that specific object, which will cause that object to become white. This algorithm will save the shape of an object as well as the location of the object. The entire extraction process produces a large linked list with all the necessary data needed for the rest of the recognition process. By this step, the sheet music has been completely processed and the result is a large collection of data in form of objects that need to be identified and given meaning.

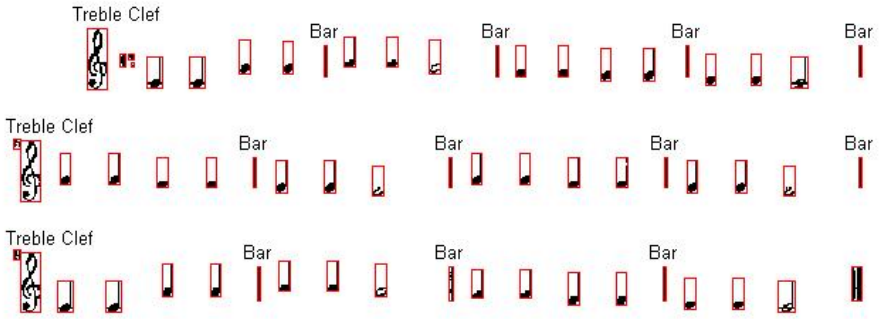

Figure 3: With the staff line removed, the flood fill algorithm is then applied to every object within the page to extract data from each individual object. The flood fill algorithm works by taking one seed pixel and moving away from that seed until it detects the border of the object, then filling everything inside the border with another color. The data is then processed in the next step of the OMR system.

Musical Object Classification

The classification and recognition process makes up the bulk of the OMR system. Certain objects such as the treble clef and bar lines may be recognized simply by normalizing the object with the height of the staff lines and finding the size of the object or by the ratio of its height versus its width. This takes advantage of their unique shapes and is illustrated by Figure 2. However most objects require much more advanced methods to be correctly and confidently classified. Two methods were tested and tried throughout the course of the project.

Correlation

Template matching was utilized by following the method of calculating correlation between a model and the image as shown in [5]. A correlation is defined by the sum of the product of the image pixel values and the model pixel value. If the correlation is high then the model and the image must be very similar. However this method is heavily dependant on scale and so a SURF method was adopted to take advantage of its scale-invariant and rotation-invariant properties so that a more robust and flexible system could be implemented.

SURF

OpenCV's implementation of SURF was utilized. SURF (Speeded Up Robust Features) is an algorithm that creates descriptor vectors by finding the most prominent features of an image and comparing that vector with another via Euclidean distance to see how well they match together. The OpenCV function *cvExtractSURF* automatically extracts good features for this implementation, and *findNaiveNearestNeighbor* can be used as a crude way to measure the Euclidean distance between two features which are being compared. The full intricacies of the code can be seen in the appendix, but the general gist is that OpenCV automatically generates the features we need for the comparisons, and then finding the minimum squared distance between two vectors will lead to determining whether or not the features match up well or not.

Note in Figure 2 that on the upper left portion of the image the *Common Time Signature*has been segmented into three different objects. A technique has been utilized from Bainbridge *et al* [1] regarding this issue and essentially the bounding regions would be extended before template matching is performed.

Musical Object Interpretation

The word interpretation was found to be most fitting because after the OMR system has correctly classified all the objects, by itself it cannot understand how the piece flows together. The last step of OMR is for the system to understand how to play back all the objects it had separated and classified. This involves storing information such as note pitch and beat length within the notes and also checking to see if the number of beats in each individual measure of music adds up according to the time signature and more generally speaking that the rules of music writing are being followed. This step also functions as a way of verifying the results of the recognition process. There is a certain syntax that must be

followed when writing music and by recognizing this fact, the recognition rate may be increased by eliminating certain choices. The system, at this final stage of the OMR process, has processed the given input, extracted the meaningful objects, assigned a meaning to those objects, and is ready to play the recognized music.

Playback

Playback was originally realized by using the AVFoundation framework and AVAudioPlayer class within iOS to play the notes. A sound is specified and a play method is called to output the audio. This is a rather tedious method and does not offer a lot of flexibility in playback, but still served as a relatively easy to implement way to output audio. This method involves playing a sinusoid at certain frequencies which correspond to certain notes. Universally A4 (the A above middle C on the piano keyboard) is defined as 440.0 Hz, with A3 (the A which is located one octave below A4) being 220.0Hz, and so on and so forth. Every eight notes constitutes an octave, and each octave contains 12 semitones of equal size. With this information theoretically all the notes that can be played on the piano can be generated with some symbol algebra involving logarithmic functions, with some interpolations necessary to achieve more exact frequencies. AVAudioPlayer needed to have an audio file in the library already loaded before the method call is made, and the audio tones were all generated in MATLab (see appendix for code).

Frequency f of the nth key is given by: $f(n) = (440)(2^{\frac{n-49}{12}})$

However, the open source SoundBankPlayer (developed by Matthijs Hollemans and licensed under the terms MIT license) is a much more powerful class when it comes to musical playback. It allows for a full 88 note representation without needing 88 individual note files. The SoundBankPlayer can be seen as an middle step between AVAudioPlayer and the ideal playback module, allowing convenient playback.

Experimental Results

Figure 4 shows how the algorithm is negatively effected by low resolution images. Many sample music sheets had to be omitted throughout the development process because their low resolution were causing many problems that were solved upon finding higher quality images.

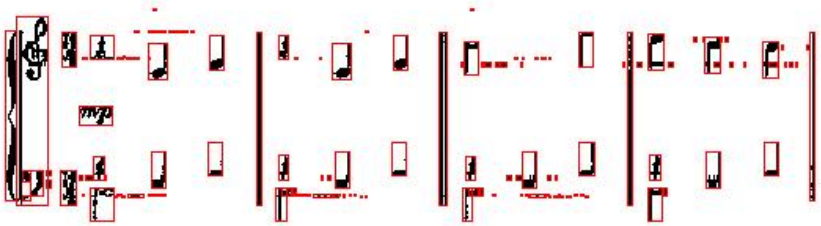

Figure 4: The resolution of the image is 533 x 136. The resolution of the previous images where the algorithms worked better was 912 x 391. When the input image resolution was too low many random artifacts tended to appear on the page and many symbols became skewed. It was experimentally determined that the target image should be at least 900 pixels wide and should not have too many objects on the page to avoid memory problems while still maintaining quality.

Discussion

Difficulties

While it may be rather juvenile, a rather legitimate source of frustration in the project involved learning to use a Macintosh computer. Much time was dedicated to wrestling with unresponsive controls, hotkeys that no longer worked, but most importantly with the development software, Xcode. To develop software for iOS, Apple requires all work to be done in Xcode, using the Objective-C language. The learning curve was rather steep and frustrating at times.

Converting the MATLab code to Objective-C proved to be a large obstacle. MATLab is able to easily deal with large matrices and computations between them, and so many functions and algorithms which were trivial to implement within MATLab did not transfer over easily and caused many deadlines to be missed and ultimately resulted in a considerable loss in efficacy. Moreover, Objective-C is not the ideal platform for image processing, as all the images are resized relative to the display and many outputs are distorted when working within Objective-C. Many features that can be used for image recognition are lost in this process.

Lastly, when dealing with high resolution pictures, there were many times when memory management was an unforeseen complication. This then caused certain code to be recoded and better written or for the pictures to be downsampled so the image sizes would be smaller, at the cost of losing quality and information.

Sources of Error

The project started under the assumption that the image to be read in was taken under ideal situations: a perfectly rectangular sheet scanned at 300 DPI with very high contrast and no extra noise. Yet reality is not so kind as to submit to our assumptions and thus our sources of error came from anything that deviated from our original assumption. If the paper was slightly rotated by even a few degrees, the staff line detection histogram would be completely useless. If the image had far too many shades of grey instead of binary black and white further issues would come up. If there were random specks in the paper they would be analysed, categorized, and add confusion to the outcome. The list goes on and on, but hope is not all lost. There are plans for future work and extensions which would provide solutions to these problems.

Future Work and Extension

Rectification

Since the project's beginning there had been plans of implementing computer vision in the form of image rectification in the project. The transformation required in rectification is obtained through 8 separate variables. These 8 variables can be obtained by 4 distinct points, which each contribute 2 variables. This was extremely easy to implement with MATLab functions *maketform* and *imtransform* (See appendix for code). Essentially the code is solving a system of eight linear equations to approximate eight values to get the transformation needed. This feature has not been implemented within the iOS application due to it being too time intensive to convert the code to Objective-C.

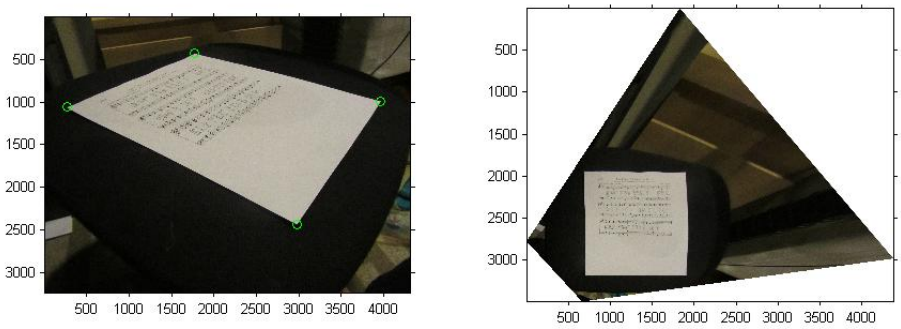

Figure 5: The four corners (selected in green) were chosen as points to estimate the homography. The result gives an image with the original sheet music, but now rendered horizontally and generally much more readable upon zooming in.

