EENG 507 Final Project

Piano Player

Haoxuan Yang, Ryan Thorpe

December 12, 2016

Haoxuan Yang
12/12/2016
Ryan Thorpe

EENG 507 Computer Vision

Final Project

INTRODUCTION

Augmented reality applications are on the rise and scientists and engineers are finding new applications. With the increase of processing power even mobile phones can perform computer computer vision operations to help the user learn new skills or analyze their surroundings. For this project we will focus on how augmented reality could help anyone learn to play the piano. It becomes more apparent every year that people wish to learn more skills in less time. Although, there are obvious downsides to such a hasty approach to learning, there is a substantial market for it. Applications like Photomath allow you to photograph a math problem and solve it in real time without knowing how to even add. Our piano playing program seeks to obviate the character building exercise of learning to read and understand musical structure. It will read music from an LCD screen and directly tell the user what key to play and when. These songs are pre-loaded onto a special piano with an LCD screen.

~Assumptions:

~~Song Selection

The user cannot play just any song. Unfortunately, the user is limited to songs that are pre-programmed into the Casio SA76. The piano has around 20 songs pre-loaded song which allowed enough opportunities to test the image processing program. The piano does not have a standard number of keys and we will only use the first 26 white keys as shown in figure 1

Figure 1 Shows the Casio SA76 piano. Only the white keys will be used and their numbering starts for the left as shown.

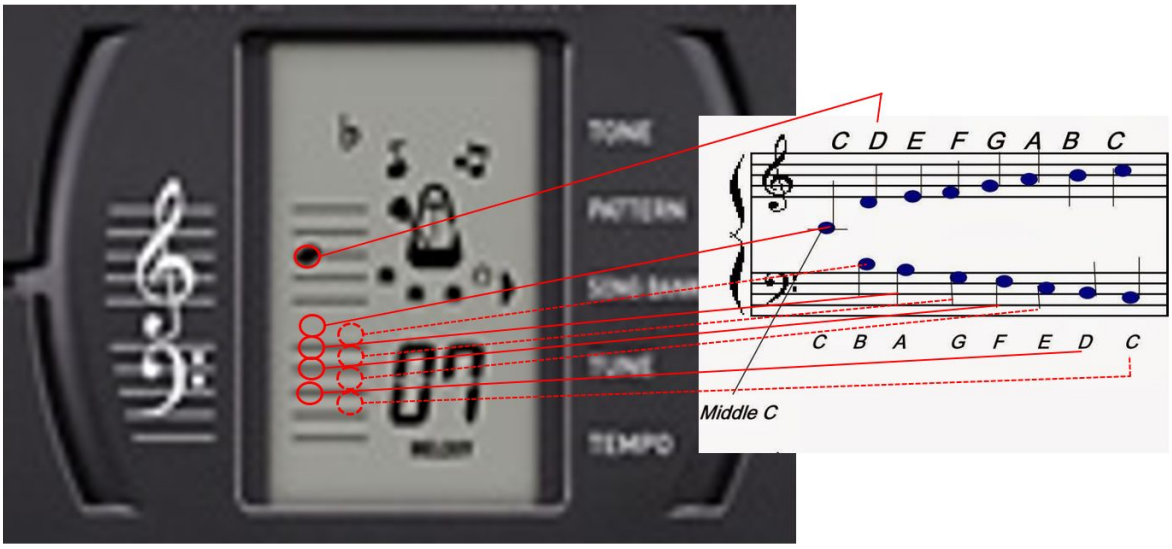


Figure 2. Shows how dots on the LCD screen actually correspond to musical notes

~~Camera Resolution

The camera used was a Microsoft lifecam with 720p of resolution. We recommended cameras with with at least 720p of resolution.

~~Lighting

However, the program is very susceptible to lighting and positional environmental factors. We tested this program in rooms with a somewhat diffuse and constant white light. When the piano is played directly under a light fixture, harsh glaring surfaces on the LCD screen, the white CCC window and on the white piano keys, themselves. We speculate that any point source light should be softened with a piece of white paper to diffuse the light evenly across the piano.

~~Piano Modification

Although piano modification was kept to a minimum some changes were required. For the music reader to be able to localize the LCD screen a white CCC window will need to be added to the face of the piano. This window will be used to calculate the corner homography points needed to perform the orthophoto correction. The inner width and height must be precisely measured and used as constants for the ortho-correcting process.

~~Piano Position

~~~*Music Reader*

Position of the piano is also a key factor for the program to correctly recognize music notes and overlay the next note to be played. Since the LCD screen is recessed into the keyboard creating an orthophoto can vary depending upon the camera's angle of view. This view is similar to parallax and will be discussed later. Ideally, the camera will be positioned directly over the LCD screen. The program does allow for some degree of off angle camera positions.

~~~Key Highlighter and Finger Detector

The keyboard needs to be level with the image borders. This means that the camera should view the piano to be centered within its field of view with little rotation or skew.

~~~The Finger Detector

The finger detector reads one of the user's fingers at a time in the demo video, but it can be applied to multiple fingers (we haven't tried it). The finger detector is also dependent on the position of the keyboard and cannot tolerate much image rotation.

PREVIOUS WORK

As mentioned earlier, mobile phones have made massive strides in computation power and can be used for real time image processing. In the below case a mobile phone can watch a printed keyboard and key a sound that corresponds to the key being played. See image 3. The main deficiency with this approach is that no music is played to help the use learn new songs. In addition, the keyboard only has seven keys which severely limits the number of songs that can be played. Our approach will be able to view the entire keyboard layout; even though the SA76 is a scaled down version of a fullsize keyboard.

Image 3. Shows a cell phone differentiating the key with a finger over it. Once it is decided the proper note will be played. Notice the textured background that helps the program draw borders around the keyboard. The small black keys are also textured in a known way.

More closely aligned with our project there exists a piano player that directly projects the notes to be played onto a full size stand alone piano. The feedback is through a recording microphone whereas our feedback was through a fingertip recognition algorithm. This is a very robust program that can telegraph the next note to be played on both the top of the piano and on the key itself. See image 4. The main drawback with this approach is that a large piano is needed to provide a backdrop for the falling notes. A projector is also needed to display notes onto a flat surface. Our program only needs a computer monitor and webcam to play an augmented reality piano song.

Figure 4. Shows Hasselts University's virtual piano player with notes that cascade down the piano up to the piano keys.

PROGRAM IMPLEMENTATION AND APPLIED TECHNIQUES

~Music Reader - Calibrate

~~Read Video

Reading in the video to MATLAB is a non-trivial task that depends greatly on the lighting and position of the keyboard. Initially, we sought to have this process automated but due to the large changes in light or camera angles that vary from setup to setup this could not be done. This forced us to record a video of the music being played on the LCD screen. This video was used for the rest of the program. See figure 5 for the experimental setup. We developed a MATLAB function called *calibrate()* that contains this entire calibration process and can be found the Appendix [MATLAB CAL CODE].

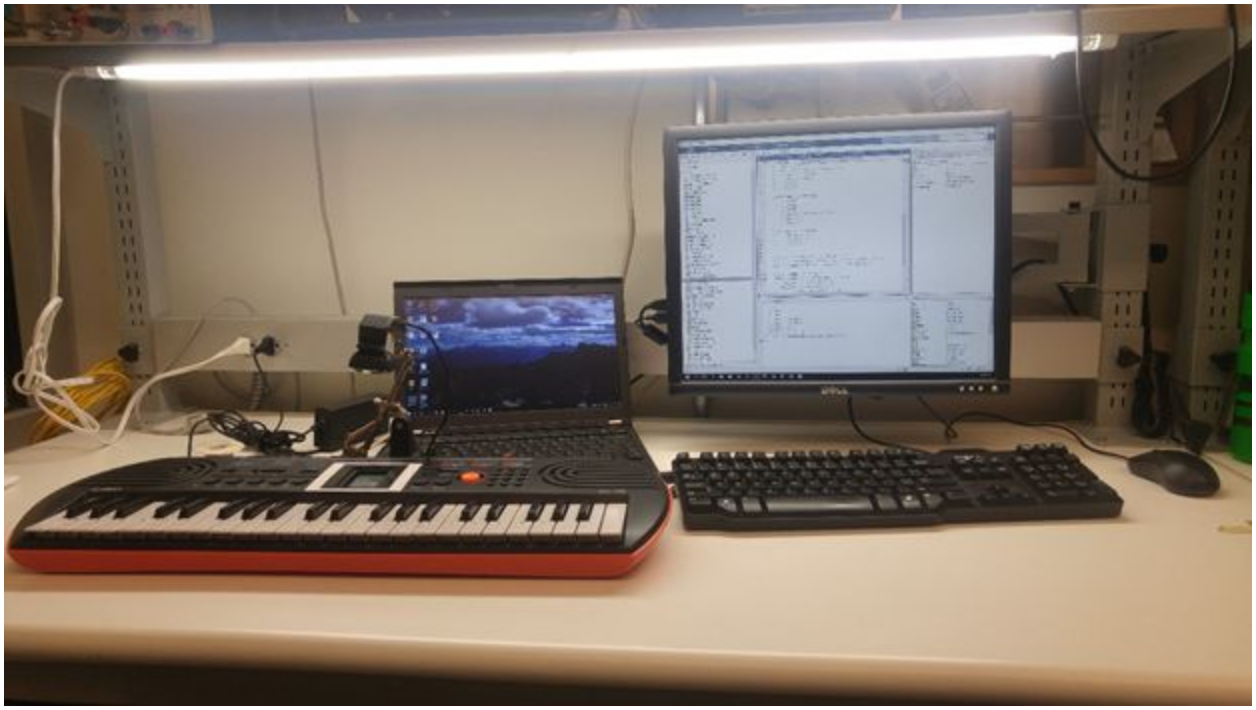


Figure 5. Shows the SA76 piano under a lab light. The light often caused glaring issues on the LCD screen. This glare forced us to constantly change the position of the piano. In addition the position of the webcam over the piano had to be close enough to see the small notes being played on the LCD screen but not place in a position that would cause shadowing or in a position that would be more susceptible to the glare from surrounding light fixtures.

Once the piano was in an acceptable position the rest of the calibration could be carried out. One weakness of our code is that if the piano is ever moved through the playing process the calibration must be carried out again. We have found through testing that the camera would tolerate up to +/- 30 degrees of off center positioning. However, as this angle grows larger the music note region of interest, which will be discussed later, requires more time to calibrate correctly. This is due to the parallax image errors that are introduced into the image in a more serious way.

~~Recognize LCD Screen Area

~~~Extracting the Sharp Sided Quadrilateral

After the piano video has been shown to have low glaring, and potentially good LCD viewing resolution the next step of the process seeks to define the inner corners of the white CCC frame. CCC markers are used in computer vision to localize a point in a dynamic environment where differentiating objects in an image proves difficult. The CCC window has the dimensions shown below in figure 6.

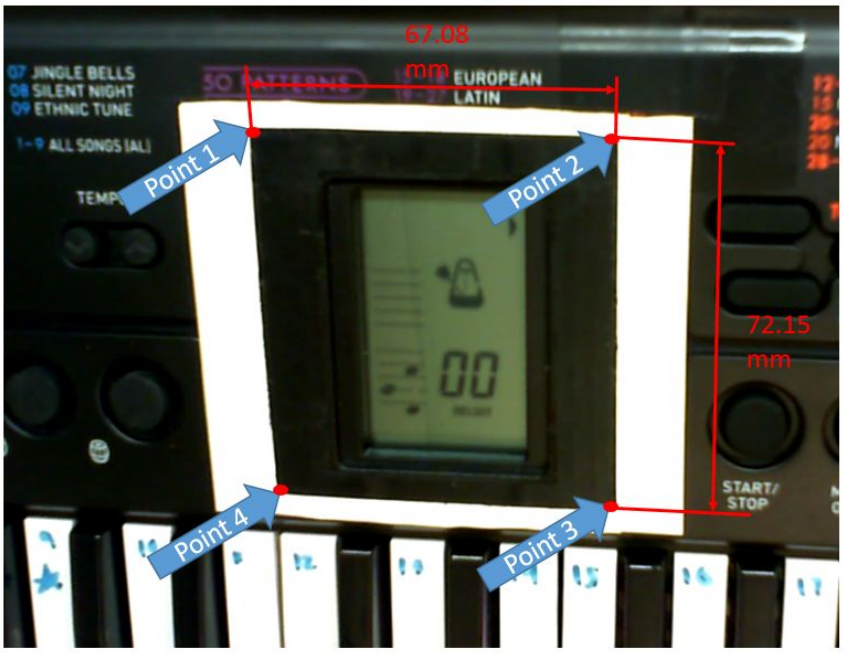


Figure 6. Shows the ground truth of the Dimension of the CCC window. The corner points that are shown in the image are the same corner points that will be used for the homography transform that normalizes the viewing of the LCD screen.

Corners 1-4 are found by first filtering the image down to the most important objects as seen in the figure 7. These images are made by first grayscaling an incoming color video frame using Otsu's method. Once converted an open and closing operation was carried out to smooth out and remove any noise that was left over. This resulting in a simplified grayscale image that would help use identify distinct object within the image. This image was called the positive image and in order to identify the CCC window we also made a negative image. MATLAB's *bwlabel()* was instrumental in identifying all the bodies in both the positive and negative images of the video frame. If no glaring MATLAB will list out around six bodies in each positive and negative image after the *regionprops()* function is called. The body that we are interested is the sharply cornered quadrilateral. That lies within the white CCC frame. Doing a nested for-loop search through all the available bodies in the both positive and negative images will yield two bodies that have near identical centroids. The second largest of these two bodies will be the "sharply cornered quadrilateral" that we seek to calculate corners for. Doing an addition masking step using *masker()* will fully extract this shape as shown in figure 7.