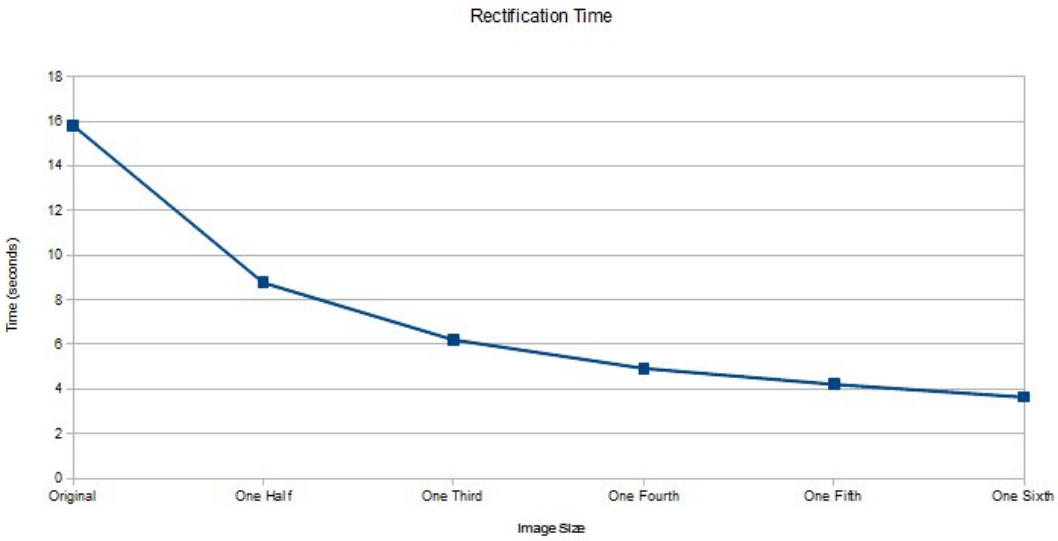

Figure 6: The performance of the rectification function, measured by how long the function took in seconds to run versus the size of the image. The original image size is 4320 x 3240, and subsequent downsampling does little to improve the efficiency of the algorithm after the fourth iteration, as it seems to converge to a value of approximately three seconds.

Refined Image Processing

There exist plans to improve the robustness of the program to combat the aforementioned sources of error. Care needs to be taken, as the smallest of specks can actually be a part of written music. For example, a small dot to the right of a note increases the note length by half the original value, whereas a small dot directly above or below the note actually decreases the note length, and a small dot could easily be mistaken as noise by most filtering processes. It seemed to make sense to include another image processing step after the recognition and analysis portion of the program has been fully realized, as it should be easier to distinguish between necessary and unnecessary information after the program can guess what should or should not exist in the page.

Editable Music

Lastly the project has an end goal of achieving an editable sheet of music. This feature deviates from the computer vision-centered focus of the project but is a necessary feature of any music software. One of the main advantages of digitizing music would be to be able to transpose the piece into a different key, as this is a highly tedious process that could be easily done by a computer. Moreover

being able to change parameters such as tempo and instrument voices will also be desirable features in any sort of musical application.

Synthesis Tool-kit for Playback

The playback portion of the project was originally planned to be realized by utilizing the Synthesis toolkit Instrument Network Interface (SKINI). SKINI was selected because it was designed to be MIDI compatible and accepts text as input, while fully allowing the usage of many different instruments with many easily variable parameters, such as dynamics and tempo.. The OMR application would generate as an output MIDI note numbers (a number between 0 - 127 which corresponds to a specific pitch), the note length (an eighth note, quarter note, half note, etc.), and a tempo (which is fixed at 96 beats per minute for simplicity's sake unless otherwise specified) which would all be accepted and outputted in any instrument and intensity desired. However, this feature did not manage to make it into the project time frame but is expected to be incorporated in the near future.

Current Trends in Robotics

Robotics has come a long way in the recent years. From commercial applications such as an automated mobile vacuum cleaner to industrial advances in manufacturing to even military usage, the rate of growth for robotics has been nothing short of amazing. Newer and newer technologies are always being tested and developed to further advance the field of robotics and computer vision.

An example of particularly impressive applications of robotics lies in the field of medical surgery . Minimally invasive surgery (MIS) is contrasted with open surgery, opting to insert instruments into the patient's body through a small incision to perform the necessary operation. The advantages of MIS include faster recovery, less physical trauma, more potential for teleoperation [4].

Surgeons are now able to utilize technology even more through the advances of robotics and computer vision to aid in their MIS operations. The Wireless Hands-Free Surgical Pointer (WHaSP) for MIS is a system developed which clearly demonstrates the usefulness of computer vision in the medical field.

During a minimally invasive surgical operation, both the surgeon instructor and the assistant trainee have to utilize both hands to operate surgical instruments. Furthermore, each person often has their own display monitor which displays vital information from their instruments during the surgery. If both surgeons were both experts, there would be no problem. However, for pedagogical purposes,

this is far from ideal. To aid in the training of MIS surgeons, the WHaSP system was developed as a way for the instructor to point to multiple video monitors without removing his/her hands from the instruments [8]. This system would need to combine and utilize data from the headset trackers and the displays used within the operating room to benefit the users. By tracking the headsets worn by the instructor surgeon, a new way is opened for training newer surgeons in the field of MIS.

In addition to research such as the WHaSP system, much work is being done for the tracking of intrabody instruments. Current methods such as optical tracking and electromagnetic tracking respectively require line of sight and lack of interference from the many metallic instruments within the operating room. Therefore an idea was proposed which would combine multiple sensors and data into one integrated tracking system to enhance the current method of intrabody instrument tracking [4].

As impressive as MIS is, it is not the end all be all of medical surgery advancement. MIS removes all hand eye coordination due to the surgeons no longer being able to see the physical points of contact and also the surgeons are deprived of force feedback from their instruments. To further complicate things, many instruments used in MIS have very limited degrees of freedom and range of motion due to how they have to fit through a very small incision to enter the body. To address this issue, researchers are working on designing miniature modular *in vivo* robots to increase the efficiency of MIS [7]. The complete system composes of three separate modules: a camera with 2 degrees of freedom (DoFs), a retraction robot with 2 DoFs, and a manipulator with 6 DoFs. While the modules themselves has not be fully manufactured, initial research shows that the theory behind this concept is sound and the camera which has been manufactured does indeed work as intended.

MIS suffers from another condition, but this condition is extremely common in the world. Surgeons working in MIS suffer fatigue, much like many other humans doing any sort of physical labor. Robotics replacing humans in doing repetitive manual tasks has been around as long as the concept of robotics has been, but to replace something as complicated as MIS is truly an impressive feat. By combining position and imaging, researchers are able to keep the error of a ceiling mounted robotic arm under 1.3mm [6]. This process of combining data and utilizing computer vision (including the manipulation of various rotational matrices) allows for up to four ceiling mounted mechanical arms to perform complicated processes in an automated fashion.

Lastly, exciting work is being done in the field of microsurgery, as well as the field of MIS. Microsurgery is simply defined as surgery requiring a microscope,

which in turn indicates a need for extreme precision. Some microsurgery operations requires manipulation of structures as small as 10 micrometers (such as the thin retinal internal limiting membrane), but normal human surgeons typically do not have precision needed, having involuntary hand motion that ranges to several hundred micrometers [3]. The Micron handset aims to improve precision by employing the same method of image stabilization in a camera and actuates its own tip to balance the involuntary human hand movements. This handset would be able to differentiate between the most minute hand movements and give unprecedented control to surgeons operating in delicate operations.

Medical practices has progressed so much in recent times directly due to the advances in robotics and computer vision. To think that there are currently devices that can help a surgeon manipulate the smallest of structures would simply be unfathomable even a few years ago. Without a doubt, robotics and computer vision will continue to augment the capabilities of our doctors and surgeons in the future, and contribute to the saving of countless lives.

References

[1] D. Bainbridge and T.C. Bell. Dealing with superimposed objects in optical music recognition. In *Image Processing and Its Applications, 1997., Sixth International Conference on*, volume 2, pages 756 –760 vol.2, jul 1997.

[2] Michael Kassler. Optical character-recognition of printed music: A review of two dissertations. *Perspectives of New Music*, 11(1):pp. 250–254, 1972.

[3] R.A. MacLachlan, B.C. Becker, J. Cuevas Tabare ands, G.W. Podnar, L.A. Lobes, and C.N. Riviere. Micron: An actively stabilized handheld tool for microsurgery. *Robotics, IEEE Transactions on*, 28(1):195 –212, feb. 2012.

[4] Hongliang Ren, D. Rank, M. Merdes, J. Stallkamp, and P. Kazanzides. Multisensor data fusion in an integrated tracking system for endoscopic surgery. *Information Technology in Biomedicine, IEEE Transactions on*, 16(1):106 – 111, jan. 2012.

[5] Florence Rossant. A global method for music symbol recognition in typeset music sheets. *Pattern Recognition Letters*, 23(10):1129 – 1141, 2002.

[6] C. Staub, A. Knoll, T. Osa, and R. Bauernschmitt. Autonomous high precision positioning of surgical instruments in robot-assisted minimally invasive surgery under visual guidance. In *Autonomic and Autonomous Systems (ICAS), 2010 Sixth International Conference on*, pages 64 –69, march 2010.

[7] G. Tortora, A. Dimitracopoulos, P. Valdastri, A. Menciassi, and P. Dario. Design of miniature modular in vivo robots for dedicated tasks in minimally invasive surgery. In *Advanced Intelligent Mechatronics (AIM), 2011 IEEE/ASME International Conference on*, pages 327 –332, july 2011.