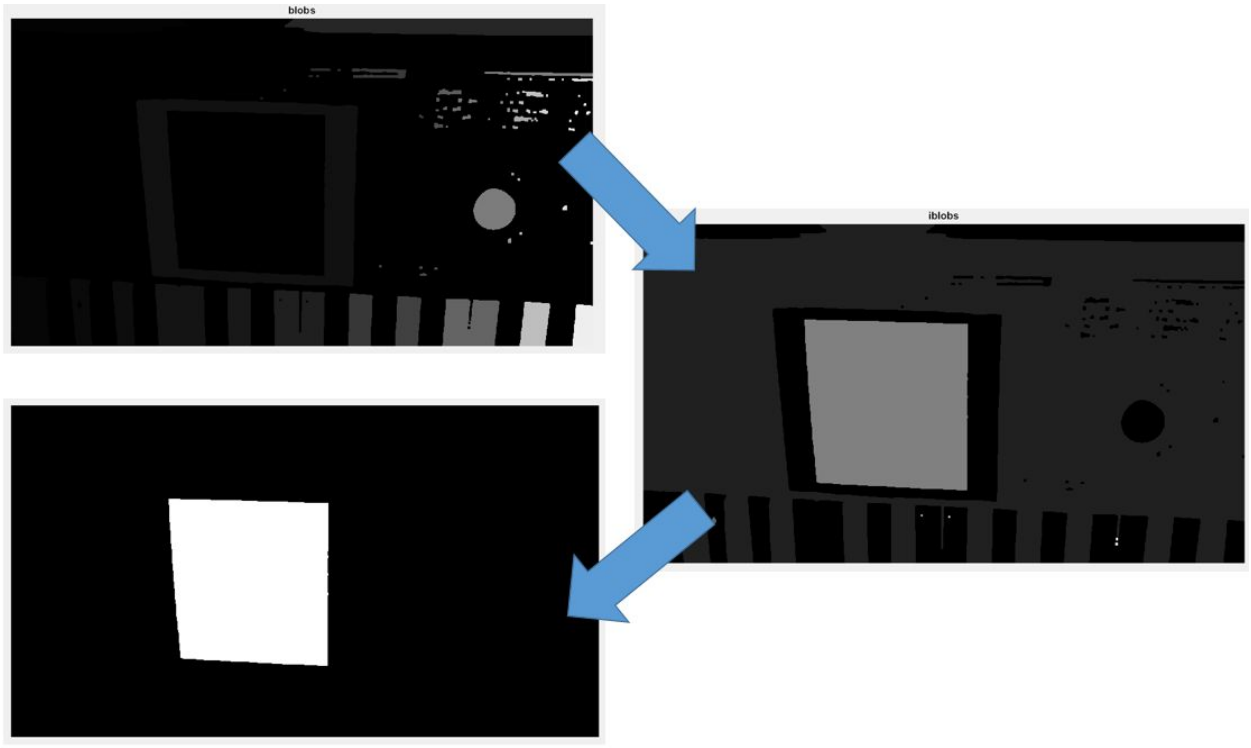


Figure 7. Shows the steps needed to extract the sharpy cornered region that will have its corner point locations extracted. The other two images in the top and at the right are the labeled images that were built from the positive and negative images that resulted from an Otsu grayscaling of the incoming color video frame.

~~~Finding Corner Points

Before attempting to calculate the corner locations a significant effort was made to extract an object that had clearly defined edges. This was because using hough lines on a four sided object that does not have straight edges in an effort to find its corners becomes difficult. Hough lines effectively fit a line to a clustering of edge points so that a best fit line can be defined. This becomes problematic when edges can no longer be approximated with just one line but many. Think of how piecewise curve requires many lines to describe its behavior instead of a scatter plot that only requires one line to describe its behavior. To do this we modified Professor Hoff's findCheckBoard() function to output only four lines describing the CCC window. See figure 8

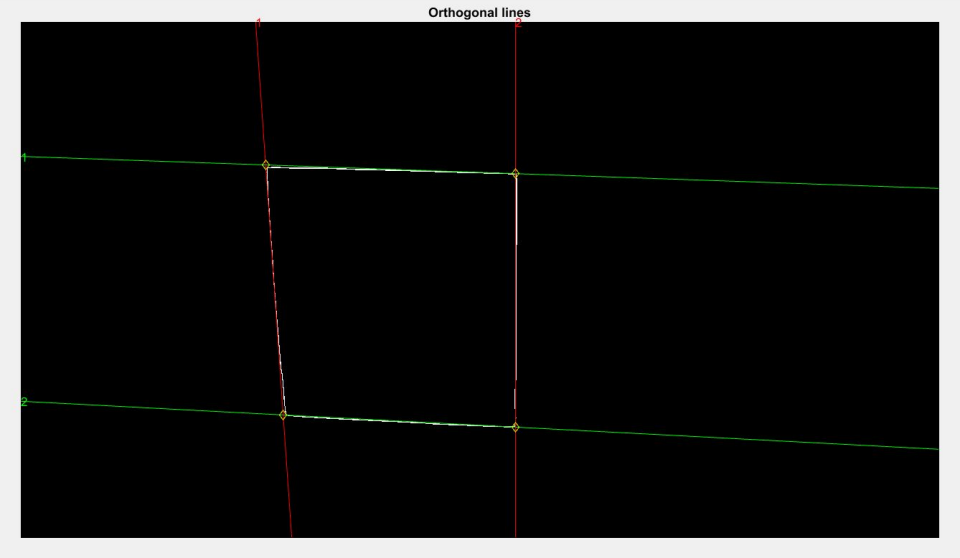


Figure 8. Shows only four lines that are best fits to the edges of the interior of the CCC window.

With these lines we were able to calculate the intersections using the *findIntersections()* portion of Professor Hoff's code. See figure 9 for the final set of CCC window corners.

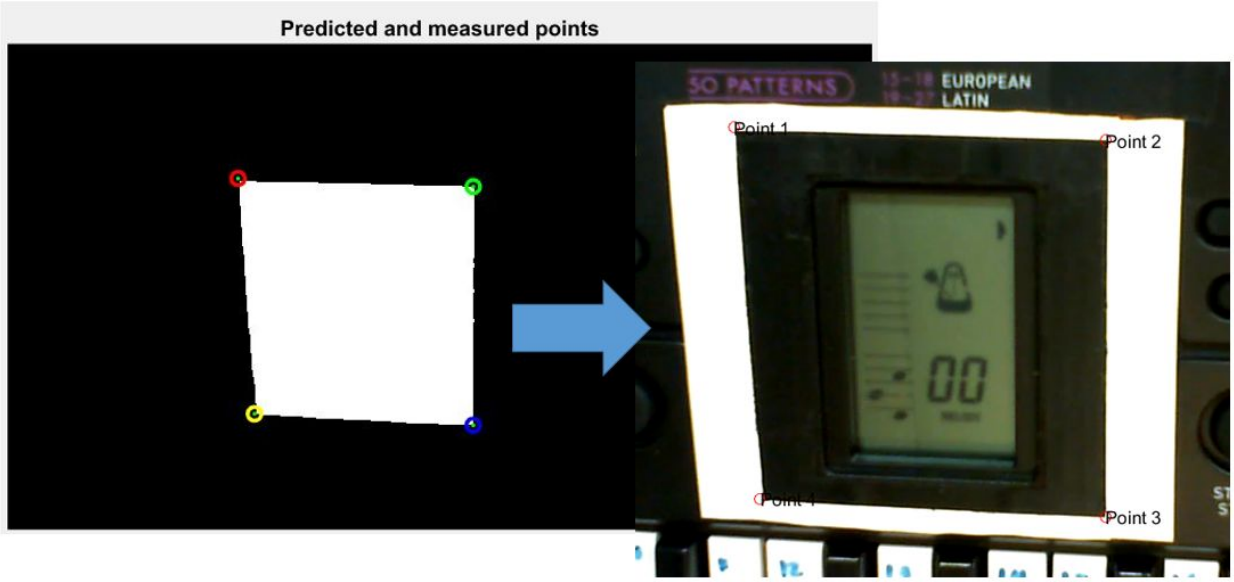


Figure 9. CORNERS

Figure 9. Shows the extracted CCC window corners. The red point is point 1, the green point 2, the blue is point 3 and the yellow point is point 4.

~~~Focusing in on LCD Region of Interest

The corners found in the last section can now be used to do an ortho correction to all incoming video frames. This correction will allow the user to see a normal view of the LCD screen despite the camera's viewing angle. The function, *ortho_LCD()*, will accomplish this task. MATLAB's *fitgeotrans()* and *imref2d()* were necessary to orthogonalize an incoming frame and then assign a viewing window that could stretch, compress and resize the transformed image. It is common that after *fitgeotrans()* is used the transformed image will either be too small or not in a region of interest. *imref2d()* gives the user the flexibility to calibrate the window bounds to only view the portion of the LCD screen shown in figure 12.

Figure 10. ROI

Figure 10. Shows a red dashed rectangle that outlines the Region of Interest (ROI) on the LCD screen. This region contains only the changing music notes while ignoring the metronome image shown on the right.

The region of interest now requires several layers of processing. First it needs to be converted to grayscale using Otsu's method. After that the noise needs to be cleaned up with an opening operation. However, the element size had to be limited to seven so that notes would not be washed out with the opening. Here the window specified by the region of interest shown in figure 10 will affect how Otsu's method threshold the notes. If more of white region remains in the ROI from a crude window definition the thresholding will completely wash out the notes on account of the the extra 'one' valued pixels that shift the histograms to favor a more binary image. This binary image will describe the black plastic around the LCD and the LCD's blank background while leaving out information on the musical notes because they are defined more as gray; not simply black or white. If drastic lighting or camera position changes occur new window values may need to be parameterized. The constants shown in figure 11 will need to be changed to allow for an acceptable ROI image. Use the values in figure 11 as nominal values and slightly increase or decrease them as needed to attain an acceptable ROI thresholded binary image

```
xmin = 21;
xmax = 34;
ymin = 12;
ymax = 60;
win = [xmin,xmax,ymin,ymax];
I_LCD = ortho_LCD(corners,I,win,1);
```

```
function [I_LCD] = ortho_LCD(Pimg2,I,win,debug)

% Compute transform, from corresponding control points
Tform2 = fitgeotrans(Pimg2,Pworld2,'projective');

ref2Doutput = imref2d(...
    [480,150],...
    [win(1),win(2)],...
    [win(3),win(4)]);
    % [image_height,image_width] -> [480,320],...
    % [xmin xmax]
    % [ymin ymax]
```

Figure 11. Shows that the window parameters are first changed in the *calibrate()* function and passed into the *ortho_LCD()* function. Here, they are passed into the *imref2d()* modifier function. This modifier allows the ROI window to be resized if a change to the experimental setup occurs.

Figure 12. Shows successive thresholding and opening to finally produce a reliable binary image of the music notes

~~Extracting Music Note Information

Once an image has been thresholded and turned into a binary image as shown in figure 12 the music note locations can be computed. The function, *get_coords()* takes in a binary image of the ROI and outputs the x,y coordinates of the notes. This was accomplished by using

MATLAB's *imfindcircles()* function. This function will cycle through an image to find the centroid of circular objects. The function needs only the diameters of the circles you wish to find and two confidence parameters that can be tuned. The parameters are 'sensitivity' and 'EdgeThreshold'. Setting these to 0.95 and 0.1 respectively set a lower expectation of perfect circularity while only looking for edges with high contrast. The centroids of the quasi circular notes are shown in the right-most image of figure 12. Often times there were two centroids calculated for a single note. To remedy this *globber_func5()* was used to combine nearby centroids into one average centroid. This averaged centroid is shown by the red stars in figure 13. Since the ROI has its own coordinate system the x and y coordinates don't have a global reference frame.

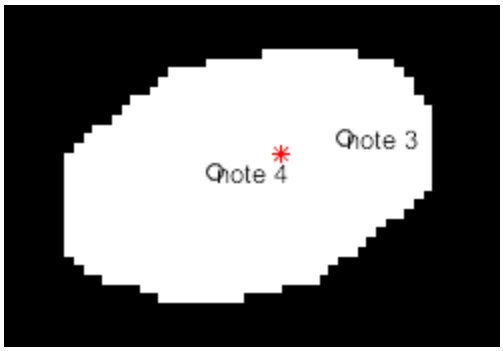


Figure 13. Shows two nearby note centroids being averaged into one.

~~Mapping Note Location to Music Note

Next, the calibration keys 1,4 and 7 were pressed. The three notes appeared on the LCD screen shown in figure 12 and their x,y, coordinates were extracted using the *get_coords()* function. These three locations are with respect to the local ROI frame and need to be mapped to musical notes. A detailed set of measurements was made that localized all dot positions to actual musical notes, the measurements can be found in Appendix [LCD MEASUREMENTS]. Once the distances between notes were known the program was able to map the ROI x,y coordinates to musical notes the overlaid plot is shown fully in Appendix [LCD NOTES] and partially in figure 14.

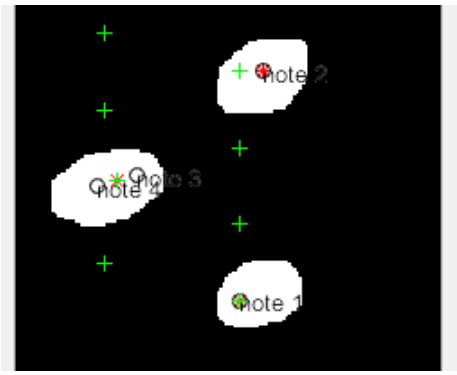


Figure 14 Shows green plus markers as the known x,y positions of each music note the LCD screen has the ability to display. It is clear how close each of the white music note "blobs" are to their respective music note locations.

Afterwards, our code used the *music_read()* function to cycle through each of the pressed notes to compare them to the known x,y music note positions. If the norm is within a tolerance MATLAB recognizes that particular body as seen figure 14 as a note and saves the piano key to an array to be used by the key highlighting part of the program later.

~~ Final Calibration Checks and Results

Once the code computes which piano key number corresponds to the binary image MATLAB outputs the prospective piano keys. If the calibration procedure recognized the correct three calibration keys you will get a success message, if not you will be prompted to retry the calibration procedure from the beginning of this section. After the calibration procedure we will now have two key files. One is the 'note' array which has the known x,y locations that MATLAB will be checking at every video frame for music note correspondence. The second is the CCC window corner points. The points will allow the program perform an orthophoto correction for every video frame which will allow for all aforementioned LCD screen analysis. Having these two arrays on hand drastically reduces computational time during the next video analysis section.

~Music Reader - Video Analysis

The video analysis section process all video frames after the initial calibration image. Thankfully the same steps will be taken to extract musical note and output the corresponding piano key press. The main difference will be that we already have the musical note lookup table within the 'note' array and the CCC window corner points. Figure 15 shows that the process in the video analysis section is just an automated version of the calibration stage. The final output of will be an array of all the piano keys to be pressed at 1 second intervals.

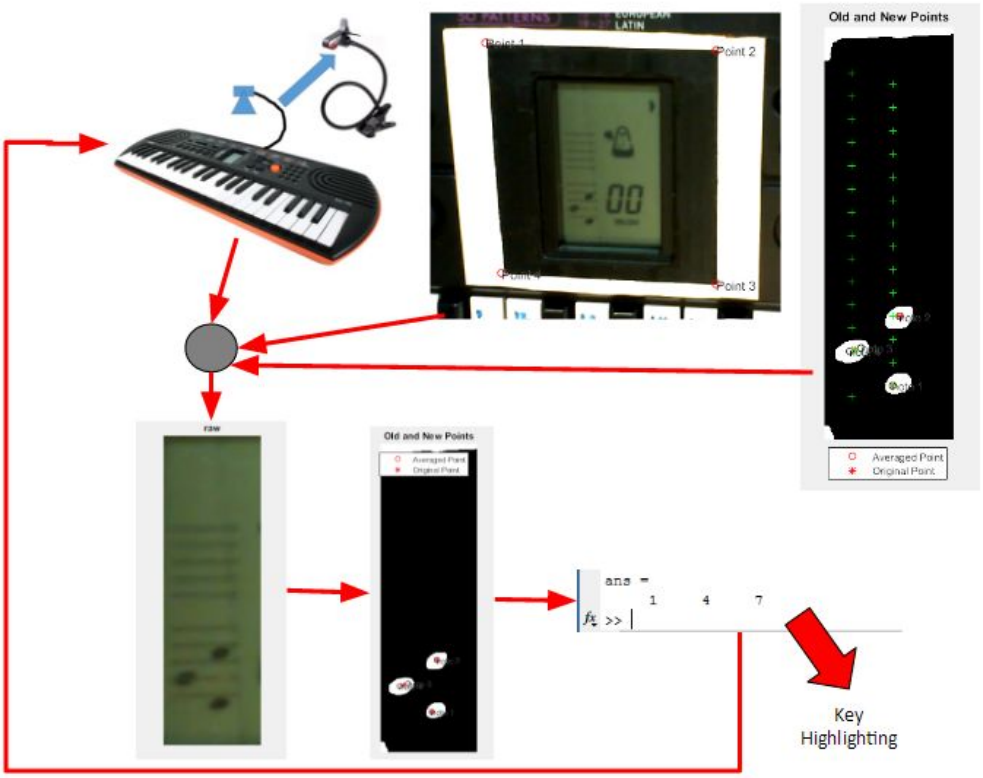


Figure 15. Shows that the CCC corner points along with the xy note lookup table enables the program to process, read music and output corresponding piano key presses for the remaining frames of a video.