[8] C. D. W. Ward, A. L. Trejos, M. D. Naish, R. V. Patel, and C. M. Schlachta. The whasp: A wireless hands-free surgical pointer for minimally invasive surgery. *Mechatronics, IEEE/ASME Transactions on*, PP(99):1 –9, 2012.

[9] M. Williamson. Aiming for the moon: the engineering challenge of apollo. *Engineering Science and Education Journal*, 11(5):164 – 172, oct 2002.

Appendix

MATLab Rectification Code

```
I = imread('IMG_0004.jpg');
% I = downsample(I,2) % downsamples the image and reduces size
% have to redo ginput after downsampling
tform = maketform('projective',[273 1059; 1773 423; 3963 987; 2979 2445],...
                               [0 0; 1 0; 1 1; 0 1]);
% the coordinates are obtained by using ginput and clicking on the corners
% of the sheet. Maketform then maps those points to corners on a square of
% length 1.
[B] = imtransform(I,tform,'bicubic','size',size(I)+[250 50 0],'fill',255);
figure, subplot(1,2,1), imshow(I), axis on
subplot(1,2,2), imshow(B), axis on
```

MATLab Tone Generation Code

```
Fs = 1000;      % Samples per second
% Remember that samples needs to be double frequency for Nyquist
toneFreq = 440;  % Tone frequency, in Hertz
nSeconds = 1;   % Duration of the sound
y = sin(linspace(0,nSeconds*toneFreq*2*pi,round(nSeconds*Fs)));
%sound(y,Fs);  % Play sound at sampling rate Fs
wavwrite(y,Fs,8,'440.wav');  % Save as an 8-bit, 1 kHz signal
```

Xcode ApplicationDelegate.m

```
#import "ApplicationDelegate.h"
#import "MainViewController.h"
```

```
@implementation ApplicationDelegate

@synthesize window;
@synthesize viewController;

#pragma mark -
#pragma mark Application lifecycle

/*!
 * @discussion  Override point for customization after application launch.
 */
- ( BOOL )application: ( UIApplication * )application didFinishLaunchingWithOptions: (
{
    ( void )application;
    ( void )launchOptions;

    [ window addSubview: viewController.view ];
    [ window makeKeyAndVisible ];

    return YES;
}

/*!
 * @discussion  Sent when the application is about to move from active to
 *              inactive state. This can occur for certain types of temporary
 *              interruptions (such as an incoming phone call or SMS message)
 *              or when the user quits the application and it begins the
 *              transition to the background state.
 *              Use this method to pause ongoing tasks, disable timers, and
 *              throttle down OpenGL ES frame rates. Games should use this
 *              method to pause the game.
 */
- ( void )applicationWillResignActive: ( UIApplication * )application
{
    ( void )application;
}

/*!
 * @discussion  Use this method to release shared resources, save user data,
 *              invalidate timers, and store enough application state
 *              information to restore your application to its current state
 *              in case it is terminated later.
 *              If your application supports background execution, called
```

```
 *              instead of applicationWillTerminate: when the user quits.
 */
- ( void )applicationDidEnterBackground: ( UIApplication * )application
{
    ( void )application;
}

/*!
 * @discussion  Called as part of  transition from the background to the
 *              inactive state: here you can undo many of the changes made on
 *              entering the background.
 */
- ( void )applicationWillEnterForeground: ( UIApplication * )application
{
    ( void )application;
}

/*!
 * @discussion  Restart any tasks that were paused (or not yet started) while
 *              the application was inactive. If the application was previously
 *              in the background, optionally refresh the user interface.
 */
- ( void )applicationDidBecomeActive: ( UIApplication * )application
{
    ( void )application;
}

/*!
 * @discussion  Called when the application is about to terminate.
 *              See also applicationDidEnterBackground:.
 */
- ( void )applicationWillTerminate: ( UIApplication * )application
{
    ( void )application;
}

#pragma mark -
#pragma mark Memory management

/*!
 * @discussion  Free up as much memory as possible by purging cached data
 *              objects that can be recreated (or reloaded from disk) later.
 */
```

```
- ( void )applicationDidReceiveMemoryWarning: ( UIApplication * )application
{
    ( void )application;
}

/*!
 * @discussion  Called when the object is freed by the Objective-C runtime
 */
- ( void )dealloc
{
    [ viewController release ];
    [ window release ];
    [ super dealloc ];
}

@end
```

XCode fastmatch.m

```
//
//  fastmatch.m
//  OpenCV-iPhone
//
//  Created by Yung-Ho  Chang on 4/9/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//

#import "fastmatch.h"
#include "MyCvPoint.h"
@implementation fastmatch
-(Boolean) FastMatchTemplate:(const IplImage *) source: (const IplImage*) target: (NSM
    BOOL findMultipleTargets = true;
    int matchPercentage = 70;
    int numMax = 4;
    int numDownPyrs = 4;
    int searchExpansion =15;
    IplImage *target_temp, *source_temp;
    printf("larger source.width=%d source.height=%d target.width=%d target.height=%d\n
    //resize source if target is larger
    if( (source->width<target->width) ||(source->height<target->height)){
        printf("a\n");
        int target_width = target->width, target_height = target->height;
```

```
    while((source->width<target_width) ||(source->height<target_height)){
        target_height = target_height/2;
        target_width = target_width /2;
    }
    target_temp = cvCreateImage(cvSize(target_width, target_height), IPL_DEPTH_8U,
    cvResize(target, target_temp, CV_INTER_LINEAR);
}
else{
    target_temp = cvCloneImage(target);
}
source_temp = cvCloneImage(source);


printf("resize over\n");
if(source->depth != target_temp->depth || source->nChannels != target_temp->nChann
    target_temp = cvCreateImage(cvGetSize(target), IPL_DEPTH_8U, 1);
    cvCvtColor(target, target_temp, CV_BGR2GRAY);
    cvThreshold(target_temp, target_temp, 128, 255, CV_THRESH_BINARY|CV_THRESH_OTS
}

CvSize sourceSize = cvGetSize(source_temp);
CvSize targetSize = cvGetSize(target_temp);

int depth = source->depth;
int numChannels =  source->nChannels;


//Create copies of images so we can modify
IplImage* sourceCopy = cvCloneImage(source_temp);
IplImage* targetCopy = cvCloneImage(target_temp);

//Down pyramid
for(int count = 0; count< numDownPyrs; count++){
    //soruce image part
    sourceSize.width /=2;
    sourceSize.height/=2;

    IplImage* smallSource=NULL;
    smallSource = cvCreateImage(sourceSize, depth, numChannels);
    cvPyrDown(sourceCopy, smallSource, CV_GAUSSIAN_5x5);

    cvReleaseImage(&sourceCopy);
    sourceCopy = cvCloneImage(smallSource);
```

```
    cvReleaseImage(&smallSource);

    targetSize.width/=2;
    targetSize.height/=2;

    IplImage* smallTarget =NULL;
    smallTarget = cvCreateImage(targetSize, depth, numChannels);
    cvPyrDown(targetCopy, smallTarget, CV_GAUSSIAN_5x5);

    cvReleaseImage(&targetCopy);
    targetCopy = cvCloneImage(smallTarget);
    cvReleaseImage(&smallTarget);
}
//perfrom the match on shrunken images
CvSize smallTargetSize = cvGetSize(targetCopy);
CvSize smallSourceSize = cvGetSize(sourceCopy);

CvSize resultSize;
resultSize.width= smallSourceSize.width-smallTargetSize.width+1;
resultSize.height=smallSourceSize.height-smallTargetSize.height+1;

IplImage *result = cvCreateImage(resultSize, IPL_DEPTH_32F, 1);

cvMatchTemplate(sourceCopy, targetCopy, result, CV_TM_CCOEFF_NORMED);

//release memory
cvReleaseImage(&sourceCopy);
cvReleaseImage(&targetCopy);

//find top match locations
NSMutableArray *locations = [[NSMutableArray alloc] initWithCapacity:numMax];
//CvPoint *locations[numMax];

printf("location\n");
[self MultipleMaxLoc: result  :locations :numMax];

cvReleaseImage(&result);

//search the large image at return locations
sourceSize = cvGetSize(source_temp);
targetSize = cvGetSize(target_temp);

printf("size = %d\n", [locations count]);
```

```
//create a copy of source in order to adjust ROI for search
IplImage *searchImage = cvCloneImage(source_temp);
for(int currMax =0; currMax<numMax; currMax++){
    //transform point ot its coressponding point in larger image
    MyCvPoint *temp = [locations objectAtIndex:currMax];
    //CvPoint *temp = locations[currMax];
    int temp_x = temp.getX*(int)pow(2, numDownPyrs);
    int temp_y = temp.getY*(int)pow(2, numDownPyrs);
    temp_x+= targetSize.width/2;
    temp_y+= targetSize.height/2;
    [temp setX:temp_x];
    [temp setY:temp_y];
    [locations replaceObjectAtIndex:currMax withObject:temp];
    printf("location area\n");
    const MyCvPoint *searchPoint = [locations objectAtIndex:currMax];

    //if we are searching for multiple targets and we have found a target or multi
   if([foundPointList count] != 0 && findMultipleTargets)
   {
       Boolean thisTargetFound = false;

       int numPoint = [foundPointList count];
       for(int currPoint = 0; currPoint<numPoint; currPoint++)
       {
           const MyCvPoint *foundPoint = [foundPointList objectAtIndex:currPoint];
           if(abs(searchPoint.getX-foundPoint.getX)<=searchExpansion*2 &&abs(searc
               thisTargetFound=true;
               break;
           }

       }
       //if current targe is found, continue to next point
       if(thisTargetFound)
       {
           continue;
       }
   }
    printf("location2\n");
    //Set source image's ROI to slightly larger than target image, center at curre
    CvRect searchRoi;
    searchRoi.x = searchPoint.getX-(target_temp->width)/2-searchExpansion;
    searchRoi.y = searchPoint.getY-(target_temp->height)/2-searchExpansion;
    searchRoi.width=target_temp->width+searchExpansion*2;
```

```
searchRoi.height=target_temp->height+searchExpansion*2;

//make sure ROI doesn't extend outside
if(searchRoi.x<0){
    searchRoi.x=0;
}
if(searchRoi.y<0){
    searchRoi.y=0;
}
if((searchRoi.x+searchRoi.width)>(sourceSize.width-1)){
    int numPixelsOver=(searchRoi.x+searchRoi.width)-(sourceSize.width-1);
    searchRoi.width-=numPixelsOver;
}
if((searchRoi.y+searchRoi.height)>(sourceSize.height-1)){
    int numPixelsOver=(searchRoi.y+searchRoi.height)-(sourceSize.height-1);
    searchRoi.height-=numPixelsOver;
}

cvSetImageROI(searchImage, searchRoi);

printf("search larger image\n");
//perform the search on larger images
resultSize.width = searchRoi.width-target_temp->width+1;
resultSize.height= searchRoi.height-target_temp->height+1;

result = cvCreateImage(resultSize, IPL_DEPTH_32F, 1);
printf("cvmatchtemplate start\n");
cvMatchTemplate(searchImage, target_temp, result, CV_TM_CCOEFF_NORMED);
printf("cvmatchtemplate end\n");
cvResetImageROI(searchImage);

//find the best match location
double minValue, maxValue;
CvPoint minLoc, maxLoc;
cvMinMaxLoc(result, &minValue, &maxValue, &minLoc, &maxLoc, NULL);
maxValue*=100;


//transform point back to original image
maxLoc.x+= searchRoi.x + target_temp->width/2;
maxLoc.y+= searchRoi.y+target_temp->height/2;

cvReleaseImage(&result);
```

```
        printf("matchPercentage area\n");
        if(maxValue>=matchPercentage){
            //add point to the list
            MyCvPoint *temp2;
            temp2 = [[MyCvPoint alloc] init:maxLoc.x :maxLoc.y];
            [foundPointList addObject:temp2];
            NSNumber *num = [NSNumber numberWithDouble:maxValue];
            [confideenceList addObject:num];

            if(! findMultipleTargets){
                break;
            }
        }

    }

    if([foundPointList count]==0){
        printf("\nTarget was not found to reuqired confidence of %d.\n",matchPercentag
        [locations release];
        cvReleaseImage(&searchImage);
        cvReleaseImage(&target_temp);
        cvReleaseImage(&source_temp);
        return false;
    }
    [locations release];
    cvReleaseImage(&searchImage);
    cvReleaseImage(&target_temp);
    cvReleaseImage(&source_temp);
    return true;
}

-(void) MultipleMaxLoc:(const IplImage*) image: (NSMutableArray*) locations:(int) numM
    //locations = [[NSMutableArray alloc] initWithCapacity:numMax];
    printf("Multi start\n");
    for(int count=0; count<numMax; count++){
        MyCvPoint *start_temp;
        start_temp = [[MyCvPoint alloc] init:0 :0];
        [locations addObject:start_temp];
    }

    //create array for track maxima
    double maxima[numMax];
    for(int i=0; i<numMax; i++){
```

```
    maxima[i] = 0.0;
}

//extract raw data for analysis
float* data;
int step;
CvSize size;

cvGetRawData(image, (uchar**)&data, &step, &size);


step/= sizeof(data[0]);

printf("getData end\n");
for(int y=0; y<size.height; y++, data= data+step)
{
    //printf("y\n");
    for(int x=0; x<size.width;x++)
    {
        //printf("x\n");
        for(int j=0; j<numMax; j++){
            //printf("j data[x]=%f maxima[j]=%f\n", data[x], maxima[j]);

            if(data[x] > maxima[j]){

                for (int k=numMax-1; k>j; k--) {
                    printf("k=%d, k-1=%d\n", k, k-1);
                    maxima[k]=maxima[k-1];

                    [locations exchangeObjectAtIndex:k withObjectAtIndex:k-1];
                }

                //insert value
                maxima[j] = (double) data[x];
                MyCvPoint *temp;
                temp = [[MyCvPoint alloc] init:x :y];

                [locations replaceObjectAtIndex:j withObject:temp];
                printf("ha\n");
                break;
            }
        }
    }
```

```
    }
    printf("end\n");
}




@end
```

XCode MainViewController.m

```
#import "MainViewController.h"
#import <opencv/cv.h>
#import "note.h"
#import "fastmatch.h"
#import "MyCvSurfPoint.h"
#import "MyCvSurfPoint2.h"
#import "Myfloat.h"
#import "MyIpl.h"
#import "MySheet.h"
#import <QuartzCore/CABase.h>

@implementation MainViewController
@synthesize theimageView, choosePhoto, takePhoto, test;
//Music playback function

#pragma mark -
#pragma mark OpenCV Support Methods

- (IplImage *)CreateIplImageFromUIImage:(UIImage *)image {
CGImageRef imageRef = image.CGImage;

CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();
IplImage *iplimage = cvCreateImage(cvSize(image.size.width, image.size.height), IPL_DE
CGContextRef contextRef = CGBitmapContextCreate(iplimage->imageData, iplimage->width,
iplimage->depth, iplimage->widthStep,
colorSpace, kCGImageAlphaPremultipliedLast|kCGBitmapByteOrderDefault);
CGContextDrawImage(contextRef, CGRectMake(0, 0, image.size.width, image.size.height),
CGContextRelease(contextRef);
CGColorSpaceRelease(colorSpace);

IplImage *ret = cvCreateImage(cvGetSize(iplimage), IPL_DEPTH_8U, 3);
```

```
cvCvtColor(iplimage, ret, CV_RGBA2BGR);
cvReleaseImage(&iplimage);

return ret;
}

// NOTE You should convert color mode as RGB before passing to this function
- (UIImage *)UIImageFromIplImage:(IplImage *)image {
NSLog(@"IplImage (%d, %d) %d bits by %d channels, %d bytes/row %s", image->width, imag

CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();
NSData *data = [NSData dataWithBytes:image->imageData length:image->imageSize];
CGDataProviderRef provider = CGDataProviderCreateWithCFData((CFDataRef)data);
CGImageRef imageRef = CGImageCreate(image->width, image->height,
image->depth, image->depth * image->nChannels, image->widthStep,
colorSpace, kCGImageAlphaNone|kCGBitmapByteOrderDefault,
provider, NULL, false, kCGRenderingIntentDefault);
UIImage *ret = [UIImage imageWithCGImage:imageRef];
CGImageRelease(imageRef);
CGDataProviderRelease(provider);
CGColorSpaceRelease(colorSpace);
return ret;
}
-(IplImage *) CreateIplImageFromUIImageGray: (UIImage *) input_image{
    IplImage *temp = [self CreateIplImageFromUIImage:input_image];
    IplImage *result = cvCreateImage(cvGetSize(temp), IPL_DEPTH_8U, 1);
    cvCvtColor(temp, result, CV_BGR2GRAY);
    cvThreshold(result, result, 90, 255, CV_THRESH_BINARY | CV_THRESH_OTSU);
    cvReleaseImage(&temp);
    return result;
}

//init
- (void)playArpeggioWithNotes:(NSMutableArray*)notes delay:(double)delay
{
if (!playingArpeggio)
{
playingArpeggio = YES;
arpegioNotes = [notes retain];
arpeggioIndex = 0;
arpeggioDelay = delay;
arpeggioStartTime = CACurrentMediaTime();
}
```

```
}

- (void)startTimer
{
timer = [NSTimer scheduledTimerWithTimeInterval: 0.05  // 50 ms
 target: self
   selector: @selector(handleTimer:)
   userInfo: nil
repeats: YES];
}

- (void)stopTimer
{
if (timer != nil && [timer isValid])
{
[timer invalidate];
timer = nil;
}
}

- (void)handleTimer:(NSTimer*)timer
{
if (playingArpeggio)
{
// Play each note of the arpeggio after "arpeggioDelay" seconds.
double now = CACurrentMediaTime();
if (now - arpeggioStartTime >= arpeggioDelay)
{
NSNumber* number = (NSNumber*)[arpegioNotes objectAtIndex:arpeggioIndex];
[player noteOn:[number intValue] gain:0.4f];

++arpeggioIndex;
if (arpeggioIndex == [arpegioNotes count])
{
playingArpeggio = NO;
[arpegioNotes release];
arpegioNotes= nil;
}
else  // schedule next note
{
arpeggioStartTime = now;
}
}
```

```
}
}



- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
if ((self = [super initWithNibName:nibNameOrNil bundle:nibNameOrNil]))
{
playingArpeggio = NO;

// Create the player and tell it which sound bank to use.
player = [[SoundBankPlayer alloc] init];
[player setSoundBank:@"Piano"];

// We use a timer to play arpeggios.
[self startTimer];
}
return self;
}

#pragma mark -
#pragma mark Utilities for intarnal use
-(IplImage *) cvSubImage:(IplImage *) img : (int) x: (int) y: (int) width: (int) heigh
    CvRect r;
    IplImage *sub = cvCreateImage(cvSize(width, height), img->depth, img->nChannels);
    r = cvRect(x, y, width, height);
    cvSetImageROI(img, r);
    sub = cvCloneImage(img);
    r=cvRect(0, 0, img->width, img->height);
    cvSetImageROI(img, r);
    return sub;
}

//For Surf usage
-(double) compareSURFDescriptors:(const float*) image1Descriptor: (const float*) image
    double totalCost =0;
    for (int i =0; i<descriptorscount; i+=4) {
        float descriptor1[4] = {image1Descriptor[i+0], image1Descriptor[i+1], image1De
        float descriptor2[4] = {image2Descriptor[i+0], image2Descriptor[i+1], image2De
        float descriptor3[4] = {descriptor2[0]-descriptor1[0], descriptor2[1]-descript
        totalCost += descriptor3[0]*descriptor3[0]+descriptor3[1]*descriptor3[1]+descr
        if(totalCost>lastMinSquaredDistance)
```

```
            break;
    }
    return totalCost;
}

-(int) findNaiveNearestNeighbor:(const float*) image1Descriptor:(const CvSURFPoint*) i
    int descriptorCount = (int)(image2Descriptors->elem_size/sizeof(float));
    //numeric_limits<double>::max() = 1.79769*pow(10,308)
    double minSquaredDistance = 1.79769* pow(10, 308);
    double lastMinSquaredDistance = 1.79769* pow(10, 308);

    int neighbor;
    for(int i =0; i<image2Descriptors->total; i++){
        const CvSURFPoint* image2KeyPoint = (const CvSURFPoint*) cvGetSeqElem(image2Ke
        const float* image2Descriptor = (const float*) cvGetSeqElem(image2Descriptors,

        if (image1KeyPoint->laplacian!= image2KeyPoint->laplacian){
            continue;
        }
        double squaredDistance =[self compareSURFDescriptors:image1Descriptor :image2D
        if(squaredDistance<minSquaredDistance){
            neighbor =i;
            lastMinSquaredDistance = minSquaredDistance;
            minSquaredDistance = squaredDistance;
        }
        else{
            if(squaredDistance<lastMinSquaredDistance){
                lastMinSquaredDistance = squaredDistance;
            }
        }
    }

    if(minSquaredDistance<0.7*lastMinSquaredDistance){
        return neighbor;
    }
    return -1;

}


-(void) temposort: (IplImage *) source: (note*) source_note{
    //printf("tempsort start\n");
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
```

```
CvMemStorage *storage = cvCreateMemStorage(0);

UIImage *gclef = [UIImage imageNamed:@"GClef.jpg"];
UIImage *quarter = [UIImage imageNamed:@"quarter_note.jpg"];
UIImage *half = [UIImage imageNamed:@"half_note.jpg"];
//convert to Ipliamge
IplImage *temp_GClef = [self CreateIplImageFromUIImageGray:gclef];
IplImage *temp_quarter = [self CreateIplImageFromUIImageGray:quarter];
IplImage *temp_half = [self CreateIplImageFromUIImageGray:half];

CvSeq *GClef_keypoints,  *quarter_keypoints, *half_keypoints;
CvSeq *GClef_descriptors, *quarter_descriptors, *half_descriptors;
CvSURFParams params = cvSURFParams(500, 1);
cvExtractSURF(temp_GClef, 0, &GClef_keypoints, &GClef_descriptors, storage, params
cvExtractSURF(temp_quarter, 0, &quarter_keypoints, &quarter_descriptors, storage,
cvExtractSURF(temp_half, 0, &half_keypoints, &half_descriptors, storage, params, 0

//using cvExtractSurf
CvSeq* image1Keypoints;
CvSeq* image1Descriptors;
CvSeq* image2Keypoints;
CvSeq* image2Descriptors;

//only value with a hessian greater than 500 are consider as keypoint
cvExtractSURF(source, 0, &image1Keypoints, &image1Descriptors, storage, params, 0)
cvExtractSURF(temp_half, 0, &image2Keypoints, &image2Descriptors, storage, params,

//find matching keypoint in both image
NSMutableArray *matchlist1= [[NSMutableArray alloc] init];
NSMutableArray *matchlist2 = [[NSMutableArray alloc] init];
NSMutableArray *gclefmatchlist = [[NSMutableArray alloc] init];
NSMutableArray *quartermatchlist = [[NSMutableArray alloc] init];
NSMutableArray *halfmatchlist = [[NSMutableArray alloc] init];

for(int i=0; i<image1Descriptors->total; i++){
    const CvSURFPoint* image1KeyPoint = (const CvSURFPoint*) cvGetSeqElem(image1Ke
    const float* image1Descriptor = (const float*) cvGetSeqElem(image1Descriptors,
    int nearestNeighbor = [self findNaiveNearestNeighbor:image1Descriptor :image1K
    int nearest_gclef = [self findNaiveNearestNeighbor:image1Descriptor :image1Key
    int nearest_quarter = [self findNaiveNearestNeighbor:image1Descriptor: image1K
    int nearest_half = [self findNaiveNearestNeighbor:image1Descriptor :image1KeyP
    MyCvSurfPoint* check;
    CvSURFPoint* check_point;
```

```
    if(nearest_gclef!= -1){
        check_point = (CvSURFPoint*) cvGetSeqElem(GClef_keypoints, nearest_gclef);
        check = [[MyCvSurfPoint alloc] init_point:check_point->pt];
        [gclefmatchlist addObject:check];
    }
    if(nearest_quarter !=-1){
        check_point = (CvSURFPoint*) cvGetSeqElem(quarter_keypoints, nearest_quart
        check = [[MyCvSurfPoint alloc] init_point:check_point->pt];
        [quartermatchlist addObject:check];
    }
    if(nearest_half !=-1){
        check_point = (CvSURFPoint*) cvGetSeqElem(half_keypoints, nearest_half);