~Key Highlighter

~~Find Corners of Keyboard

~~~Read First Frame

Since using Matlab to generate four corners of the keyboard by Hough Line can detect the keyboard in every frame at the cost of much more processing time, and it is also strongly influenced by the lighting condition, we came up with an idea to take advantage of the fact that the camera is stably fixed above the keyboard. In this case, we can get the first frame from the video and generate four corners of the keyboard for future use. Figure 16 shows the step to get position of the corners. We used and modified a MATLAB functions from the lecture called *findCheckerBoard()* that contains this entire process and can be found the Appendix [MATLAB FIND CHECKERBOARD NEW CODE].

Figure 16. KEYBOARD CORNERS

~~~Adjustment

The corners we get can be used in calculating the position of each key based on the knowledge we have. And a small adjustment is needed in the code because the key just below the camera appears to be a real rectangle but the keys on the two sides look like parallelograms. In order to give user a nicer and clear guidance, we made some adjustment.

~~~Save to arrays

In order to increase the speed of processing images in the video, we don't have to get four corners of the keyboard in every frame, so we create two arrays to save the position of each keys as outputs of this function and we use these two arrays to get the info of keys in the rest frames.

~~Highlight the keys for user and songs

~~~Convert time notes from time to frame

The information we get from the video reader is a note table based on time. Converting table notes to frame space make it easier for us to determine which keys should be highlighted in red color in every frame.

~~~Highlight keys with different colors

In the demo video, we use red rectangles to highlight keys which should be pressed from the information of a song. And we use green rectangles to highlight keys which are being pressed by user from the information collected from finger detector. We developed two MATLAB functions called *highlight_keys()* and *highlight_keys_for_song()* that contain this entire highlighting process and can be found the Appendix [MATLAB HIGHLIGHTING CODE].

~Finger Detector

~~Color Threshold

The reason why we choose the background which has the similar RGB color with our hands is to improve the robustness of our system. If it works in this background, it can also work in other environment. At the beginning, we collected color data of hand and background from different frames and chose the proper threshold trying to get rid of the background. Figure 17 shows how we get the information from the pictures. We developed a MATLAB function called *findskin()* that contains this entire color threshold and bw image creating process and can be found the Appendix [MATLAB FIND SKIN CODE].

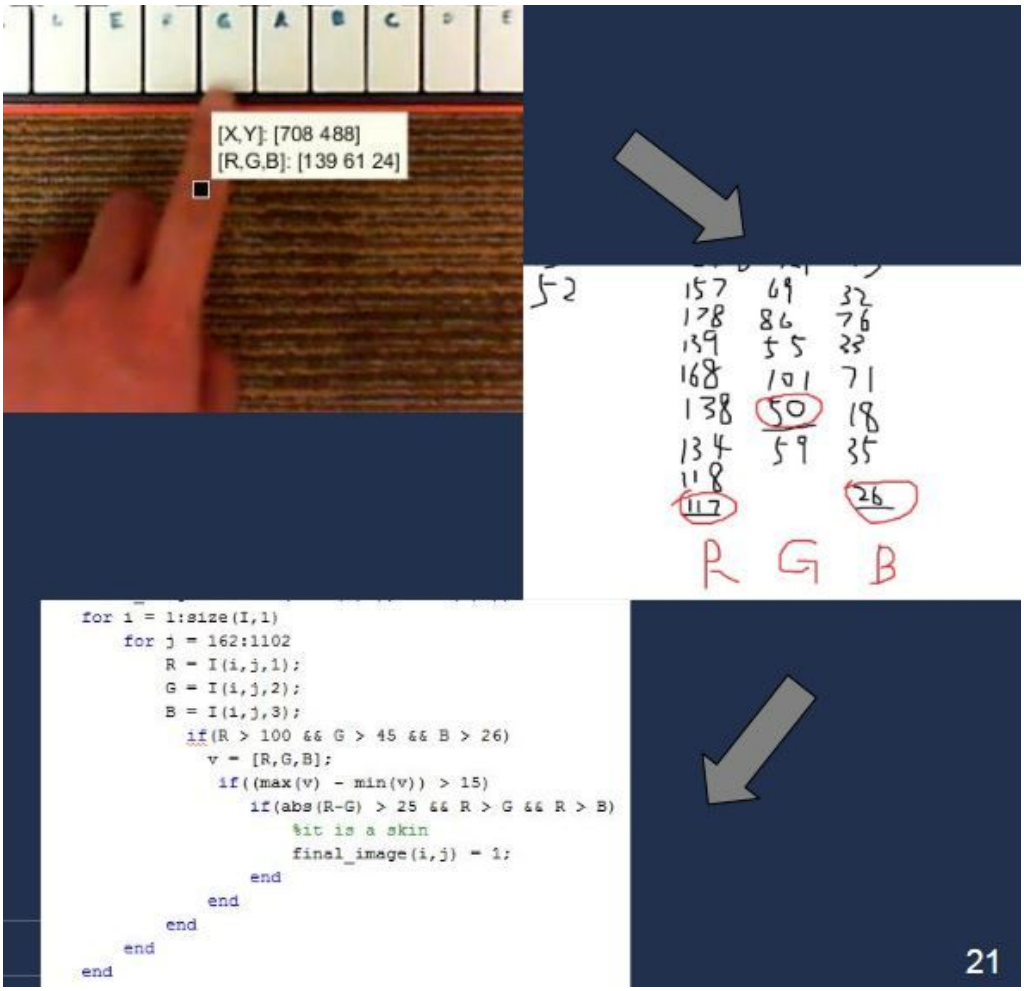


Figure 17. COLLECT RGB

But the only thing we were able to remove was the piano (we found it is very useful in the following steps). The result was not good because as we can see in figure 18, the blobs are always connected no matter how we try to use erosion and dilation to separate hands from the background.

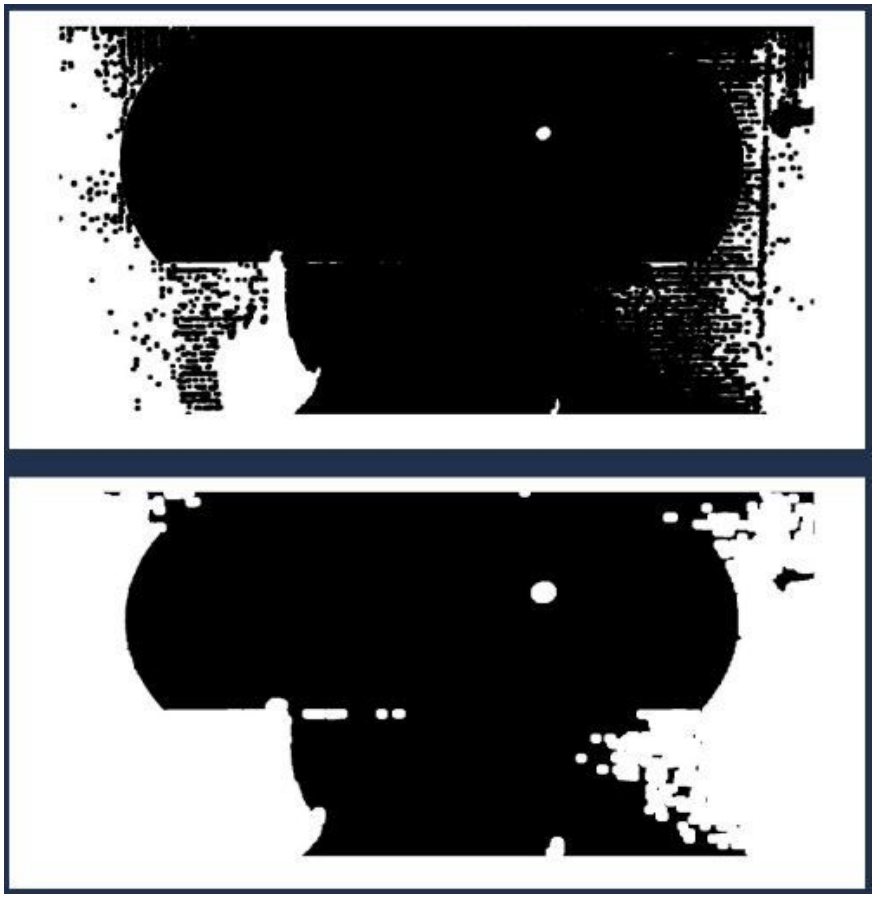


Figure 18. BAD RESULT

We came up with the idea that we could get rid of some parts of the background when creating bw image in the function *findskin().* Figure 19 shows that how we use the position from the keyboard we previously got to set the limitation in order to get rid of the noise. Figure 20 shows that the left and the right part of the noise has been removed.

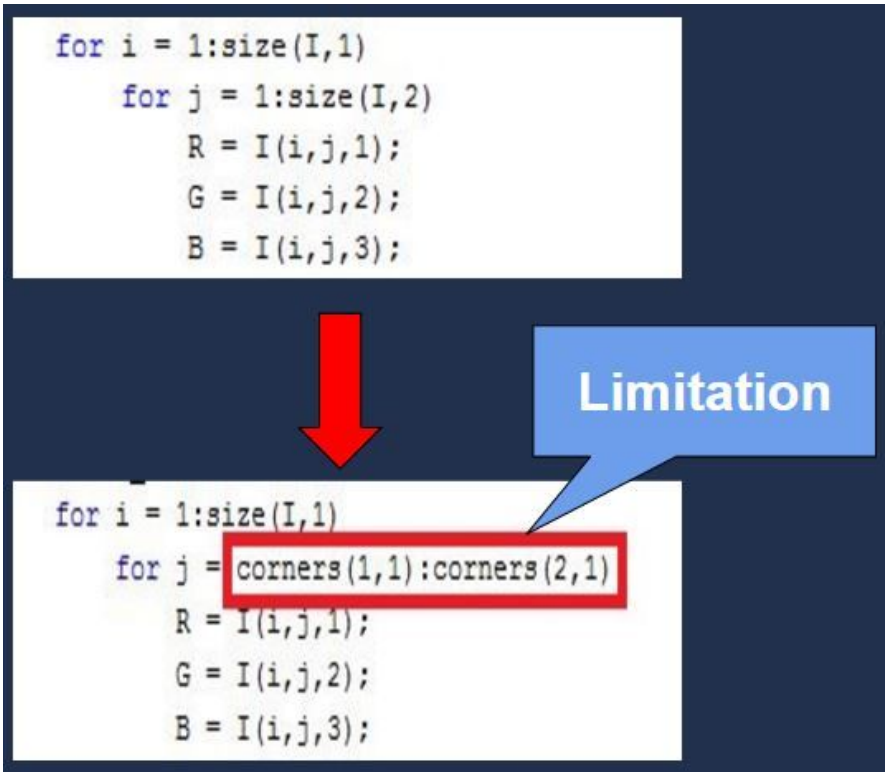


Figure 19. APPLY LIMITATION

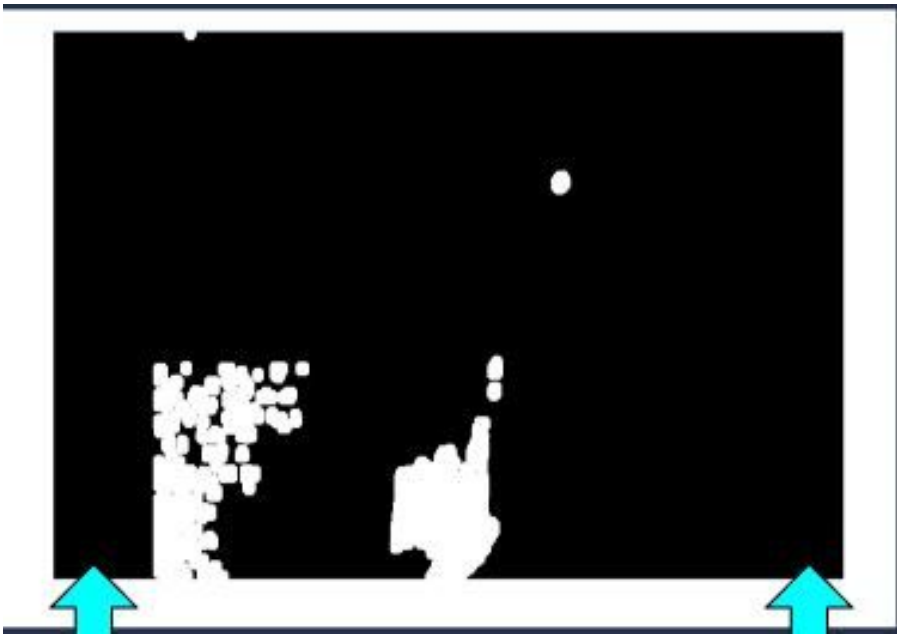


Figure 20. DESIRED RESULT

Since we get the desired result, we do some basic steps to get the different blobs, as we can see in the figure 21.

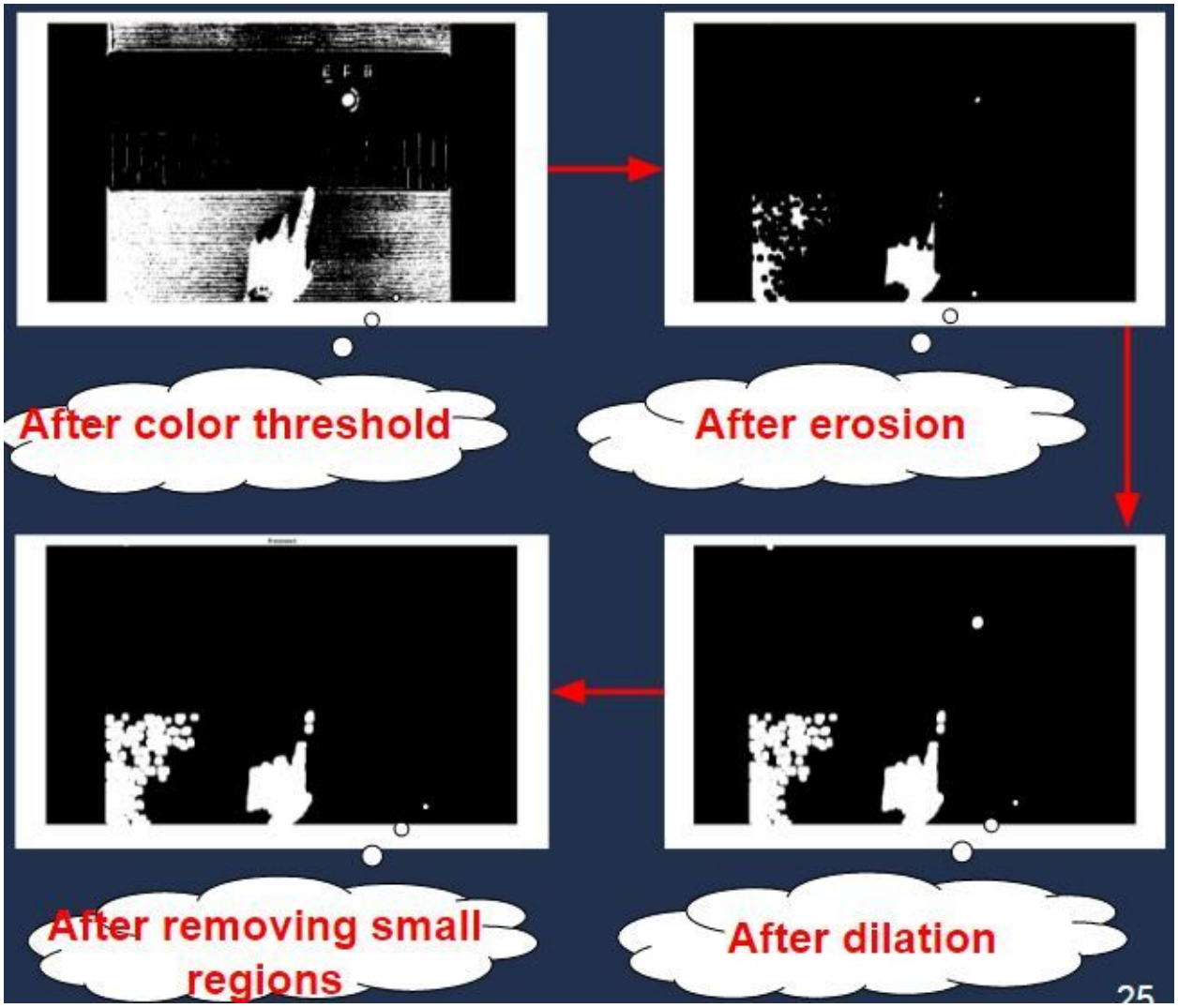


Figure 21. BASIC STEPS

~~Find Peaks

After erosion and dilation, we have several blobs in the image. We pick the highest peak in every boundary of the blob, as we can see in the figure 22.

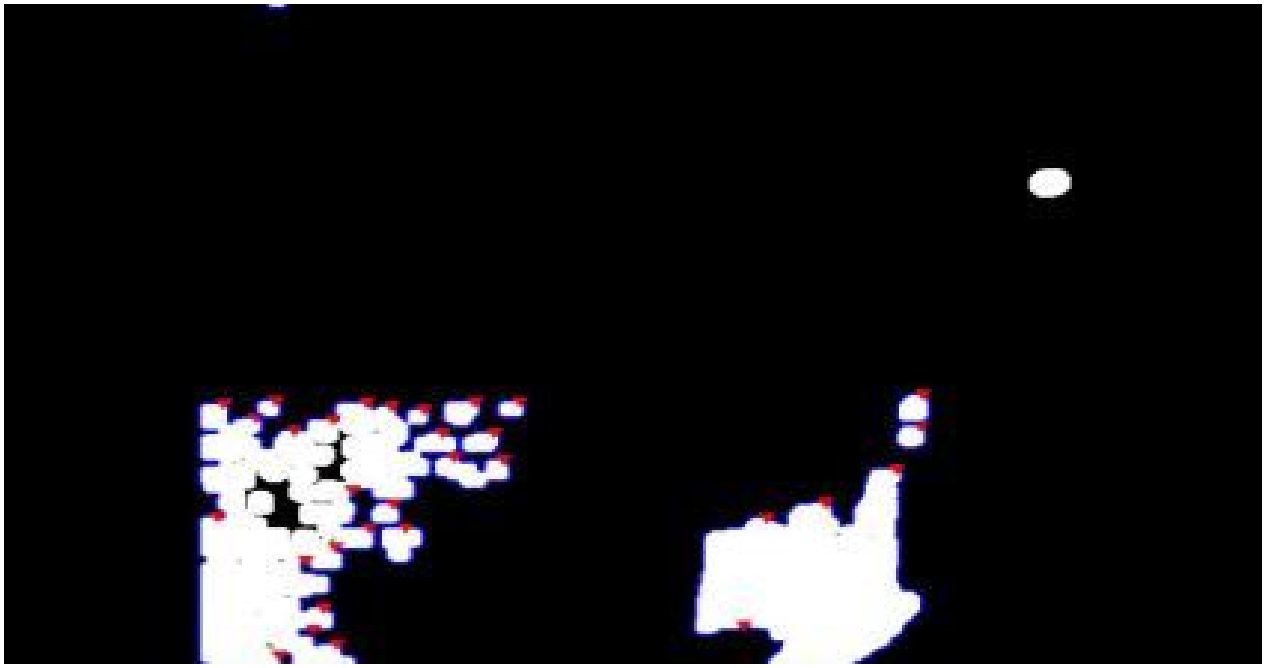


Figure 22. MULTIPLE PEAKS

Since we have the positions of the four corners of the keyboard, method like “Region of Interest” has been used here to detect if any peaks are inside of the four corners. Once detect the desired peaks inside of this region, we can consider it is a fingertip and use small triangles to mark them because after color threshold, we can make sure that there is no noise blobs on the keyboard region, as we can see in the figure 23. We developed a MATLAB function called *HandGesture()* that contains this entire finger detecting process and can be found the Appendix [MATLAB FINGER DETECTOR CODE].

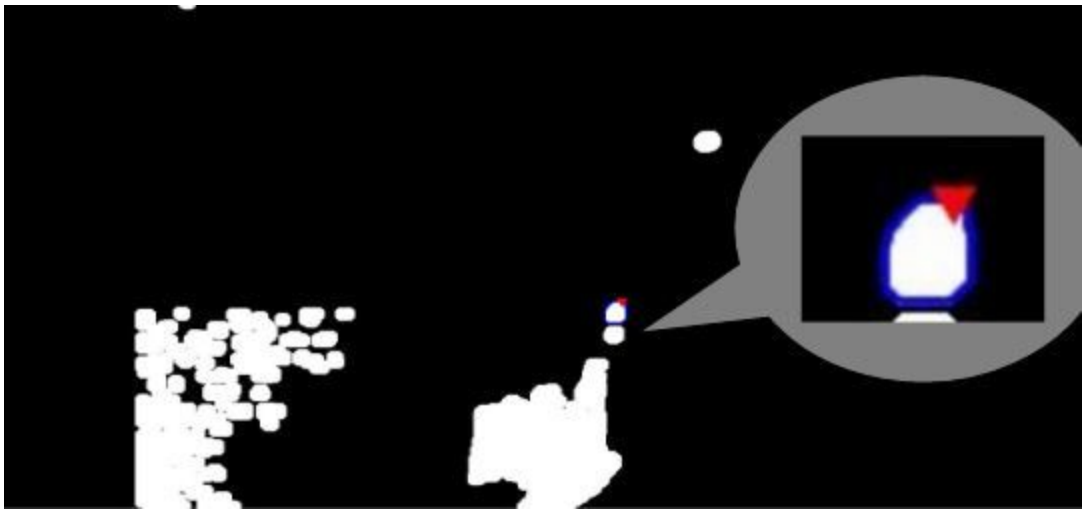


Figure 23. HIGHEST PEAK

EXPERIMENTS AND RESULTS

~LCD Window Thresholding

As mentioned in the Focusing in on LCD Region of Interest section the LCD window bounds play a key role in identifying music notes. We see in figure 24 that the size of the white border region will change the global threshold. Through experimentation it was it was concluded that 'xmin' played a large role and a well performing nominal value of 19 was found. If 'xmin' becomes too small more white will appear and washout notes but if it is too large the ROI will be clipped and information will be lost.

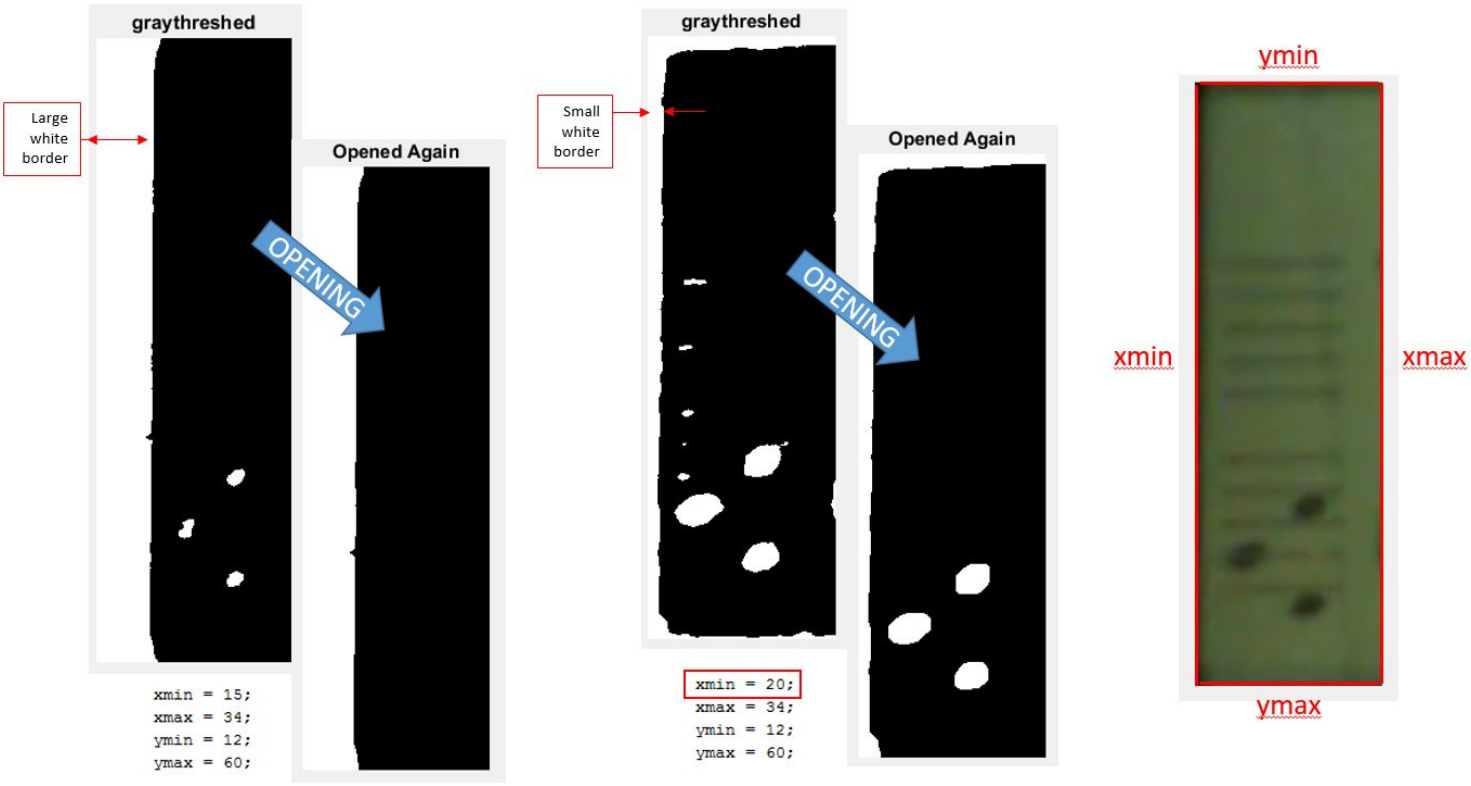


Figure 24 Shows the influence of window size on the thresholding of the music notes.

~Making a Piano Key Output Array

The final output from the music reading portion of the code is an array that has all the piano keys the user needs to press in order to play that particular song. Figures 25 shows the how well the program was able to recognize a musical note and output the correct piano key. *The accuracy of the note recognizer turned out to be about 57%.*

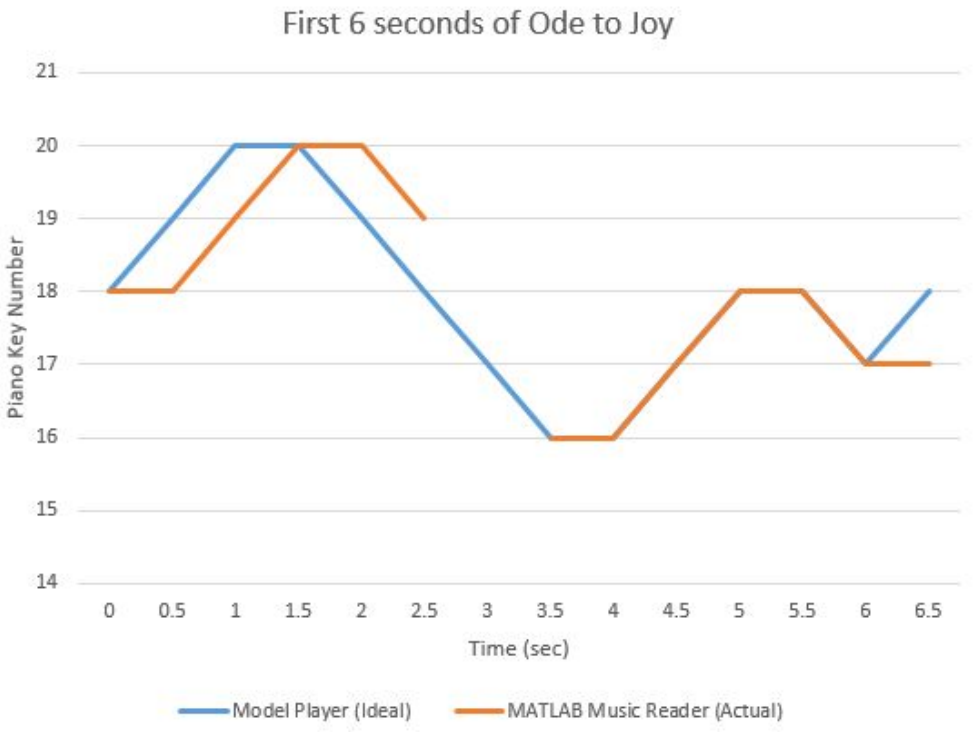


Figure 25 shows that the recognizer (MATLAB Music Reader)was more or less able to read each note at a sample rate of 2 hz.

~Highlighter and Finger Detector

The final output from the key highlighting and finger detecting portion of the code is as follows.

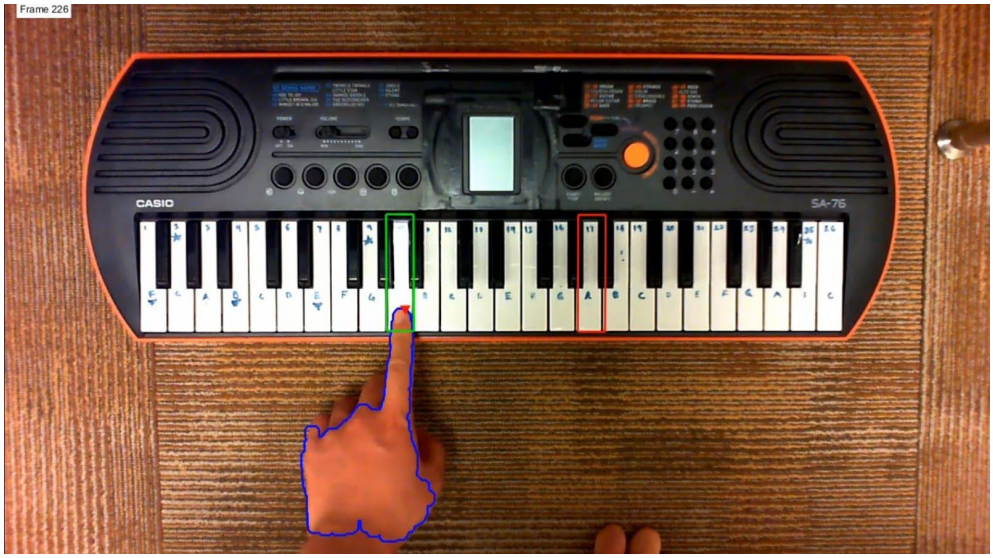


Figure 26 shows that the key highlighter and finger detector (MATLAB Music Reader)were able to detect and highlight keys in different colors and hands in blue boundaries and fingertips in small red triangles.

DISCUSSION

~Achievements

AS of now, the program can read incoming music notes, carry out finger detecting (as can be seen in the demo video), and highlight specified piano key. Basically, we get pretty good results but the program is not perfect.

~Improvements needed

~~Multiple Finger Support

We only tested the video with single finger tapping the key and we haven't tried multiple fingers. But our code is supposed to work well with multiple fingers since our code is to detect whichever blobs on the keyboard (we know the position of the four corners of the keyboard previously) and then pick the highest peak on every boundaries generated as user's fingers.

~~Robust GUI

An user-friendly interface is needed because we don't want user to be frustrated by switching between his hands and his camera. The proposed user interface can be seen in figure 26.

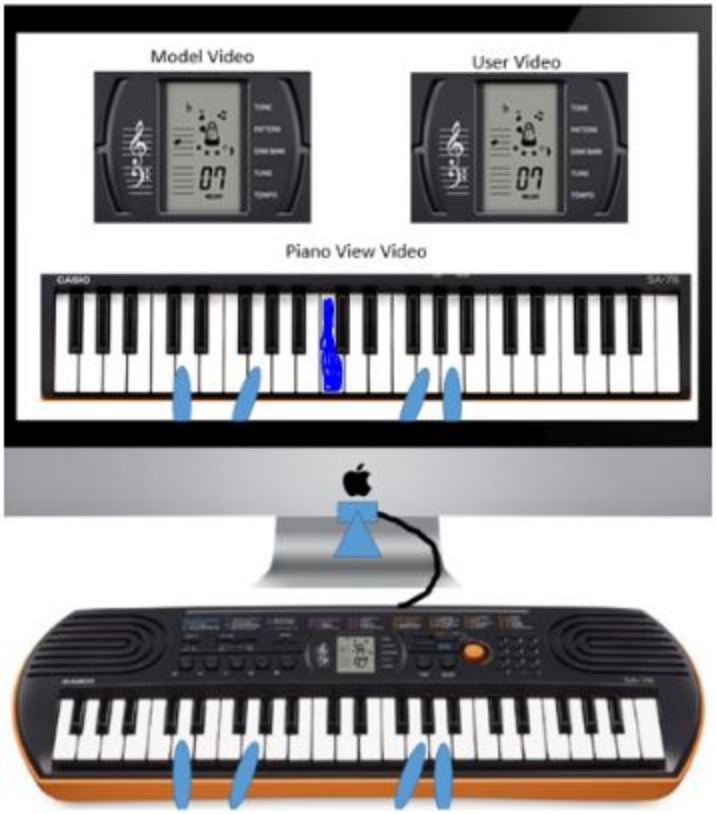


Figure 27 shows the keyboard underneath a monitor with two videos. The left video shows the model video that describes what notes should be played. The right video show what notes are actually being played.

~~Overall Design

Some animations like bars falling down in a certain speed can be applied to it in order to let user know how long these keys will be pressed.

~~Improving Note Recognition Accuracy

The reader may have noticed that the accuracy is lower than expected especially since the reader has the ability to discern notes with good accuracy during the calibration. The error comes from a simplified Model Player graph, this is the blue line in figure 25. For example the first two notes should be B notes, but the rapid succession of the notes was finer than 0.5 seconds. This caused us to average the first two notes together even though the actual song plays the the B note for about 0.5 second longer. This explains the lateral offset that the MATLAB reader produces from the Ideal player. Further development that synchronizes both the user clock and the piano clock would remedy this issue

APPENDIX

[MATLAB MAIN RUN FILE] *main()*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%
% Ryan Thorpe
% EENG 507 Computer Vision
% Final Project Piano Player
%
% >>> Main File
% Executes all sub-functions to read music and output song notes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%

clear cam
clc
close all
mode = 1; % 1 for Movie, 2 for stream
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
% Switch Between Video File Analysis and Webcam Stream
%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if mode == 1 %looks up video file and makes object
   movieObjInput = VideoReader('cal_ode.mp4'); % open file
%    movieObjInput = VideoReader('cal_jingle_bells.mp4'); % open file
   get(movieObjInput) % display all information about movie
   nFrames = movieObjInput.NumberOfFrames;
   cam = 0;
   n = 5; %n=15 for analysis every half seconf n=30 for every one second
   t = 1; %delta time for video frames (sec)
else

   total = 60; %sec
   cam = webcam(1);
   nFrames = total/t;
   n = 1;
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
% Calibration Procedure
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

%checks if A 'note' and window (win) variable are around---------
A = exist('note','var');
if A == 0
  [note,cal_keys,status,win,corners] = calibrate(mode,cam);
  disp(cal_keys);
  disp('Calibration...')
  disp(status);
  if strcmp(status,'FAILURE')
     disp('try again')
  else
     save('calibrate.mat','note','win')
  end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
% Frame by Frame Analysis of Video or Webcam
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

else %If there is a 'note' and 'win' variable continue with video
I = 1;
clear xy key;
tic
  for iFrame=1:n:nFrames
     if mode == 1 %video mode reads a video file
        I = read(movieObjInput,iFrame); % get one RGB image
     else %live stream mode reads a webcam stream
        I = snapshot(cam);
        pause(t-0.07); %takes about 1/10 sec to run above loop
     end
```

```
        I_LCD = ortho_LCD(corners,I,win,0); %Performs Ortho Correction
        xy = get_coords(I_LCD,0); %gets x,y coords of notes
        %combines nearby (duplicate) notes
        [xy_new,xy_old] = globber_func5(xy,0);
        [u,v]=size(xy_new);

        if u>0
            %translate x,y notes to piano key numbers
            intermediate = music_read(note,xy_new);
            gg = length(intermediate);
            key(I,1:gg) = intermediate;
        else %happens if no notes are on LCD
            intermediate = 0;
            key(I,:) = intermediate;
        end

        I = I+1;
        clear xy xy_new;
        disp(I);

    end
toc
end

clear cam
```

[MATLAB CAL CODE] *calibrate()*

```
function [note,cal_keys,status,win,corners] = calibrate(mode,cam)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%
% Ryan Thorpe
% EENG 507 Computer Vision
% Final Project Piano Player
%
% >>> Calibration File
% Executes all sub-functions to calibrate environmental factors to the
% LCD reader
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%
```

```
if mode == 2
    I = snapshot(cam);
else
    %read a video
    iFrame = 1; %read fisrt movie frame
    movieObjInput = VideoReader('cal_ode.mp4'); % open file
%     movieObjInput = VideoReader('cal_jingle_bells.mp4'); % open file
    get(movieObjInput) % display all information about movie
    nFrames = movieObjInput.NumberOfFrames;
    I = read(movieObjInput,iFrame); % get one RGB image
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
% Performs video analysis sequence
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

corners = get_LCD_corners(I,1); %Finds four inner corners of CCC window
%x and y bounds of the Region of Interest LCD window
xmin = 19;
xmax = 34;
ymin = 12;
ymax = 60;
win = [xmin,xmax,ymin,ymax];
I_LCD = ortho_LCD(corners,I,win,1); %Performs Ortho Correction
imshow(I_LCD);

xy = get_coords(I_LCD,1); %gets x,y coords of notes
%combines nearby (duplicate) notes
[xy,~] = globber_func5(xy,1);
note = xy2note(xy,1); %Makes the LCD music note lookup table
%compares extracted music notes to music note lookup table for mathces
cal_keys = music_read(note,xy);

%checks to see if calibration fails
check = isequal(cal_keys,[1;4;7]);
if check == 1
    status = 'SUCCESS';
```

```
else
  status = 'FAILURE';
end
```