        check = [[MyCvSurfPoint alloc] init_point:check_point->pt];
        [halfmatchlist addObject:check];
    }

    if(nearestNeighbor == -1){
        continue;
    }
    MyCvSurfPoint *temp_surfPoint;
    CvSURFPoint* temp_s = (CvSURFPoint*) cvGetSeqElem(image1Keypoints, i);
    temp_surfPoint = [[MyCvSurfPoint alloc] init_point:temp_s->pt];

    MyCvSurfPoint *temp_surfPoint2;
    CvSURFPoint* temp_s2 = (CvSURFPoint*)cvGetSeqElem(image2Keypoints, nearestNeig
    temp_surfPoint2 = [[MyCvSurfPoint alloc] init_point:temp_s2->pt];
    [matchlist1 addObject:temp_surfPoint];
    [matchlist2 addObject:temp_surfPoint2];

}

//find maxima of matches
int count_queue[3]={[gclefmatchlist count], [quartermatchlist count], [halfmatchli
int count_queue_max = count_queue[0];
for(int count = 0; count<3; count++){
    if(count_queue[count]>count_queue_max){
        count_queue_max = count_queue[count];
    }
}

//calculate ratio and display
NSMutableString *type;
```

```
type = [NSMutableString stringWithString:@"Don't know\n"];
if([gclefmatchlist count] == count_queue_max){
    type = [NSMutableString stringWithString:@"Gclef\n"];
    printf("surf count=%d, counti=%d counti1=%d\n", [matchlist1 count], image1Desc

    int center_x = source_note.getX;
    if(center_x >= 0.1*source->width){
         type = [NSMutableString stringWithString:@"Quarter or Half Note\n"];
    }

}
if([quartermatchlist count] ==count_queue_max){
    type = [NSMutableString stringWithString:@"Quarter or Half Note\n"];
    printf("surf count=%d, counti=%d counti1=%d\n", [matchlist1 count], image1Desc
}
if([halfmatchlist count] == count_queue_max){
    type = [NSMutableString stringWithString:@"Quarter or Half Note\n"];
    printf("surf count=%d, counti=%d counti1=%d\n", [matchlist1 count], image1Desc
}

//There has been problem since the low resolution of images, causes each note has
//It has been difficult to determine which is half note and which is whole note.
//I have create a code that check the how thick is the black pixel, if it is <10%
if([type isEqualToString:@"Quarter or Half Note\n"]){
    IplImage *type_temp = [source_note getImg];
    int image_width = [source_note getW];
    int image_height =[source_note getL];
    int center_x = image_width/2;
    int darkpixelcount = 0;
    int darkwhiteocc = 0;
    int whtiedarkocc = 0;
    int current=0, previous=-1;
    for(int temp_count = 0; temp_count<image_height; temp_count++){
        CvScalar check = cvGet2D(type_temp, temp_count, center_x);
        if(check.val[0]==0){
            current = 1;
            darkpixelcount++;
        }
        if(check.val[0]==255){
            current = 0;
            darkpixelcount=0;
        }
        if(previous==1 && current==0){
```

```
            darkwhiteocc++;
        }
        if(previous==0 && current==1){
            whtiedarkocc++;
        }
        previous = current;

    }
    if(darkpixelcount <= image_height*0.1|| whtiedarkocc>1){
        type = [NSMutableString stringWithString:@"Half Note\n"];
        if(darkwhiteocc>=4){
            type = [NSMutableString stringWithString:@"Gclef\n"];
        }
    }
    else{
        type = [NSMutableString stringWithString:@"Quarter Note\n"];

        //Detecting Bar
        //During our testing, there seem that bar share same feature with quarter
        if(darkpixelcount >=0.6*image_height){
            type = [NSMutableString stringWithString:@"Bar line\n"];
        }
        if(darkwhiteocc>=4){
            type = [NSMutableString stringWithString:@"Gclef\n"];
        }


    }
}

//locating missing gclef
if([type isEqualToString:@"Quarter Note\n"] || [type isEqualToString:@"Half Note\n
    IplImage *type_temp2 = [source_note getImg];
    int image_width = [source_note getW];
    int image_height =[source_note getL];
    int center_x = image_width/2;
    int darkpixelcount = 0;
    int darkwhiteocc = 0;
    int whtiedarkocc = 0;
    int count_distance_bot = 0;
    int count_distance_top = 0;
    Boolean bol_check_top= false;
    Boolean bol_check_bot= false;
```

```
int current=0, previous=-1;
for(int temp_count = 0; temp_count<image_height; temp_count++){
    CvScalar check = cvGet2D(type_temp2, temp_count, center_x);
    if(check.val[0]==0){
        current = 1;
        darkpixelcount++;
    }
    if(check.val[0]==255){
        current = 0;
        darkpixelcount=0;
    }
    if(previous==1 && current==0){
        darkwhiteocc++;
    }
    if(previous==0 && current==1){
        whtiedarkocc++;
    }
    if(temp_count==0 && current==1){
        bol_check_top = TRUE;
    }
    if(temp_count>image_height*0.5 && whtiedarkocc==1){
        bol_check_bot = TRUE;
    }
    if(bol_check_bot){
        count_distance_bot++;
    }
    if(bol_check_top){
        count_distance_top++;
    }
    if(darkwhiteocc == 2){
        bol_check_top = FALSE;
    }
    if(whtiedarkocc==2){
        bol_check_bot = FALSE;
    }
    previous = current;
}

if([type isEqualToString:@"Quarter Note\n"])
{
    if(whtiedarkocc==1){
        int temp_pix_y = source_note.getMaxY- (darkpixelcount/2);
        [source_note setPitch_y:temp_pix_y];
```

```
            }
            if(darkwhiteocc==1){
                int temp_pix_y = source_note.getMinX +(darkpixelcount/2);
                [source_note setPitch_y:temp_pix_y];
            }
        }
        if([type isEqualToString:@"Half Note\n"]){
            if(count_distance_top>count_distance_bot){
                int temp_pix_y = source_note.getMinY- (count_distance_top/2);
                [source_note setPitch_y:temp_pix_y];
            }
            else{
                int temp_pix_y =source_note.getMaxY-(count_distance_bot/2);
                [source_note setPitch_y:temp_pix_y];
            }
        }
    }

    [source_note setType:type];
    cvReleaseMemStorage(&storage);
    [pool release];
}



-(IplImage *) mypaint: (IplImage *) img :(NSInteger) r: (NSInteger) c{

//Convert to binary
IplImage *im_bw =  cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
cvThreshold(img, im_bw, 90, 255, CV_THRESH_BINARY);

    IplImage *bw_out = cvCloneImage(im_bw);

int M = bw_out->width;
    int neighbor_offsets[4]= {-1, M, 1, -M};
NSMutableArray *active_pixels;
    active_pixels = [[NSMutableArray alloc] init];
    int temp_loc = r*M+c+1;
    [active_pixels addObject:[NSNumber numberWithInt:temp_loc]];
    CvScalar s = cvScalar(255,0,0,0);

    while([active_pixels count]!=0){
        NSMutableArray *active_temp;
```

```
active_temp =[[NSMutableArray alloc] init];
for(int count=0; count<[active_pixels count]; count++){

    int temp_loc2 = [[active_pixels objectAtIndex:count] intValue];
    int temp_r = 0;
    if(temp_loc2>M){
        temp_r=temp_loc2/M;
    }
    int temp_c = temp_loc2-M*temp_r;
    cvSet2D(bw_out, temp_r, temp_c, s);

    //The new active pixels list is the set of neighbbors of current list
    //Paintbuckets
    for(int M_count =0; M_count<4; M_count++){
        int temp = neighbor_offsets[M_count]+temp_loc2;

        [active_temp addObject:[NSNumber numberWithInt:temp]];
    }
}

[active_pixels removeAllObjects];

for(int count2=0; count2<[active_temp count]; count2++){

    int check = [[active_temp objectAtIndex:count2] intValue];
    int check_r=0;
    if(check>M){
        check_r = check/M;
    }
    int check_c = check-M*check_r;
    if(check_c < bw_out->width && check_r < bw_out->height && check_c>=0 && ch
    {
        //remove from active_pixels list pixels that are already 1.
        //We keep the flood fill operation "inside" of white boundaries
        CvScalar Check_s = cvGet2D(bw_out, check_r, check_c);
        if(Check_s.val[0]== 0){
            [active_pixels addObject:[NSNumber numberWithInt:[[active_temp obj
        }
    }
}

[active_temp release];
//Remove duplicates from the list
```

```
        NSArray *copy = [active_pixels copy];
        NSInteger index = [copy count] -1;
        for(id object in [copy reverseObjectEnumerator]){
            if([active_pixels indexOfObject: object inRange: NSMakeRange(0, index)]!=N
                [active_pixels removeObjectAtIndex:index];
            }
            index--;
        }
        [copy release];
    }
    IplImage *bw_out2= cvCreateImage(cvSize(bw_out->width, bw_out->height), IPL_DEPTH_
    cvCopy(bw_out, bw_out2, NULL);
    cvResetImageROI(bw_out);
    return bw_out2;
}

-(void)imageProc {
NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
if(theimageView.image) {
cvSetErrMode(CV_ErrModeParent);

//create grayscale IplImage from UIImage
/*IplImage *img_color = [self CreateIplImageFromUIImage:theimageView.image];
IplImage *img = cvCreateImage(cvGetSize(img_color), IPL_DEPTH_8U, 1);
cvCvtColor(img_color, img, CV_BGR2GRAY);
cvReleaseImage(&img_color);


//Convert gray scale to binary
        cvThreshold(img, img, 90, 255, CV_THRESH_BINARY);
        */

UIImage *grace = [UIImage imageNamed:@"wordlesstwinkle.jpg"];
        IplImage *img = [self CreateIplImageFromUIImageGray:grace];
        int temp [img->height];
        int count=0;

        //Count number of black pixel in each row
for(int y=0; y < img->height; y++){
for(int x=0; x < img->width; x++){
                CvScalar s = cvGet2D(img, y, x);
if(s.val[0]==0){
count =count+1;
```

```
}
}
            temp[y]=count;
            count=0;
        }

        MySheet *sheet= [[MySheet alloc] init_note];
//Remove staff line
for(int tempcount=0;tempcount<img->height-1; tempcount++){
            if(temp[tempcount]>0.5*img->width){
                [sheet addstaff:tempcount];
for(int tempcount2=0; tempcount2<img->width; tempcount2++){
uchar * ptr2 = cvPtr2D(img, tempcount, tempcount2, NULL);
if(ptr2[0] < 127){
                        uchar * ptr21 = cvPtr2D(img, tempcount-1, tempcount2, NULL);
                        uchar * ptr22 = cvPtr2D(img, tempcount+1, tempcount2, NULL);
                        if(ptr21[0]>127||ptr22[0]>127){
                        CvScalar s2 = cvScalar(255,0,0,0);
                        cvSet2D(img, tempcount, tempcount2, s2);
}
}
}
}
}

        IplImage * temp_image = cvCloneImage(img);
        IplImage *imagedata[1000];
        //Recount the black pixel;
        for(int i = 0; i < temp_image->height; i++){
            for(int j=0; j < temp_image->width; j++){
                CvScalar scheck= cvGet2D(temp_image, i, j);
                if(scheck.val[0]==0){
                    count = count+1;
                    temp_image = [self mypaint:temp_image: i: j];
                    imagedata[count-1]=temp_image;

                }
            }
        }

        IplImage *difference [1000];
        note *note_data[1000];
        temp_image = cvCloneImage(img);
```

```
//Extraction
int max_x, max_y, min_x, min_y;
IplImage *temp_note;
for(int k=0; k < count; k++){
    max_x =0; max_y=0; min_x = temp_image->width; min_y=temp_image->height;
    difference[k] = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
    if(k==0)
    {
        for(int diff_y=0; diff_y<temp_image->height; diff_y++){
            for(int diff_x=0; diff_x<temp_image->width; diff_x++){
                CvScalar check_diff=cvGet2D(temp_image, diff_y, diff_x);
                CvScalar check_diff2=cvGet2D(imagedata[0], diff_y, diff_x);
                if(check_diff.val[0]==0 && check_diff2.val[0]==255){
                    CvScalar img_pix = cvScalar(0, 0, 0, 0);
                    if(diff_x>=max_x){
                        max_x = diff_x;
                    }
                    if(diff_x<=min_x){
                        min_x = diff_x;
                    }
                    if(diff_y>=max_y){
                        max_y = diff_y;
                    }
                    if(diff_y<=min_y){
                        min_y = diff_y;
                    }
                    cvSet2D(difference[k], diff_y, diff_x, img_pix);
                }
                else{
                    CvScalar img_pix2 = cvScalar(255, 0, 0, 0);
                    cvSet2D(difference[k], diff_y, diff_x, img_pix2);
                }
            }
        }
        note_data[k]=[[note alloc] initWithCoord:max_x :min_x :max_y :min_y];
        temp_note = [self cvSubImage:difference[k]: min_x: min_y: [note_data[k
        [note_data[k] setImg:temp_note];
        cvReleaseImage(&temp_note);
    }
    else
    {
        for(int diff_y2=0; diff_y2<img->height; diff_y2++){
```

```
            for(int diff_x2=0; diff_x2<img->width; diff_x2++){
                CvScalar check_diff3=cvGet2D(imagedata[k], diff_y2, diff_x2);
                CvScalar check_diff4=cvGet2D(imagedata[k-1], diff_y2, diff_x2)
                if(check_diff3.val[0]==255 && check_diff4.val[0]==0){
                    CvScalar img_pix3 = cvScalar(0, 0, 0, 0);
                    if(diff_x2>=max_x){
                        max_x = diff_x2;
                    }
                    if(diff_x2<=min_x){
                        min_x = diff_x2;
                    }
                    if(diff_y2>=max_y){
                        max_y = diff_y2;
                    }
                    if(diff_y2<=min_y){
                        min_y = diff_y2;
                    }
                    cvSet2D(difference[k], diff_y2, diff_x2, img_pix3);
                }
                else{
                    CvScalar img_pix4 = cvScalar(255, 0, 0, 0);
                    cvSet2D(difference[k], diff_y2, diff_x2, img_pix4);
                }
            }
        }

        note_data[k]=[[note alloc] initWithCoord:max_x :min_x :max_y :min_y];
        temp_note = [self cvSubImage:difference[k]: min_x: min_y: [note_data[k

        [note_data[k] setImg:temp_note];

        cvReleaseImage(&temp_note);
    }
}

IplImage *data_temp;
data_temp = [note_data[0] getImg];
int index_num =5;
//index 0 = gclef, 5 = half note, 8 = quarter note

img = cvCloneImage(difference[index_num]);
[self temposort:difference[index_num] : note_data[index_num]];
```

```
NSMutableString *tempString = [note_data[index_num] getType];
NSLog(@"Type=%@\n", tempString);


for(int count_temp=0; count_temp< count;count_temp++){
    [self temposort:difference[count_temp]: note_data[count_temp]];
    [note_data[count_temp] setDiff:difference[count_temp]];
    [sheet addNote:note_data[count_temp]];
    tempString = [note_data[count_temp] getType];

}

[sheet sort];
[sheet getPitchlist];

NSMutableArray *pitcharray=[[NSMutableArray alloc] init];
for(int count =0; count <[sheet getSize]; count++){
    note *temp_note = [sheet getNote:count];
    [pitcharray addObject:[NSNumber numberWithInt:[temp_note getPitch]]];
}

[self playArpeggioWithNotes:pitcharray delay:0.06];

NSMutableArray *MyUIImag = [[NSMutableArray alloc] init];
for(int temp_count=0; temp_count<[sheet getSize]; temp_count++){
    IplImage *UI_temp = [sheet getImageAtIndex:temp_count];
    IplImage *image = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 3);
    for(int y2=0; y2<UI_temp->height; y2++) {
        for(int x2=0; x2<UI_temp->width; x2++) {
            char *p = image->imageData + y2 * image->widthStep + x2 * 3;
            *p = *(p+1) = *(p+2) = UI_temp->imageData[y2 * UI_temp->widthStep
        }
    }
    [MyUIImag addObject:[self UIImageFromIplImage:image]];
}
UIImageView *animationView =[[UIImageView alloc] initWithFrame:CGRectMake(0, 0
//animationView.backgroundColor =[UIColor purpleColor];
animationView.animationImages = MyUIImag;
animationView.animationDuration =20;
animationView.animationRepeatCount=0;
[animationView startAnimating];
[self.view addSubview:animationView];
[animationView release];
```

```
        /*
        // Convert black and whilte to 24bit image then convert to UIImage to show
IplImage *image = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 3);
for(int y2=0; y2<img->height; y2++) {
for(int x2=0; x2<img->width; x2++) {
char *p = image->imageData + y2 * image->widthStep + x2 * 3;
*p = *(p+1) = *(p+2) = img->imageData[y2 * img->widthStep + x2];
}
}
cvReleaseImage(&img);
        theimageView.image = [self UIImageFromIplImage:image];
        cvReleaseImage(&image);

printf("done\n");
         */
}

[pool release];
}



#pragma mark -
#pragma mark IBAction

-(IBAction) test:(id) sender
{
[self performSelectorInBackground:@selector(imageProc) withObject:nil];
}

-(IBAction) getPhoto:(id) sender
{
UIImagePickerController *picker = [[UIImagePickerController alloc] init];
picker.delegate = (id)self;

if((UIButton *) sender == choosePhoto)
{
picker.sourceType = UIImagePickerControllerSourceTypeSavedPhotosAlbum;
}
else
{
```

```
picker.sourceType = UIImagePickerControllerSourceTypeCamera;
}
[self presentModalViewController:picker animated:YES];
}

#pragma mark -
#pragma mark UIViewControllerDelegate
/*!
 * @discussion  Implement loadView to create a view hierarchy programmatically,
 *              without using a nib.
 */
/*
 - ( void )loadView
{}
*/

/*!
 * @discussion  Implement viewDidLoad to do additional setup after loading the
 *              view, typically from a nib.
 */
- ( void )viewDidLoad
{
    //[ super viewDidLoad ];
if(![UIImagePickerController isSourceTypeAvailable:UIImagePickerControllerSourceTypeCa
{
takePhoto.hidden = YES;
}
}


-(void) imagePickerController:(UIImagePickerController *) picker didFinishPickingMedia
[picker dismissModalViewControllerAnimated:YES];
theimageView.image = [info objectForKey:@"UIImagePickerControllerOriginalImage"];
}
/*!
 * @discussion  Override to allow orientations other than the default portrait
 *              orientation.
 */
- ( BOOL )shouldAutorotateToInterfaceOrientation: ( UIInterfaceOrientation )interfaceO
{
    return ( interfaceOrientation == UIInterfaceOrientationPortrait );
}
```

```
/*!
 * @discussion  Release any cached data, images, etc that aren't in use.
 */
- ( void )didReceiveMemoryWarning
{
    [ super didReceiveMemoryWarning ];
}

/*!
 * @discussion  Release any retained subviews of the main view.
 */
- ( void )viewDidUnload
{}

/*!
 * @discussion  Called when the object is freed by the Objective-C runtime
 */
- ( void )dealloc
{
    [ super dealloc ];
}

@end
```