[MATLAB FIND CHECKERBOARD] *findCheckerBoard()*

Only Modified to Suppress Plots See EENH 507 Website

[MATLAB MUSIC NOTE COORDINATE FINDER] *get_coords()*

```
function output = get_coords(I_LCD,debug)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%
% Ryan Thorpe
% EENG 507 Computer Vision
% Final Project Piano Player
%
% >>> Note Coordinate Calculator
% Uses Matlab's findfircles function to output the x,y postions in the
% Local ROI window of all recognizable music notes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%

[centers, ~, ~] = imfindcircles(I_LCD,[10 15],...
  'ObjectPolarity','bright','Sensitivity',.95,'EdgeThreshold',0.01);

output = centers;

if debug == 1 %debugging plots
  figure, imshow(I_LCD,[]);
  hold on;
  dim = size(output);
  if isempty(dim) == 0;
    plot(output(:,1),output(:,2),'ro')
      for i = 1:dim(1)
        ptlabel = ['note ', num2str(i)];
        text(output(i,1),output(i,2),ptlabel)
      end
  else
    output = [0,0];
```

```
    end
end
```

[MATLAB CCC WINDOW CORNER POINT EXTRACTOR] *get_LCD_corners()*

```
function corners = get_LCD_corners(I,debug)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%
% Ryan Thorpe
% EENG 507 Computer Vision
% Final Project Piano Player
%
% >>> CCC window Corner Extractor
% Uses MATLAB blobs to indentify target CCC window then uses Hough Lines
% to characterize the CCC windows inner edge. The intersections of these
% lines become the 4 ortho-photo corners
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%

%Cleans up an incoming thresholded image
B = im2bw(I,graythresh(I));
    SA = strel('disk',3);
    B2 = imopen(B, SA);
    B3 = imclose(B2,SA);
    B = B3;
    iB = ~B;
[L,num] = bwlabel(B,4);
[iL,inum] = bwlabel(iB,4);
if debug == 1 %debugging plots
    figure, imshow(I,[]); title('raw');
    figure, imshow(L,[]); title('blobs');
    figure, imshow(iL,[]); title('iblobs');
end

%Uses Matlab Blobs to label all bodies
blobs = regionprops(L,'Centroid',...
    'Perimeter','Area','BoundingBox');
iblobs = regionprops(iL,'Centroid',...
    'Perimeter','Area','BoundingBox');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%
%
% Checks bodies for Concentricity and Minimum Areas
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

e_match = 30; %maximum difference of concentric body centroids
k=0;
p_l = .8; %perimeter ratio upper bound
p_h = 1.3; %perimeter ratio lower bound

for i = 1:num
    for j = 1:inum
        k = k+1;
            norm_check(k) = ...
                norm((blobs(i).Centroid - iblobs(j).Centroid),2);
            if(norm_check(k) < e_match)
                if(iblobs(j).Area > 10000)
                    p_check = (blobs(i).Perimeter / iblobs(j).Perimeter);
                    if p_check > p_l && p_check < p_h
                        index = j;
                    end
                end
            end
    end
end


I_mask = masker(index,iL); %Simplified CCC window to a solid quadrilateral
if debug == 1 %debugging plots
    figure, imshow(I_mask,[]);
end

%Takes the Solid CCC window and adds hough lines along its 4 edges
%the intersections of these lines make up the 4 orthophoto corners
[corners, nMatches, avgErr] = findCheckerBoard(I_mask,0);

if debug == 1 %debugging plots
    output = corners;
    figure, imshow(I,[]);
    hold on;
    plot(output(:,1),output(:,2),'ro')
```

```
    for i = 1:length(corners)
        ptlabel = ['Point ', num2str(i)];
        text(corners(i,1),corners(i,2),ptlabel)
    end
    disp(corners);
end

end
```

[MATLAB MUSIC NOTE COMBINER] *globber_func5()*

```
function [xy_new,xy_old] = globber_func5(xy,debug)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ryan Thorpe
% EENG 507 Computer Vision
% Final Project Piano Player
%
% >>> Globber Function
% Checks all combinations of calculated x,y coordinates and averages
% together nearby points into one. This eliminates nearby duplicate notes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

xy_old = xy;
tol = 30; %distance between points on L2 norm

dim = size(xy);
%checks if input array is a single point. if so below is skipped
    if dim(1) > 1
        for i = 1:dim(1)
            for j = (i+1):dim(1)
             %does a routine neighbor check
             check = norm((xy(i,:))-(xy(j,:)),2);
             if check < tol
                    %neighbor found!
                    avx = (xy(i,1) + xy(j,1))/2; %find average x
                    avy = (xy(i,2) + xy(j,2))/2; %find average x
                    xy(i,:) = [avx, avy];
                    xy(j,:) = []; %clears (former) candidate point
                    dim = size(xy); %calculate new xy length
```

```
%                start =
                if dim(1) < j; break; end;
                if dim(1) == j; %does a last point neighbor check
                    check = norm((xy(i,:))-(xy(j,:)),2);
                    if check < tol
                        %neighbor found!
                        avx = (xy(i,1) + xy(j,1))/2; %find average x
                        avy = (xy(i,2) + xy(j,2))/2; %find average x
                        xy(i,:) = [avx, avy];
                        xy(j,:) = []; %clears (former) candidate point
                        dim = size(xy); %calculate new xy length
                        break;
                    end

                end
            end

        if dim(1) == j; break; end;
        end

        dim = size(xy); %calculate new xy length
        if dim(1) == 1; break; end; %breaks out at last point
    end


    if debug == 1 %debugging plots
        hold on
        plot(xy(:,1),xy(:,2),'r*',xy_old(:,1),xy_old(:,2),'ko')
        legend('Averaged Point','Original Point')
        title('Old and New Points')
    end

  end
  xy_new = xy;
end
```

[MATLAB CCC WINDOW MASKER] *masker()*

```
function m_img = masker(index,L)
  dims = size(L);
  m_img = zeros(dims);
  for i = 1:dims(2)
```

```
      for j = 1:dims(1)
          if(L(j,i) == index)
              m_img(j,i) = 1;
          end
      end
  end

% m_img = logical(m_img_double);
% imshow(m_img);
end
```

[MATLAB MUSIC NOTE TO PIANO KEY NUMBER] *music_read()*

```
function key = music_read(note,xy)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%
% Ryan Thorpe
% EENG 507 Computer Vision
% Final Project Piano Player
%
% >>> Music Reader
% Compares the recognized note x,y locations with the lookup table
% contained in the 'note' array
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%

dim = size(xy);
dim_n = size(note);
npn = dim(1); %number of played notes
n_t = dim_n(1); %total number of notes piano can play

tol = 20; %note camera placement error (pixels)
k = 1;
key = 0;
for i = 1:npn

  for j = 1:n_t

      if norm(xy(i,:)-note(j,:),2) < tol
          %you have a match!
          key(k) = j;
          k = k + 1;
```

```
      break
    end

  end

end

key = sort(key);
key = key';
```

[MATLAB ORTHO PROJECTION CORRECTION] *ortho_LCD()*

```
function [I_LCD] = ortho_LCD(Pimg2,I,win,debug)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%
% Ryan Thorpe
% EENG 507 Computer Vision
% Final Project Piano Player
%
% >>> Ortho Normal Correction or Images
% Uses the 4 corners calculated from the solid CCC window to perform an
% Orthophto warp on incoming images
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%

% Define location of control points in the world (in cm).
% Actual dimensions of Solid CCC window
Pworld2 = [
  0, 0; % X,Y units in cm
  67.08, 0;
  67.08, 72.15;
  0, 72.15;
  ];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
% Region of Interest Specification
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```
% Compute transform, from corresponding control points
Tform2 = fitgeotrans(Pimg2,Pworld2,'projective');
%Allows user to specify the output image dimensions and REGION OF INTEREST
ref2Doutput = imref2d(...
   [480,150],...
   [win(1),win(2)],...
   [win(3),win(4)]);
   % [image_height,image_width] -> [480,320],...
   % [xmin xmax]
   % [ymin ymax]

% Transform input image to output image
Iout2 = imwarp(I,Tform2,'OutputView', ref2Doutput);

if debug == 1 %debugging plots
   figure, imshow(Iout2,[]);
   title('raw')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
% Parameters for Upper LCD Thresholding
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

dim = size(Iout2);
Iout3 = logical(dim(1:2));
percent_y = 15; %upper LCD extends 15% downwards in ROI
percent_x = 5; %upper LCD extends 5% to the right in ROI
n = round(dim(1)*percent_y/100,0);
m = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
% Various Openings and Closings to Form Music Notes
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

% open  -> black spreads
```

```
% close -> white spreads

%Use to open top 1/4 of image Image----------------
Iout2top = Iout2(1:n,m:end,:);
level = graythresh(Iout2top);
Iout3top = im2bw(Iout2top,level);
Iout3top = ~Iout3top;
if debug == 1 %debugging plots
   figure, imshow(Iout3top,[]);
   title('TOP graythreshed')
end
   %use to open top portion of image
   SA_open = strel('disk',7);
   Iout3top = imopen(Iout3top, SA_open);
   if debug == 1 %debugging plots
      figure, imshow(Iout3top,[]);
      title('Top Opened')
   end

%Graythresh Rest of Image---------------------------------
level2 = graythresh(Iout2);
Iout3 = im2bw(Iout2,level2);
Iout3 = ~Iout3;
if debug == 1 %debugging plots
   figure, imshow(Iout3,[]);
   title('graythreshed')
end
%Put together top and bottom thresholded images---------
Iout3(1:n,m:end) = Iout3top;
if debug == 1 %debugging plots
   figure, imshow(Iout3,[]);
   title('top and bottom put together')
end

% % Use to close Rest of Image----------------
% SA_close = strel('disk',7);
% Iout3 = imclose(Iout3,SA_close);
% figure, imshow(Iout3,[]);
% title('Closed')

%Use to open Image----------------
SA_open2 = strel('disk',7);
Iout3 = imopen(Iout3, SA_open2);
```

```
if debug == 1 %debugging plots
    figure, imshow(Iout3,[]);
    title('Opened Again')
end
I_LCD = Iout3;
if debug == 1 %debugging plots
    figure, imshow(I_LCD,[]);
    title('Final')
end
end
```

[MATLAB MUSIC NOTE CORRESPONDENCE CHECK] *xy2note()*

```
function note = xy2note(xy,debug)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%
% Ryan Thorpe
% EENG 507 Computer Vision
% Final Project Piano Player
%
% >>> XY coordinates to Music Notes
% Uses the known 3 calibration note blobs
% to derive positions for the rest of the notes
% outputs a 28 x 2 array with all note positions on LCD screen
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%

[n_note,~] = size(xy);
note = zeros(n_note,2);
%sorts and inserts calibrations notes in location matrix
%calibration notes are keys 1 and 9
[~,I]=sort(xy(:,2),'descend');
B=xy(I,:); %use the row indices from sort() to sort all row of A.
note(1,:) = B(1,:);
note(4,:) = B(2,:);
note(7,:) = B(3,:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
% Key Variable Parameterization Based Off Measured Values
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%

a = (B(1,2)-B(3,2))/3;
av_hx1 = (B(1,1)+B(3,1))/2;
f = av_hx1 - B(2,1);
e = a/2;

ox = B(1,1); %x origin point (first note x-coord)
oy = B(1,2);  %y origin point (first note y-coord)

% note(x,:) = [(origin_x),(origin_y)];
% note(x,:) = [   (ox)   ,   (oy)   ];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
% Piano Key Positions Based Off Measured Values
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

%note 1 see line 19 above
note(2,:) = [(ox - f),(oy - e)];
note(3,:) = [(ox),(oy-a)];
%note 4 see line 20 above

note(5,:) = [(ox),(oy-2*a)];
note(6,:) = [(ox-f),(oy-e-2*a)];
note(7,:) = [(ox),(oy-3*a)];
note(8,:) = [(ox-f),(oy-e-3*a)];

note(9,:) = [(ox),(oy-4*a)];
note(10,:) = [(ox-f),(oy-e-4*a)];
note(11,:) = [(ox),(oy-5*a)];
note(12,:) = [(ox-f),(oy-e-5*a)];

note(13,:) = [(ox),(oy-6*a)];
note(14,:) = [(ox-f),(oy-e-6*a)];
note(15,:) = [(ox),(oy-7*a)];
note(16,:) = [(ox-f),(oy-e-7*a)];

note(17,:) = [(ox),(oy-8*a)];
note(18,:) = [(ox-f),(oy-e-8*a)];
```

```
note(19,:) = [(ox),(oy-9*a)];
note(20,:) = [(ox-f),(oy-e-9*a)];

note(21,:) = [(ox),(oy-10*a)];
note(22,:) = [(ox-f),(oy-e-10*a)];
note(23,:) = [(ox),(oy-11*a)];
note(24,:) = [(ox-f),(oy-e-11*a)];

note(25,:) = [(ox),(oy-12*a)];
note(26,:) = [(ox-f),(oy-e-12*a)];
note(27,:) = [(ox-f),(oy-e-13*a)]; %special 'sharp' character
note(28,:) = [(ox),(oy-13*a)]; %special 'flat' character

if debug == 1
   hold on
   plot(note(:,1),note(:,2),'g+')
end

end
```