XCode MyCvPoint.m

```
//
//  MyCvPoint.m
//  OpenCV-iPhone
//
//  Created by Yung-Ho  Chang on 4/11/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//

#import "MyCvPoint.h"

@implementation MyCvPoint
-(id) init:(int)input_x :(int)input_y{
    x = input_x;
    y = input_x;
    return self;
}
```

```
-(int) getX{
    return x;
}

-(int) getY{
    return y;
}

-(void) setX:(int) input_x{
    x = input_x;
}
-(void) setY:(int) input_y{
    y=input_y;
}
@end
```

XCode MyCvSurfPoint2

```
//
//  MyCvSurfPoint2.m
//  OpenCV-iPhone
//
//  Created by Yung-Ho  Chang on 4/28/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//

#import "MyCvSurfPoint2.h"

@implementation MyCvSurfPoint2
-(id) init_point:(CvSURFPoint *)input_point{
    point = input_point;
    return self;
}
-(CvSURFPoint*) getSurfPoint{
    return point;
}
-(void) setPoint:(CvSURFPoint*) input_Point{
    point = input_Point;
}
@end
```

XCode Myfloat.m

```
//
//  Myfloat.m
//  OpenCV-iPhone
//
//  Created by Yung-Ho  Chang on 4/28/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//

#import "Myfloat.h"

@implementation Myfloat
-(id)init_float:(float *)input_temp{
    temp = input_temp;
    return self;
}
-(float*) getfloat{
    return temp;
}
-(void) setfloat:(float *)input_temp{
    temp = input_temp;
}
@end
```

XCode MyIpl.m

```
//
//  MyIpl.m
//  OpenCV-iPhone
//
//  Created by Yung-Ho  Chang on 4/29/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//

#import "MyIpl.h"

@implementation MyIpl
-(id)init_Ipl:(IplImage *)input_image{
    image = cvCloneImage(input_image);
    return self;
}
```

```
-(IplImage*) getIpl{
    return image;
}
-(void) setIpl:(IplImage *)input_image{
    image = cvCloneImage(input_image);
}
-(int) getheight{
    return image->height;
}
-(int) getwidth{
    return image->width;
}
@end
```