[MATLAB MAIN RUN FILE FOR KEY HIGHTLIGHTER AND FINGER DETECTOR]
main_for_highlight_and_hand()

```
clear all
close all
clc
format short g;
movieObj = VideoReader('FINGER2.mp4'); % read movie
nFrames = movieObj.NumberOfFrames; % get number of frames
fprintf('Opening movie file with %d images\n', nFrames); % print number of frames in this movie
load TIME_NOTES_1_SEC.mat %get the time notes
movieObjOutput = VideoWriter('myResults.mp4', 'MPEG-4'); % create file for output
open(movieObjOutput); %Open output movie.
whitekeyinitial=0; %initialize everything
blackkeyinitial=0;
whichkeys=0;
for iFrame=1:1:nFrames
   I = read(movieObj,iFrame); % read image from movie
   figure(1), imshow(I), title(sprintf('Frame %d', iFrame)); % print current frame number in the
title

  % PRINT FRAME NUMBER ON THE IMAGE
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   text(20,10,sprintf('Frame %d', iFrame), ... % Also print the frame number onto the image
    'BackgroundColor', 'w', ...
    'FontSize', 10); % Default = 10
  % FOR INITIALIZING THE POSITION OF THE KEYBOARD AND KEYS

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   if iFrame==1

[whitekeyinitial,blackkeyinitial]=highlight_keys(I,iFrame,whichkeys,whitekeyinitial,blackkeyinitial);
% highlight keys being pressed and the right ones
   end
   % FOR FINDING THE HIGHEST FINGER TIP IN THE IMAGE

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   [fingertipx, fingertipy, hand5] = HandGesture(I); % detect hands and draw boundary and
fingertips and return the position of figertips
   fingertipx=cell2mat(fingertipx);
   fingertipy=cell2mat(fingertipy);
   [u,v]=size(fingertipy);
   if u>1 % find the highest finger tip among the finger tips we have found
      [fingertipy,maxidx]=max(fingertipy);
      fingertipx=fingertipx(maxidx,1);
   end
  % FOR FINDING WHICH KEY IS BEING PRESSED

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  if ~isempty(fingertipx)
  for i=1:26
      if fingertipx > whitekeyinitial{i,1}(1) && fingertipx < whitekeyinitial{i,2}(1) && ...
         fingertipx > whitekeyinitial{i,4}(1) && fingertipx < whitekeyinitial{i,3}(1)
             whichkeys=i;
             highlight_keys(I,iFrame,whichkeys,whitekeyinitial,blackkeyinitial);
      end
```

```
  end
  end
   % HIGHLIGHT THE "RIGHT" KEYS FROM THE SONG

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
   if 15+ceil(iFrame/30)<59
   whichkeys=key(15+ceil(iFrame/30),1);
   highlight_keys_for_song(I,iFrame,whichkeys,whitekeyinitial,blackkeyinitial);
   end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
   newFrameOut = getframe; % write video
   writeVideo(movieObjOutput,newFrameOut);
   pause(0.1);
end
close all
close(movieObjOutput); % all done, close file
```

[MATLAB FIND CHECKERBOARD NEW CODE] *findCheckerBoard()*

```
function [corners, nMatches, avgErr, a1, a2] = findCheckerBoard(I)
% Find a 26x1 checkerboard in the image I.
% Returns:
%   corners: the locations of the four outer corners as a 4x2 array, in
%       the form [ [x1,y1]; [x2,y2]; ... ].
%   nMatches:  number of matching points found (ideally is 81)
%   avgErr:  the average reprojection error of the matching points
% Return empty if not found.

corners = [];
nMatches = [];
avgErr = [];

if size(I,3)>1
   I = rgb2gray(I);
end
```

```
% Do edge detection.
[~,thresh] = edge(I, 'canny');      % First get the automatic threshold
E = edge(I, 'canny', 5*thresh);     % Raise the threshold

%figure(10), imshow(E), title('Edges');


% Do Hough transform to find lines.
[H,thetaValues,rhoValues] = hough(E);


% Extract peaks from the Hough array H.  Parameters for this:
%   houghThresh:  Minimum value to be considered a peak.  Default
%       is 0.5*max(H(:))
%   NHoodSize:  Size of suppression neighborhood.  Default is
%       [size(H,1)/50, size(H,2)/50].  Must be odd numbers.
myThresh = ceil(0.05*max(H(:)));
NHoodSize = ceil([size(H,1)/50, size(H,2)/50]);
% Force odd size
if mod(NHoodSize(1),2)==0  NHoodSize(1) = NHoodSize(1)+1;  end
if mod(NHoodSize(2),2)==0  NHoodSize(2) = NHoodSize(2)+1;  end
peaks = houghpeaks(H, ...
   20, ...             % Maximum number of peaks to find
   'Threshold', myThresh, ...      % Threshold for peaks
   'NHoodSize', NHoodSize);    % Default = floor(size(H)/50);

% Display Hough array and draw peaks on Hough array.
% figure(11), imshow(H, []), title('Hough'), impixelinfo;
% for i=1:size(peaks,1)
%     rectangle('Position', ...
%         [peaks(i,2)-NHoodSize(2)/2, peaks(i,1)-NHoodSize(1)/2, ...
%         NHoodSize(2), NHoodSize(1)], 'EdgeColor', 'r');
% end

% Show all lines.
% figure(10), imshow(E), title('All lines');
% drawLines( ...
%     rhoValues(peaks(:,1)), ...      % rhos for the lines
%     thetaValues(peaks(:,2)), ...    % thetas for the lines
%     size(E), ...                    % size of image being displayed
%     'y');
```

```
% Find two sets of orthogonal lines.
[lines1, lines2, a1, a2] = findOrthogonalLines( ...
   rhoValues(peaks(:,1)), ...      % rhos for the lines
   thetaValues(peaks(:,2)));       % thetas for the lines

 % Sort the lines, from top to bottom (for horizontal lines) and left to
% right (for vertical lines).
lines1 = sortLines(lines1, size(E));
lines2 = sortLines(lines2, size(E));

% Show the two sets of lines.
% figure(12), imshow(E), title('Orthogonal lines');
% drawLines( ...
%     lines1(2,:), ...    % rhos for the lines
%     lines1(1,:), ...    % thetas for the lines
%     size(E), ...        % size of image being displayed
%     'g');               % color of line to display
% drawLines( ...
%     lines2(2,:), ...    % rhos for the lines
%     lines2(1,:), ...    % thetas for the lines
%     size(E), ...        % size of image being displayed
%     'r');               % color of line to display

% Intersect every pair of lines, one from set 1 and one from set 2.
% Output is the x,y coordinates of the intersections:
%   xIntersections(i1,i2): x coord of intersection of i1 and i2
%   yIntersections(i1,i2): y coord of intersection of i1 and i2
[xIntersections, yIntersections] = findIntersections(lines1, lines2);

% Plot all measured intersection points.
% hold on
% plot(xIntersections(:),yIntersections(:),'yd');
% hold off

% Define a "reference" image.
IMG_SIZE_REF = 100;      % Reference image is IMG_SIZE_REF x IMG_SIZE_REF

% Get predicted intersections of lines in the reference image.
[xIntersectionsRef, yIntersectionsRef] = createReference(IMG_SIZE_REF);

% Find the best correspondence between the points in the input image and
% the points in the reference image.  If found, the output is the four
% outer corner points from the image, represented as a a 4x2 array, in the
```

```
% form [ [x1,y1]; [x2,y2]; ... ].
[corners, nMatches, avgErr] = findCorrespondence( ...
   xIntersections, yIntersections, ...         % Input image points
   xIntersectionsRef, yIntersectionsRef, ...   % Reference image points
   I);


end




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function drawLines(rhos, thetas, imageSize, color)
% % This function draws lines on whatever image is being displayed.
% % Input parameters:
% %   rhos,thetas: representation of the line (theta in degrees)
% %   imageSize:  [height,width] of image being displayed
% %   color:  color of line to draw
% %
% % Equation of the line is rho = x cos(theta) + y sin(theta), or
% %    y = (rho - x*cos(theta))/sin(theta)
%
% for i=1:length(thetas)
%     if abs(thetas(i)) > 45
%         % Line is mostly horizontal.  Pick two values of x,
%         % and solve for y = (-ax-c)/b
%         x0 = 1;
%         y0 = (-cosd(thetas(i))*x0+rhos(i))/sind(thetas(i));
%         x1 = imageSize(2);
%         y1 = (-cosd(thetas(i))*x1+rhos(i))/sind(thetas(i));
%     else
%         % Line is mostly vertical. Pick two values of y,
%         % and solve for x = (-by-c)/a
%         y0 = 1;
%         x0 = (-sind(thetas(i))*y0+rhos(i))/cosd(thetas(i));
%         y1 = imageSize(1);
%         x1 = (-sind(thetas(i))*y1+rhos(i))/cosd(thetas(i));
%     end
%
%     line([x0 x1], [y0 y1], 'Color', color);
```

```
%     text(x0,y0,sprintf('%d', i), 'Color', color);
% end
%
% end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Find two sets of orthogonal lines.
% Inputs:
%   rhoValues:  rho values for the lines
%   thetaValues:  theta values (should be from -90..+89 degrees)
% Outputs:
%   lines1, lines2:  the two sets of lines, each stored as a 2xN array,
%       where each column is [theta;rho]
function [lines1, lines2, a1, a2] = findOrthogonalLines( ...
   rhoValues, ...      % rhos for the lines
   thetaValues)        % thetas for the lines

% Find the largest two modes in the distribution of angles.
bins = -90:10:90;       % Use bins with widths of 10 degrees
[counts, bins] = histcounts(thetaValues, bins);    % Get histogram
[~,indices] = sort(counts, 'descend');

% The first angle corresponds to the largest histogram count.
a1 = (bins(indices(1)) + bins(indices(1)+1))/2;     % Get first angle

% The 2nd angle corresponds to the next largest count.  However, don't
% find a bin that is too close to the first bin.
for i=2:length(indices)
   if (abs(indices(1)-indices(i)) <= 2) || ...
         (abs(indices(1)-indices(i)+length(indices)) <= 2) || ...
         (abs(indices(1)-indices(i)-length(indices)) <= 2)
      continue;
   else
      a2 = (bins(indices(i)) + bins(indices(i)+1))/2;
      break;
   end
end
%fprintf('Most common angles: %f and %f\n', a1, a2);

% Get the two sets of lines corresponding to the two angles.  Lines will
% be a 2xN array, where
```

```
%   lines1[1,i] = theta_i
%   lines1[2,i] = rho_i
lines1 = [];
lines2 = [];
for i=1:length(rhoValues)
   % Extract rho, theta for this line
   r = rhoValues(i);
   t = thetaValues(i);

   % Check if the line is close to one of the two angles.
   D = 25;     % threshold difference in angle
   if abs(t-a1) < D || abs(t-180-a1) < D || abs(t+180-a1) < D
      lines1 = [lines1 [t;r]];
   elseif abs(t-a2) < D || abs(t-180-a2) < D || abs(t+180-a2) < D
      lines2 = [lines2 [t;r]];
   end
end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Sort the lines.
% If the lines are mostly horizontal, sort on vertical distance from yc.
% If the lines are mostly vertical, sort on horizontal distance from xc.
function lines = sortLines(lines, sizeImg)

xc = sizeImg(2)/2;  % Center of image
yc = sizeImg(1)/2;

t = lines(1,:);     % Get all thetas
r = lines(2,:);     % Get all rhos

% If most angles are between -45 .. +45 degrees, lines are mostly
% vertical.
nLines = size(lines,2);
nVertical = sum(abs(t)<45);
if nVertical/nLines > 0.5
   % Mostly vertical lines.
   dist = (-sind(t)*yc + r)./cosd(t) - xc;  % horizontal distance from center
else
   % Mostly horizontal lines.
   dist = (-cosd(t)*xc + r)./sind(t) - yc;  % vertical distance from center
```

```
end

[~,indices] = sort(dist, 'ascend');
lines = lines(:,indices);

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Intersect every pair of lines, one from set 1 and one from set 2.
% Output arrays contain the x,y coordinates of the intersections of lines.
%   xIntersections(i1,i2): x coord of intersection of i1 and i2
%   yIntersections(i1,i2): y coord of intersection of i1 and i2
function [xIntersections, yIntersections] = findIntersections(lines1, lines2)
N1 = size(lines1,2);
N2 = size(lines2,2);

xIntersections = zeros(N1,N2);
yIntersections = zeros(N1,N2);
for i1=1:N1
   % Extract rho, theta for this line
   r1 = lines1(2,i1);
   t1 = lines1(1,i1);

   % A line is represented by (a,b,c), where ax+by+c=0.
   % We have r = x cos(t) + y sin(t), or  x cos(t) + y sin(t) - r = 0.
   l1 = [cosd(t1); sind(t1); -r1];

   for i2=1:N2
      % Extract rho, theta for this line
      r2 = lines2(2,i2);
      t2 = lines2(1,i2);

      l2 = [cosd(t2); sind(t2); -r2];

      % Two lines l1 and l2 intersect at a point p where p = l1 cross l2
      p = cross(l1,l2);
      p = p/p(3);

      xIntersections(i1,i2) = p(1);
      yIntersections(i1,i2) = p(2);
   end
```

```
end

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Get predicted intersections of lines in the reference image.
function [xIntersectionsRef, yIntersectionsRef] = createReference(sizeRef)

sizeSquare = sizeRef/8;     % size of one square

% Predict all line intersections.
[xIntersectionsRef, yIntersectionsRef] = meshgrid(1:9, 1:9);
xIntersectionsRef = (xIntersectionsRef-1)*sizeSquare + 1;
yIntersectionsRef = (yIntersectionsRef-1)*sizeSquare + 1;

% Draw reference image.
Iref = zeros(sizeRef+1, sizeRef+1);
% figure(13), imshow(Iref), title('Reference image');
%
% % Show all reference image intersections.
% hold on
% plot(xIntersectionsRef, yIntersectionsRef, 'y+');
% hold off

end



% Find the best correspondence between the points in the input image and
% the points in the reference image.  If found, the output is the four
% outer corner points from the image, represented as a a 4x2 array, in the
% form [ [x1,y1]; [x2,y2], ... ].
function [corners, nMatchesBest, avgErrBest] = findCorrespondence( ...
   xIntersections, yIntersections, ...         % Input image points
   xIntersectionsRef, yIntersectionsRef, ...   % Reference image points
   I)
% Get the coordinates of the four outer corners of the reference image,
% in clockwise order starting from the top left.
pCornersRef = [ ...
   xIntersectionsRef(1,1), yIntersectionsRef(1,1);
   xIntersectionsRef(1,end), yIntersectionsRef(1,end);
```

```
    xIntersectionsRef(end,end), yIntersectionsRef(end,end);
    xIntersectionsRef(end,1), yIntersectionsRef(end,1) ];

M = 4;      % Number of lines to search in each direction
DMIN = 4;   % To match, a predicted point must be within this distance

nMatchesBest = 0;   % Number of matches of best candidate found so far
avgErrBest = 1e9;   % The average error of the best candidate

N1 = size(xIntersections,1);
N2 = size(xIntersections,2);
for i1a=1:min(M,N1)
    for i1b=N1:-1:max(N1-M,i1a+1)
        for i2a=1:min(M,N2)
            for i2b=N2:-1:max(N2-M,i2a+1)

                % Get the four corners corresponding to the intersections
                % of lines (1a,2a), (1a,2b), (1b,2b, and (1b,2a).
                pCornersImg = zeros(4,2);
                pCornersImg(1,:) = [xIntersections(i1a,i2a) yIntersections(i1a,i2a)];
                pCornersImg(2,:) = [xIntersections(i1a,i2b) yIntersections(i1a,i2b)];
                pCornersImg(3,:) = [xIntersections(i1b,i2b) yIntersections(i1b,i2b)];
                pCornersImg(4,:) = [xIntersections(i1b,i2a) yIntersections(i1b,i2a)];

                % Make sure that points are in clockwise order.
                % If not, exchange points 2 and 4.

                v12 = pCornersImg(2,:) - pCornersImg(1,:);
                v13 = pCornersImg(3,:) - pCornersImg(1,:);
                if v12(1)*v13(2) - v12(2)*v13(1) < 0
                    temp = pCornersImg(2,:);
                    pCornersImg(2,:) = pCornersImg(4,:);
                    pCornersImg(4,:) = temp;
                end


                % Fit a homography using those four points.
                T = fitgeotrans(pCornersRef, pCornersImg, 'projective');

                % Transform all reference points to the image.
                pIntersectionsRefWarp = transformPointsForward(T, ...
                    [xIntersectionsRef(:) yIntersectionsRef(:)]);
```

```
        % For each predicted reference point, find the closest
        % detected image point.
        dPts = 1e6 * ones(size(pIntersectionsRefWarp,1),1);
        for i=1:size(pIntersectionsRefWarp,1)
           x = pIntersectionsRefWarp(i,1);
           y = pIntersectionsRefWarp(i,2);
           d = ((x-xIntersections(:)).^2 + (y-yIntersections(:)).^2).^0.5;
           dmin = min(d);
           dPts(i) = dmin;
        end

        % If the distance is less than DMIN, count it as a match.
        nMatches = sum(dPts < DMIN);

        % Calculate the avg error of the matched points.
        avgErr = mean(dPts(dPts < DMIN));

        % Keep the best combination found so far, in terms of
        % the number of matches and the minimum error.
        if nMatches < nMatchesBest
           continue;
        end
        if (nMatches == nMatchesBest) && (avgErr > avgErrBest)
           continue;
        end


        % Got a better combination; save it.
        avgErrBest = avgErr;
        nMatchesBest = nMatches;
        corners = pCornersImg;
        % Display the predicted and measured points.
%           figure(14), imshow(I,[]);
%           title('Predicted and measured points');
%           hold on
%           plot(xIntersections(:), yIntersections(:), 'g.');
%           plot(pIntersectionsRefWarp(:,1), pIntersectionsRefWarp(:,2), 'yo');
%           hold off
%           rectangle('Position', [pCornersImg(1,1)-10 pCornersImg(1,2)-10 20 20], ...
%              'Curvature', [1 1], 'EdgeColor', 'r', 'LineWidth', 2);
%           rectangle('Position', [pCornersImg(2,1)-10 pCornersImg(2,2)-10 20 20], ...
%              'Curvature', [1 1], 'EdgeColor', 'g', 'LineWidth', 2);
```

```
%               rectangle('Position', [pCornersImg(3,1)-10 pCornersImg(3,2)-10 20 20], ...
%                   'Curvature', [1 1], 'EdgeColor', 'b', 'LineWidth', 2);
%               rectangle('Position', [pCornersImg(4,1)-10 pCornersImg(4,2)-10 20 20], ...
%                   'Curvature', [1 1], 'EdgeColor', 'y', 'LineWidth', 2);
%               fprintf(' Found %d matches, average error = %f\n', ...
%                   nMatchesBest, avgErrBest);
            break;
         end
         break;
      end
      break;
   end
   break;
end

end
```

[MATLAB HIGHLIGHTING CODE]*highlight_keys_for_song()* and *highlight_keys()*

```
function [whitekey,blackkey] =
highlight_keys_for_song(I,iFrame,whichkeys,whitekeyinitial,blackkeyinitial)
%if whichkeys is 0, we won't highlight any keys.
if iFrame==1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[corners, nMatches, avgErr, a1, a2] = findCheckerBoard(I);
% Returns:
%   corners: the locations of the four outer corners as a 4x2 array, in
%       the form [ [x1,y1]; [x2,y2]; ... ].
%   nMatches:  number of matching points found (ideally is 81)
%   avgErr:  the average reprojection error of the matching points
%       Return empty if not found.
%   a1:  clockwise angle from horizontal x+.
%   a2:  counterclockwise angle from vertical y-.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% There is a slight deviation of angle when shooting the movie, so the
% whole keyboard is like a "parallelogram".
diffxupper=(corners(2,1)-corners(1,1))/26; % width of sigle whitekey(upper) = X2(x of
point2)-X1(x of point1)
diffxlower=(corners(3,1)-corners(4,1))/26;% width of sigle whitekey(lower) = X3(x of point3)-X4(x
of point4)
diffyleft=corners(4,2)-corners(1,2);
diffyright=corners(3,2)-corners(2,2);
diffylefttoright=(diffyleft-diffyright)/26; % difference of y per keys from left to right especially
keyboard is not horizontal
% The whole keyboard is not absolutely "horizontal" in the camera, so we
% need "diffyupper" and "diffylower" as an adjustion.
diffyupper=sind(a1)*(corners(2,1)-corners(1,1))/26/26;
diffylower=sind(a1)*(corners(3,1)-corners(4,1))/26/26;

whitekey=cell(26,4); % create whitekey cell array
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:27 % identify and ciecle out every corner in the keyboard
   if i<18
       rectangle('Position', [corners(1,1)+0+(diffxupper+0.5)*(i-1)-10
corners(1,2)-10+diffyupper*(i-1) 10 10], ...
                 'Curvature', [1 1], 'EdgeColor', 'r', 'LineWidth', 1);
       rectangle('Position', [corners(4,1)+0+(diffxlower+0.5)*(i-1)-10
corners(4,2)-10+diffyupper*(i-1) 10 10], ...
                 'Curvature', [1 1], 'EdgeColor', 'r', 'LineWidth', 1);

   end
   if i>17
       rectangle('Position', [corners(1,1)+18+(diffxupper-0.5)*(i-1)-10
corners(1,2)-10+diffylower*(i-1) 10 10], ...
                 'Curvature', [1 1], 'EdgeColor', 'r', 'LineWidth', 1);
       rectangle('Position', [corners(4,1)+18+(diffxlower-0.5)*(i-1)-10
corners(4,2)-10+diffylower*(i-1) 10 10], ...
                 'Curvature', [1 1], 'EdgeColor', 'r', 'LineWidth', 1);
   end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
j=0;
```

```
for i=1:26  % record the position of four corners(clockwise) of the white keys into "whitekey" cell
array.
   if i<18
      whitekey{i,1}= [corners(1,1)+0+(diffxupper+0.5)*(i-1),
corners(1,2)+diffyupper*(i-1)-10+diffylefttoright*(j-1)/2];
      whitekey{i,4}= [corners(4,1)+0+(diffxlower+0.5)*(i-1),
corners(4,2)+diffyupper*(i-1)-diffylefttoright*(j-1)/2];
      whitekey{i,2}= [corners(1,1)+0+(diffxupper+0.5)*(i),
corners(1,2)+diffyupper*(i)-10+diffylefttoright*j/2];
      whitekey{i,3}= [corners(4,1)+0+(diffxlower+0.5)*(i),
corners(4,2)+diffyupper*(i)-diffylefttoright*j/2];
   end
   if i>17
      whitekey{i,1}= [corners(1,1)+18+(diffxupper-0.5)*(i-1),
corners(1,2)+diffylower*(i-1)-10+diffylefttoright*(j-1)/2];
      whitekey{i,4}= [corners(4,1)+18+(diffxlower-0.5)*(i-1),
corners(4,2)+diffylower*(i-1)-diffylefttoright*(j-1)/2];
      whitekey{i,2}= [corners(1,1)+18+(diffxupper-0.5)*(i),
corners(1,2)+diffylower*(i)-10+diffylefttoright*j/2];
      whitekey{i,3}= [corners(4,1)+18+(diffxlower-0.5)*(i),
corners(4,2)+diffylower*(i)-diffylefttoright*j/2];
   end
end
blackkey=cell(18,4); % create blackkey cell array
NN=0; % initialize NN
for i=1:18  % record the position of four corners(clockwise) of the black keys into "blackkey" cell
array.
   if i==1 || i==4 || i==6 || i==9 || i==11 || i==14 || i==16 %left blackkeys
      if i ==1 j=1; end
      if i ==4 j=5;NN=1; end
      if i ==6 j=8; end
      if i ==9 j=12;NN=1; end
      if i ==11 j=15; end
      if i ==14 j=19;NN=1; end
      if i ==16 j=22; end
      if NN==1
         blackkey{i,1}= [whitekey{j,2}(1)+diffxupper*0.07-diffxupper/2, whitekey{j,1}(2)];
         blackkey{i,4}= [whitekey{j,3}(1)+diffxlower*0.07-diffxlower/2,
whitekey{j,4}(2)-diffyleft*4/8];
         blackkey{i,2}= [whitekey{j,2}(1)+diffxupper*0.07, whitekey{j,2}(2)];
         blackkey{i,3}= [whitekey{j,3}(1)+diffxlower*0.07, whitekey{j,3}(2)-diffyleft*4/8];
      else
         blackkey{i,1}= [whitekey{j,2}(1)+diffxupper*0.1-diffxupper/2, whitekey{j,1}(2)];
```

```
            blackkey{i,4}= [whitekey{j,3}(1)+diffxlower*0.1-diffxlower/2, whitekey{j,4}(2)-diffyleft*4/8];
            blackkey{i,2}= [whitekey{j,2}(1)+diffxupper*0.1, whitekey{j,2}(2)];
            blackkey{i,3}= [whitekey{j,3}(1)+diffxlower*0.1, whitekey{j,3}(2)-diffyleft*4/8];
        end
    end
    if i==2 || i==7 || i==12 || i==17  % middle blackkeys
        if i ==2 j=2; end
        if i ==7 j=9; end
        if i ==12 j=16; end
        if i ==17 j=23; end
            blackkey{i,1}= [whitekey{j,2}(1)+diffxupper*0.25-diffxupper/2, whitekey{j,1}(2)];
            blackkey{i,4}= [whitekey{j,3}(1)+diffxlower*0.25-diffxlower/2,
whitekey{j,4}(2)-diffyleft*4/8];
            blackkey{i,2}= [whitekey{j,2}(1)+diffxupper*0.25, whitekey{j,2}(2)];
            blackkey{i,3}= [whitekey{j,3}(1)+diffxlower*0.25, whitekey{j,3}(2)-diffyleft*4/8];
    end
    if i==3 || i==5 || i==8 || i==10 || i==13 || i==15 || i==18 % right blackkeys
        if i ==3 j=3; end
        if i ==5 j=6;NN=1; end
        if i ==8 j=10; end
        if i ==10 j=13;NN=1; end
        if i ==13 j=17; end
        if i ==15 j=20;NN=1; end
        if i ==18 j=24; end
        if NN==1
            blackkey{i,1}= [whitekey{j,2}(1)+diffxupper*(0.5-0.07)-diffxupper/2, whitekey{j,1}(2)];
            blackkey{i,4}= [whitekey{j,3}(1)+diffxlower*(0.5-0.07)-diffxlower/2,
whitekey{j,4}(2)-diffyleft*4/8];
            blackkey{i,2}= [whitekey{j,2}(1)+diffxupper*(0.5-0.07), whitekey{j,2}(2)];
            blackkey{i,3}= [whitekey{j,3}(1)+diffxlower*(0.5-0.07), whitekey{j,3}(2)-diffyleft*4/8];
        else
            blackkey{i,1}= [whitekey{j,2}(1)+diffxupper*(0.5-0.1)-diffxupper/2, whitekey{j,1}(2)];
            blackkey{i,4}= [whitekey{j,3}(1)+diffxlower*(0.5-0.1)-diffxlower/2,
whitekey{j,4}(2)-diffyleft*4/8];
            blackkey{i,2}= [whitekey{j,2}(1)+diffxupper*(0.5-0.1), whitekey{j,2}(2)];
            blackkey{i,3}= [whitekey{j,3}(1)+diffxlower*(0.5-0.1), whitekey{j,3}(2)-diffyleft*4/8];
        end
    end
    NN=0; % reset NN
  if i>9 % slight adjustment
            blackkey{i,1}= [blackkey{i,1}(1)-2, blackkey{i,1}(2)];
            blackkey{i,4}= [blackkey{i,4}(1)-2, blackkey{i,4}(2)];
            blackkey{i,2}= [blackkey{i,2}(1)-2, blackkey{i,2}(2)];
```

```
        blackkey{i,3}= [blackkey{i,3}(1)-2, blackkey{i,3}(2)];
        if i==10 || i==15
           blackkey{i,1}= [blackkey{i,1}(1)-4, blackkey{i,1}(2)];
           blackkey{i,4}= [blackkey{i,4}(1)-4, blackkey{i,4}(2)];
           blackkey{i,2}= [blackkey{i,2}(1)-4, blackkey{i,2}(2)];
           blackkey{i,3}= [blackkey{i,3}(1)-4, blackkey{i,3}(2)];
        end
  end
end
end
%for test
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% for i= 1:26 % highlight each whitekey in left-to-right order by using line function and data from
"whitekey" cell array.
%
%     figure(1), imshow(I), title(sprintf('Frame %d', iFrame));
%     line([whitekey{i,1}(1),whitekey{i,2}(1)],[whitekey{i,1}(2),
whitekey{i,2}(2)],'Color','g','LineWidth',2)
%     line([whitekey{i,2}(1),whitekey{i,3}(1)],[whitekey{i,2}(2),
whitekey{i,3}(2)],'Color','g','LineWidth',2)
%     line([whitekey{i,3}(1),whitekey{i,4}(1)],[whitekey{i,3}(2),
whitekey{i,4}(2)],'Color','g','LineWidth',2)
%     line([whitekey{i,4}(1),whitekey{i,1}(1)],[whitekey{i,4}(2),
whitekey{i,1}(2)],'Color','g','LineWidth',2)
%     pause(0.2);
% end
%
% for i= 1:18 % highlight each blackkey in left-to-right order by using line function and data from
"blackkey" cell array.
%
%     figure(1), imshow(I), title(sprintf('Frame %d', iFrame));
%     line([blackkey{i,1}(1),blackkey{i,2}(1)],[blackkey{i,1}(2),
blackkey{i,2}(2)],'Color','g','LineWidth',2)
%     line([blackkey{i,2}(1),blackkey{i,3}(1)],[blackkey{i,2}(2),
blackkey{i,3}(2)],'Color','g','LineWidth',2)
%     line([blackkey{i,3}(1),blackkey{i,4}(1)],[blackkey{i,3}(2),
blackkey{i,4}(2)],'Color','g','LineWidth',2)
%     line([blackkey{i,4}(1),blackkey{i,1}(1)],[blackkey{i,4}(2),
blackkey{i,1}(2)],'Color','g','LineWidth',2)
%     pause(0.2);
% end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
% % JUST FOR DEMO
%  % highlight each whitekey in left-to-right order by using line function and data from
"whitekey" cell array.
%     i=keyinorder/20+1;
%     if i<27
%         line([whitekey{i,1}(1),whitekey{i,2}(1)],[whitekey{i,1}(2),
whitekey{i,2}(2)],'Color','g','LineWidth',2)
%         line([whitekey{i,2}(1),whitekey{i,3}(1)],[whitekey{i,2}(2),
whitekey{i,3}(2)],'Color','g','LineWidth',2)
%         line([whitekey{i,3}(1),whitekey{i,4}(1)],[whitekey{i,3}(2),
whitekey{i,4}(2)],'Color','g','LineWidth',2)
%         line([whitekey{i,4}(1),whitekey{i,1}(1)],[whitekey{i,4}(2),
whitekey{i,1}(2)],'Color','g','LineWidth',2)
%         pause(0.2);
%     end
%
%
%  % highlight each blackkey in left-to-right order by using line function and data from
"blackkey" cell array.
%     i=keyinorder/20+1;
%     if i<19
%         line([blackkey{i,1}(1),blackkey{i,2}(1)],[blackkey{i,1}(2),
blackkey{i,2}(2)],'Color','b','LineWidth',3)
%         line([blackkey{i,2}(1),blackkey{i,3}(1)],[blackkey{i,2}(2),
blackkey{i,3}(2)],'Color','b','LineWidth',3)
%         line([blackkey{i,3}(1),blackkey{i,4}(1)],[blackkey{i,3}(2),
blackkey{i,4}(2)],'Color','b','LineWidth',3)
%         line([blackkey{i,4}(1),blackkey{i,1}(1)],[blackkey{i,4}(2),
blackkey{i,1}(2)],'Color','b','LineWidth',3)
%         pause(0.2);
%     end
%
% if iFrame-keyinorder>19 keyinorder=keyinorder+20; end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
if iFrame>1
   if length(whichkeys)>0
      if whichkeys(1)>0
         for i=whichkeys(1,:);
```

```
            line([whitekeyinitial{i,1}(1),whitekeyinitial{i,2}(1)],[whitekeyinitial{i,1}(2),
whitekeyinitial{i,2}(2)],'Color','r','LineWidth',2)
            line([whitekeyinitial{i,2}(1),whitekeyinitial{i,3}(1)],[whitekeyinitial{i,2}(2),
whitekeyinitial{i,3}(2)],'Color','r','LineWidth',2)
            line([whitekeyinitial{i,3}(1),whitekeyinitial{i,4}(1)],[whitekeyinitial{i,3}(2),
whitekeyinitial{i,4}(2)],'Color','r','LineWidth',2)
            line([whitekeyinitial{i,4}(1),whitekeyinitial{i,1}(1)],[whitekeyinitial{i,4}(2),
whitekeyinitial{i,1}(2)],'Color','r','LineWidth',2)
%               pause(0.2);
         end
      end
   end
end
% newFrameOut = getframe;
% writeVideo(movieObjOutput,newFrameOut);
% pause(0.1);
end
```

```
function [whitekey,blackkey] = highlight_keys(I,iFrame,whichkeys,whitekeyinitial,blackkeyinitial)
%if whichkeys is 0, we won't highlight any keys.
if iFrame==1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[corners, nMatches, avgErr, a1, a2] = findCheckerBoard(I);
% Returns:
%   corners: the locations of the four outer corners as a 4x2 array, in
%      the form [ [x1,y1]; [x2,y2]; ... ].
%   nMatches:  number of matching points found (ideally is 81)
%   avgErr:  the average reprojection error of the matching points
%      Return empty if not found.
%   a1:  clockwise angle from horizontal x+.
%   a2:  counterclockwise angle from vertical y-.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% There is a slight deviation of angle when shooting the movie, so the
% whole keyboard is like a "parallelogram".
diffxupper=(corners(2,1)-corners(1,1))/26; % width of sigle whitekey(upper) = X2(x of
point2)-X1(x of point1)
diffxlower=(corners(3,1)-corners(4,1))/26;% width of sigle whitekey(lower) = X3(x of point3)-X4(x
of point4)
diffyleft=corners(4,2)-corners(1,2);
diffyright=corners(3,2)-corners(2,2);
diffylefttoright=(diffyleft-diffyright)/26; % difference of y per keys from left to right especially
keyboard is not horizontal
% The whole keyboard is not absolutely "horizontal" in the camera, so we
% need "diffyupper" and "diffylower" as an adjustion.
diffyupper=sind(a1)*(corners(2,1)-corners(1,1))/26/26;
diffylower=sind(a1)*(corners(3,1)-corners(4,1))/26/26;

whitekey=cell(26,4); % create whitekey cell array
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:27 % identify and ciecle out every corner in the keyboard
   if i<18
       rectangle('Position', [corners(1,1)+0+(diffxupper+0.5)*(i-1)-10
corners(1,2)-10+diffyupper*(i-1) 10 10], ...
                'Curvature', [1 1], 'EdgeColor', 'r', 'LineWidth', 1);
       rectangle('Position', [corners(4,1)+0+(diffxlower+0.5)*(i-1)-10
corners(4,2)-10+diffyupper*(i-1) 10 10], ...
                'Curvature', [1 1], 'EdgeColor', 'r', 'LineWidth', 1);

   end
   if i>17
       rectangle('Position', [corners(1,1)+18+(diffxupper-0.5)*(i-1)-10
corners(1,2)-10+diffylower*(i-1) 10 10], ...
                'Curvature', [1 1], 'EdgeColor', 'r', 'LineWidth', 1);
       rectangle('Position', [corners(4,1)+18+(diffxlower-0.5)*(i-1)-10
corners(4,2)-10+diffylower*(i-1) 10 10], ...
                'Curvature', [1 1], 'EdgeColor', 'r', 'LineWidth', 1);
   end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
j=0;
for i=1:26  % record the position of four corners(clockwise) of the white keys into "whitekey" cell
array.
   if i<18
```

```
        whitekey{i,1}= [corners(1,1)+0+(diffxupper+0.5)*(i-1),
corners(1,2)+diffyupper*(i-1)-10+diffylefttoright*(j-1)/2];
        whitekey{i,4}= [corners(4,1)+0+(diffxlower+0.5)*(i-1),
corners(4,2)+diffyupper*(i-1)-diffylefttoright*(j-1)/2];
        whitekey{i,2}= [corners(1,1)+0+(diffxupper+0.5)*(i),
corners(1,2)+diffyupper*(i)-10+diffylefttoright*j/2];
        whitekey{i,3}= [corners(4,1)+0+(diffxlower+0.5)*(i),
corners(4,2)+diffyupper*(i)-diffylefttoright*j/2];
    end
    if i>17
        whitekey{i,1}= [corners(1,1)+18+(diffxupper-0.5)*(i-1),
corners(1,2)+diffylower*(i-1)-10+diffylefttoright*(j-1)/2];
        whitekey{i,4}= [corners(4,1)+18+(diffxlower-0.5)*(i-1),
corners(4,2)+diffylower*(i-1)-diffylefttoright*(j-1)/2];
        whitekey{i,2}= [corners(1,1)+18+(diffxupper-0.5)*(i),
corners(1,2)+diffylower*(i)-10+diffylefttoright*j/2];
        whitekey{i,3}= [corners(4,1)+18+(diffxlower-0.5)*(i),
corners(4,2)+diffylower*(i)-diffylefttoright*j/2];
    end
end
blackkey=cell(18,4); % create blackkey cell array
NN=0; % initialize NN
for i=1:18  % record the position of four corners(clockwise) of the black keys into "blackkey" cell
array.
    if i==1 || i==4 || i==6 || i==9 || i==11 || i==14 || i==16 %left blackkeys
        if i ==1 j=1; end
        if i ==4 j=5;NN=1; end
        if i ==6 j=8; end
        if i ==9 j=12;NN=1; end
        if i ==11 j=15; end
        if i ==14 j=19;NN=1; end
        if i ==16 j=22; end
        if NN==1
            blackkey{i,1}= [whitekey{j,2}(1)+diffxupper*0.07-diffxupper/2, whitekey{j,1}(2)];
            blackkey{i,4}= [whitekey{j,3}(1)+diffxlower*0.07-diffxlower/2,
whitekey{j,4}(2)-diffyleft*4/8];
            blackkey{i,2}= [whitekey{j,2}(1)+diffxupper*0.07, whitekey{j,2}(2)];
            blackkey{i,3}= [whitekey{j,3}(1)+diffxlower*0.07, whitekey{j,3}(2)-diffyleft*4/8];
        else
            blackkey{i,1}= [whitekey{j,2}(1)+diffxupper*0.1-diffxupper/2, whitekey{j,1}(2)];
            blackkey{i,4}= [whitekey{j,3}(1)+diffxlower*0.1-diffxlower/2, whitekey{j,4}(2)-diffyleft*4/8];
            blackkey{i,2}= [whitekey{j,2}(1)+diffxupper*0.1, whitekey{j,2}(2)];
            blackkey{i,3}= [whitekey{j,3}(1)+diffxlower*0.1, whitekey{j,3}(2)-diffyleft*4/8];
```

```
        end
    end
    if i==2 || i==7 || i==12 || i==17  % middle blackkeys
        if i ==2 j=2; end
        if i ==7 j=9; end
        if i ==12 j=16; end
        if i ==17 j=23; end
            blackkey{i,1}= [whitekey{j,2}(1)+diffxupper*0.25-diffxupper/2, whitekey{j,1}(2)];
            blackkey{i,4}= [whitekey{j,3}(1)+diffxlower*0.25-diffxlower/2,
whitekey{j,4}(2)-diffyleft*4/8];
            blackkey{i,2}= [whitekey{j,2}(1)+diffxupper*0.25, whitekey{j,2}(2)];
            blackkey{i,3}= [whitekey{j,3}(1)+diffxlower*0.25, whitekey{j,3}(2)-diffyleft*4/8];
    end
    if i==3 || i==5 || i==8 || i==10 || i==13 || i==15 || i==18 % right blackkeys
        if i ==3 j=3; end
        if i ==5 j=6;NN=1; end
        if i ==8 j=10; end
        if i ==10 j=13;NN=1; end
        if i ==13 j=17; end
        if i ==15 j=20;NN=1; end
        if i ==18 j=24; end
        if NN==1
            blackkey{i,1}= [whitekey{j,2}(1)+diffxupper*(0.5-0.07)-diffxupper/2, whitekey{j,1}(2)];
            blackkey{i,4}= [whitekey{j,3}(1)+diffxlower*(0.5-0.07)-diffxlower/2,
whitekey{j,4}(2)-diffyleft*4/8];
            blackkey{i,2}= [whitekey{j,2}(1)+diffxupper*(0.5-0.07), whitekey{j,2}(2)];
            blackkey{i,3}= [whitekey{j,3}(1)+diffxlower*(0.5-0.07), whitekey{j,3}(2)-diffyleft*4/8];
        else
            blackkey{i,1}= [whitekey{j,2}(1)+diffxupper*(0.5-0.1)-diffxupper/2, whitekey{j,1}(2)];
            blackkey{i,4}= [whitekey{j,3}(1)+diffxlower*(0.5-0.1)-diffxlower/2,
whitekey{j,4}(2)-diffyleft*4/8];
            blackkey{i,2}= [whitekey{j,2}(1)+diffxupper*(0.5-0.1), whitekey{j,2}(2)];
            blackkey{i,3}= [whitekey{j,3}(1)+diffxlower*(0.5-0.1), whitekey{j,3}(2)-diffyleft*4/8];
        end
    end
    NN=0; % reset NN
  if i>9 % slight adjustment
            blackkey{i,1}= [blackkey{i,1}(1)-2, blackkey{i,1}(2)];
            blackkey{i,4}= [blackkey{i,4}(1)-2, blackkey{i,4}(2)];
            blackkey{i,2}= [blackkey{i,2}(1)-2, blackkey{i,2}(2)];
            blackkey{i,3}= [blackkey{i,3}(1)-2, blackkey{i,3}(2)];
            if i==10 || i==15
                blackkey{i,1}= [blackkey{i,1}(1)-4, blackkey{i,1}(2)];
```

```
            blackkey{i,4}= [blackkey{i,4}(1)-4, blackkey{i,4}(2)];
            blackkey{i,2}= [blackkey{i,2}(1)-4, blackkey{i,2}(2)];
            blackkey{i,3}= [blackkey{i,3}(1)-4, blackkey{i,3}(2)];
         end
  end
end
end
%for test
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
% for i= 1:26 % highlight each whitekey in left-to-right order by using line function and data from
"whitekey" cell array.
%
%     figure(1), imshow(I), title(sprintf('Frame %d', iFrame));
%     line([whitekey{i,1}(1),whitekey{i,2}(1)],[whitekey{i,1}(2),
whitekey{i,2}(2)],'Color','g','LineWidth',2)
%     line([whitekey{i,2}(1),whitekey{i,3}(1)],[whitekey{i,2}(2),
whitekey{i,3}(2)],'Color','g','LineWidth',2)
%     line([whitekey{i,3}(1),whitekey{i,4}(1)],[whitekey{i,3}(2),
whitekey{i,4}(2)],'Color','g','LineWidth',2)
%     line([whitekey{i,4}(1),whitekey{i,1}(1)],[whitekey{i,4}(2),
whitekey{i,1}(2)],'Color','g','LineWidth',2)
%     pause(0.2);
% end
%
% for i= 1:18 % highlight each blackkey in left-to-right order by using line function and data from
"blackkey" cell array.
%
%     figure(1), imshow(I), title(sprintf('Frame %d', iFrame));
%     line([blackkey{i,1}(1),blackkey{i,2}(1)],[blackkey{i,1}(2),
blackkey{i,2}(2)],'Color','g','LineWidth',2)
%     line([blackkey{i,2}(1),blackkey{i,3}(1)],[blackkey{i,2}(2),
blackkey{i,3}(2)],'Color','g','LineWidth',2)
%     line([blackkey{i,3}(1),blackkey{i,4}(1)],[blackkey{i,3}(2),
blackkey{i,4}(2)],'Color','g','LineWidth',2)
%     line([blackkey{i,4}(1),blackkey{i,1}(1)],[blackkey{i,4}(2),
blackkey{i,1}(2)],'Color','g','LineWidth',2)
%     pause(0.2);
% end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
```

```
% % JUST FOR DEMO
%  % highlight each whitekey in left-to-right order by using line function and data from
"whitekey" cell array.
%     i=keyinorder/20+1;
%     if i<27
%         line([whitekey{i,1}(1),whitekey{i,2}(1)],[whitekey{i,1}(2),
whitekey{i,2}(2)],'Color','g','LineWidth',2)
%         line([whitekey{i,2}(1),whitekey{i,3}(1)],[whitekey{i,2}(2),
whitekey{i,3}(2)],'Color','g','LineWidth',2)
%         line([whitekey{i,3}(1),whitekey{i,4}(1)],[whitekey{i,3}(2),
whitekey{i,4}(2)],'Color','g','LineWidth',2)
%         line([whitekey{i,4}(1),whitekey{i,1}(1)],[whitekey{i,4}(2),
whitekey{i,1}(2)],'Color','g','LineWidth',2)
%         pause(0.2);
%     end
%
%
%  % highlight each blackkey in left-to-right order by using line function and data from
"blackkey" cell array.
%     i=keyinorder/20+1;
%     if i<19
%         line([blackkey{i,1}(1),blackkey{i,2}(1)],[blackkey{i,1}(2),
blackkey{i,2}(2)],'Color','b','LineWidth',3)
%         line([blackkey{i,2}(1),blackkey{i,3}(1)],[blackkey{i,2}(2),
blackkey{i,3}(2)],'Color','b','LineWidth',3)
%         line([blackkey{i,3}(1),blackkey{i,4}(1)],[blackkey{i,3}(2),
blackkey{i,4}(2)],'Color','b','LineWidth',3)
%         line([blackkey{i,4}(1),blackkey{i,1}(1)],[blackkey{i,4}(2),
blackkey{i,1}(2)],'Color','b','LineWidth',3)
%         pause(0.2);
%     end
%
% if iFrame-keyinorder>19 keyinorder=keyinorder+20; end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
if iFrame>1
   if length(whichkeys)>0
      if whichkeys(1)>0
         for i=whichkeys(1,:);
            line([whitekeyinitial{i,1}(1),whitekeyinitial{i,2}(1)],[whitekeyinitial{i,1}(2),
whitekeyinitial{i,2}(2)],'Color','g','LineWidth',2)
```

```
            line([whitekeyinitial{i,2}(1),whitekeyinitial{i,3}(1)],[whitekeyinitial{i,2}(2),
whitekeyinitial{i,3}(2)],'Color','g','LineWidth',2)
            line([whitekeyinitial{i,3}(1),whitekeyinitial{i,4}(1)],[whitekeyinitial{i,3}(2),
whitekeyinitial{i,4}(2)],'Color','g','LineWidth',2)
            line([whitekeyinitial{i,4}(1),whitekeyinitial{i,1}(1)],[whitekeyinitial{i,4}(2),
whitekeyinitial{i,1}(2)],'Color','g','LineWidth',2)
%               pause(0.2);
         end
      end
   end
end
% newFrameOut = getframe;
% writeVideo(movieObjOutput,newFrameOut);
% pause(0.1);
end
```