XCode MySheet.m

```
//
//  MySheet.m
//  OpenCV-iPhone
//
//  Created by Yung-Ho  Chang on 4/29/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//

#import "MySheet.h"
#import "MyIpl.h"
#import "note.h"

@implementation MySheet
-(id) init_note{
    notes = [[NSMutableArray alloc] init ];
    staff = [[NSMutableArray alloc] init];
    staff_inv=0;
    return self;
}
-(void) addstaff:(int)input_staff{
    [staff addObject:[NSNumber numberWithInt:input_staff]];
    //printf("added\n");
    [self findStaffinv];
}

-(void) findStaffinv{
```

```
NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
NSMutableArray *staff_inv_temp = [[NSMutableArray alloc] init ];
NSMutableArray *inv = [[NSMutableArray alloc] init];
staff_inv_temp = [staff mutableCopy];
if([staff count]>1){
    while ([staff_inv_temp count]>1) {
        NSNumber *temp_NS = [staff_inv_temp objectAtIndex:0];
        NSNumber *temp_NS2 = [staff_inv_temp objectAtIndex:1];
        NSNumber *temp_inv = [NSNumber numberWithInt:[temp_NS2 intValue]-[temp_NS
        [inv addObject:temp_inv];
        [staff_inv_temp removeObjectAtIndex:0];
    }

    NSNumber *check1;
    NSNumber *countNS = [NSNumber numberWithInt:1];
    NSMutableArray *type = [[NSMutableArray alloc] init];
    NSMutableArray *countNum = [[NSMutableArray alloc] init];
    //[type addObject:[staff objectAtIndex:0]];
    //printf("find mode, invcount=%d\n", [inv count]);
    for(int temp =0; temp<[inv count]; temp++){
        check1 = [inv objectAtIndex:temp];
        //printf("mode check1=%d\n", [check1 intValue]);
        NSInteger check_index = [type indexOfObject:check1];
        if(NSNotFound == check_index){
            //printf("not found\n");
            [type addObject: check1];
            [countNum addObject:countNS];
        }
        else{
            NSNumber *newNS = [countNum objectAtIndex:check_index];
            int value = [newNS intValue];
            newNS = [NSNumber numberWithInt:value+1];
            [countNum replaceObjectAtIndex:check_index withObject:newNS];
        }
    }
    NSNumber *max = [NSNumber numberWithInt:0];
    //printf("max started=%d\n", [max intValue]);
    for(int temp2=0; temp2 < [countNum count];  temp2++){
        check1 = [countNum objectAtIndex:temp2];
        //printf("max check1=%d", [check1 intValue]);
        if([max intValue]<[check1 intValue]){
            //printf("in max\n");
            max = [ NSNumber numberWithInt:[check1 intValue]];
```

```
            }
        }
        //printf("max=%d\n", [max intValue]);
        NSInteger check_index2 = [countNum indexOfObject:max];
        staff_inv = [[type objectAtIndex:check_index2]intValue];
    }
    //printf("staff_inv=%d\n", staff_inv);
    [pool release];
}

-(void) addNote:(note *)input_note{
    [notes addObject: input_note];
}
-(IplImage*)getImageAtIndex:(int)input_index{
    note *temp_note = [notes objectAtIndex:input_index];
    IplImage *tempIpl = cvCloneImage(temp_note.getDiff);
    return tempIpl;
}
-(int) getSize{
    return [notes count];
}

-(note*) getNote:(int)input_index{
    return [notes objectAtIndex:input_index];
}

-(void) sort{
    NSMutableArray *temp_Notes = [[NSMutableArray alloc] init ];
    NSMutableArray *temparray = [[NSMutableArray alloc] init];
    NSMutableArray *temp_Notes2 = [[NSMutableArray alloc] init];
    NSMutableArray *pitchlist = [[NSMutableArray alloc] init];
    [pitchlist addObject:[NSMutableString stringWithString:@"A"]];
    [pitchlist addObject:[NSMutableString stringWithString:@"B"]];
    [pitchlist addObject:[NSMutableString stringWithString:@"C"]];
    [pitchlist addObject:[NSMutableString stringWithString:@"D"]];
    [pitchlist addObject:[NSMutableString stringWithString:@"E"]];
    [pitchlist addObject:[NSMutableString stringWithString:@"F"]];
    [pitchlist addObject:[NSMutableString stringWithString:@"G"]];
    temp_Notes2 = [notes mutableCopy];
    int range_height_min = 0;
    int range_height_max = 0;
    int staffcount=-5;
    int trig = 0;
```

```
int Gfound = 0;
while([notes count]>0){

    for(int count =0; count<[notes count]; count++){
        note *temp_note = [notes objectAtIndex:count];
        NSMutableString *temp_type = [temp_note getType];
        if([temp_type isEqualToString:@"Gclef\n"]){
            Gfound = 1;
            staffcount+=5;
            range_height_min = temp_note.getY-temp_note.getL;
            range_height_max = temp_note.getY+temp_note.getL;
            if(range_height_min<0){
                range_height_min = 0;
            }
            if(range_height_max>temp_note.getDiff->height){
                range_height_max =temp_note.getDiff->height;
            }
            [temp_Notes addObject:temp_note];
            [notes removeObjectAtIndex:count];
            break;
        }
        if(Gfound==1 && temp_note.getMinY<range_height_max && temp_note.getMaxY<ra

            if(![temp_type isEqualToString:@"Bar line\n"]){
                int pitc_y = temp_note.getPitch_y;
                int temp_pitch = [[staff objectAtIndex:staffcount] intValue];
                int distance = pitc_y -temp_pitch;
                int pitch_index = 5;
                int octive = 5;
                int key = 77;
                NSMutableString *newpitch;
                if(distance>0){
                for(int count2 =0; distance>staff_inv/2; count2++){
                    int fill = (double)staff_inv/(double)2+3;
                    if(pitch_index==0){
                        pitch_index = 6;
                        octive --;
                    }
                    if(pitch_index==2||pitch_index==5){
                        key--;
                    }
                    else{
                        key=key-2;
```

```
                }
                pitch_index--;
                distance -= fill;
            }
            newpitch = [pitchlist objectAtIndex:pitch_index];
            [temp_note setLetter:newpitch];
            [temp_note setOCtive:octive];
        }
        else{
            for(int count2 =0; distance>staff_inv/2; count2++){
                int fill = (double)staff_inv/(double) 2+3;
                if(pitch_index==6){
                    pitch_index = 0;
                    octive++;
                }
                if(pitch_index==1||pitch_index==4){
                    key++;
                }
                else
                {
                    key = key+2;
                }
                pitch_index++;
                distance += fill;
            }
            newpitch = [pitchlist objectAtIndex:pitch_index];
            [temp_note setOCtive:octive];
            [temp_note setLetter:newpitch];
            [temp_note setPitch2:key];
        }}
        [temparray addObject: temp_note];
        [notes removeObjectAtIndex:count];
        count--;
    }
    else{
        if(Gfound==1){
            trig = 1;
            Gfound=0;
            break;
        }
        continue;
    }
}
```

```
        if(trig ==1){
            //bubble sort
            for(int i = ([temparray count]-1); i>0; i--){
                for(int j=1; j<=i; j++){
                    note *sort_temp = [temparray objectAtIndex:j-1];
                    note *sort_temp2 = [temparray objectAtIndex:j];
                    if([sort_temp getMinX] > [sort_temp2 getMinX]){
                        [temparray exchangeObjectAtIndex:j-1 withObjectAtIndex:j];
                    }
                }
            }
            for(int count3 = 0; count3<[temparray count]; count3++){
                [temp_Notes addObject:[temparray objectAtIndex:count3]];
            }
            [temparray removeAllObjects];
            trig = 0;
        }

    }
    //bubble sort
    for(int i = ([temparray count]-1); i>0; i--){
        for(int j=1; j<=i; j++){
            note *sort_temp = [temparray objectAtIndex:j-1];
            note *sort_temp2 = [temparray objectAtIndex:j];
            if([sort_temp getMinX] > [sort_temp2 getMinX]){
                [temparray exchangeObjectAtIndex:j-1 withObjectAtIndex:j];
            }
        }
    }
    for(int count3 = 0; count3<[temparray count]; count3++){
        [temp_Notes addObject:[temparray objectAtIndex:count3]];
    }
    [temparray removeAllObjects];
    notes = [temp_Notes mutableCopy];
}
-(void)getPitchlist{

    for(int temp_count =0; temp_count<[notes count]; temp_count++){
        note *temp_note = [notes objectAtIndex:temp_count];
        NSLog(@"%@\n", [temp_note getletter]);
    }
}
@end
```

XCode note.m

```
//
//  note.m
//  OpenCV-iPhone
//
//  Created by Yung-Ho  Chang on 4/9/12.
//  Copyright (c) 2012 __MyCompanyName__. All rights reserved.
//

#import "note.h"
@implementation note

-(int) getX{
    return center_x;
}

-(int) getY{
    return center_y;
}

-(int) getW{
    return width;
}

-(int) getL{
    return length;
}

-(void) setX:(int)input_x{
    center_x = input_x;
}

-(void) setY:(int)input_y{
    center_y = input_y;
}
-(void) setW:(int)input_w{
    width = input_w;
}
-(void) setL:(int)input_l{
    length = input_l;
```

```
}
-(IplImage *) getImg{
    return note_image;
}
-(void) setImg:(IplImage *) input_img{
    note_image = cvCloneImage(input_img);
}
-(id) initWithCoord:(int)max_x:(int)min_x:(int)max_y:(int)min_y{
    note_min_x = min_x;
    note_min_y = min_y;
    note_max_x = max_x;
    note_max_y = max_y;
    int temp_w = max_x-min_x;
    int temp_l = max_y-min_y;
    if(temp_w<=0){
        temp_w=1;
    }
    if(temp_l<=0)
    {
        temp_l = 1;
    }
    int temp_x = min_x + temp_w/2;
    int temp_y = min_y + temp_l/2;
    [self setX:temp_x];
    [self setY:temp_y];
    [self setW:temp_w];
    [self setL:temp_l];
    return self;
}
-(void) setPitch2:(int)input_pitch{
    pitch  = input_pitch;
}
-(NSMutableString*) getType{
    return type;
}

-(int) getPitch{
    return pitch;
}

-(void) setType:(NSMutableString *)input_string{
    type = [input_string mutableCopy];
}
```

```
-(int) getMinX{
    return note_min_x;
}
-(int) getMaxX{
    return note_max_x;
}
-(int) getMinY{
    return note_min_y;
}
-(int) getMaxY{
    return note_max_y;
}
-(void) setDiff:(IplImage *)input_image{
    difference = cvCloneImage(input_image);
}
-(IplImage*) getDiff{
    return difference;
}
-(void) setPitch_y:(int) input_y{
    pitch_y = input_y;
}

-(int) getPitch_y{
    return pitch_y;
}

-(NSMutableString*) getletter{
    return pitch_letter;
}
-(void) setLetter:(NSMutableString *)inputLetter{
    pitch_letter = [inputLetter copy];
}
-(int) getOctive{
    return octive;
}
-(void) setOCtive:(int)input_int{
    octive = input_int;
}
@end
```