[MATLAB FIND SKIN CODE] *findskin()*

```
function [final_image] = findskin(I) %%skin region in rgb image%%%%%%%%%
if(size(I, 3) > 1)
   final_image = zeros(size(I,1), size(I,2));
   for i = 1:size(I,1)
      for j = 162:1102
         R = I(i,j,1);
         G = I(i,j,2);
         B = I(i,j,3);
%           if(R > 95 && G > 40 && B > 20)
          if(R > 100 && G > 45 && B > 26)
            v = [R,G,B];
%              if((max(v) - min(v)) > 15)
            if((max(v) - min(v)) > 15)
```

```
                if(abs(R-G) > 25 && R > G && R > B)
                    %it is a skin
                    final_image(i,j) = 1;
                end
            end
        end
    end
end %%% added
%     figure, imshow(final_image);
%     disp('before');
BW=final_image;
%else
%    BW=im2bw(I);
 % figure, imshow(BW);
%disp('convert');
%end
L = bwlabel(BW,4);

BB = regionprops(L, 'BoundingBox');

BB1 =struct2cell(BB);
BB2 = cell2mat(BB1);
[s1 s2]=size(BB2);

% mx=0; % find max value in BB2
%
% for k=3:4:s2-1
%     p=BB2(1,k)*BB2(1,k+1);
%     if p>mx && (BB2(1,k)/BB2(1,k+1))<1.8
%         mx=p;
%         j=k;
%     end
% end
 end
```

[MATLAB FINGER DETECTOR CODE] *HandGesture()*

```
function [fingertipx, fingertipy, hand5] = HandGesture(I)
```

```
% fingertipx= x coord of peaks(finger tips) on the boundary of hand
% fingertipy= y coord of peaks(finger tips) on the boundary of hand
% hand5= (useless) skin color filtered, eroded,dilated and median filtered image of hand
hand = findskin(I); % hand with noise
%  figure(2);imshow(hand);
fingertipx=0;
fingertipy=0;
% hand2= imerode(hand,strel('disk',18));
% figure(2);imshow(hand2);
%
% hand3= imdilate(hand2,strel('disk',10));
% figure(3);imshow(hand3);

% hand = bwareaopen(hand, 10000);
% hand = imfill(hand,'holes');
% figure(4);imshow(hand3);title('Small Areas removed & Holes Filled');
hand2 = imerode(hand,strel('disk',10));                    %erode image
%   imshow(hand2)
hand3 = imdilate(hand2,strel('disk',10));                  %dilate iamge
%    figure(2);imshow(hand3)
% hand4 = medfilt2(hand3, [5 5]);                           %median filtering
% figure(2);imshow(hand4);title('Eroded,Dilated & Median Filtered');
hand5 = bwareaopen(hand3, 50);                             %finds objects, noise or regions
with pixel area lower than 2000 and removes them
%  figure(3);imshow(hand5);title('Processed');              %displays image with reduced noise
hand5 = flipdim(hand5,1);                                  %flip image rows
%  figure(4);imshow(hand3);title('Flip Image');
%   imshow(hand5)

REG=regionprops(hand5,'all');                              %calculate the properties of
regions for objects found
CEN = cat(1, REG.Centroid);                                %calculate Centroid
[B, L, N, A] = bwboundaries(hand5,'noholes');              %returns a label matrix L
where objects and holes are labeled.
                                                           %returns N, the number of objects found, and A,
an adjacency matrix.
% RND = 0;                                                 % set variable RND to zero; to prevent
errors if no object detected
if length(B)>0
      fingertipx=cell(length(B),1);
      fingertipy=cell(length(B),1);
%calculate the properties of regions for objects found
   for k =1:length(B)                                      %for the given object k
```

```
        PER = REG(k).Perimeter;                         %Perimeter is set as perimeter
calculated by region properties
        ARE = REG(k).Area;                              %Area is set as area calculated by
region properties
        RND = (4*pi*ARE)/(PER^2);                       %Roundness value is calculated

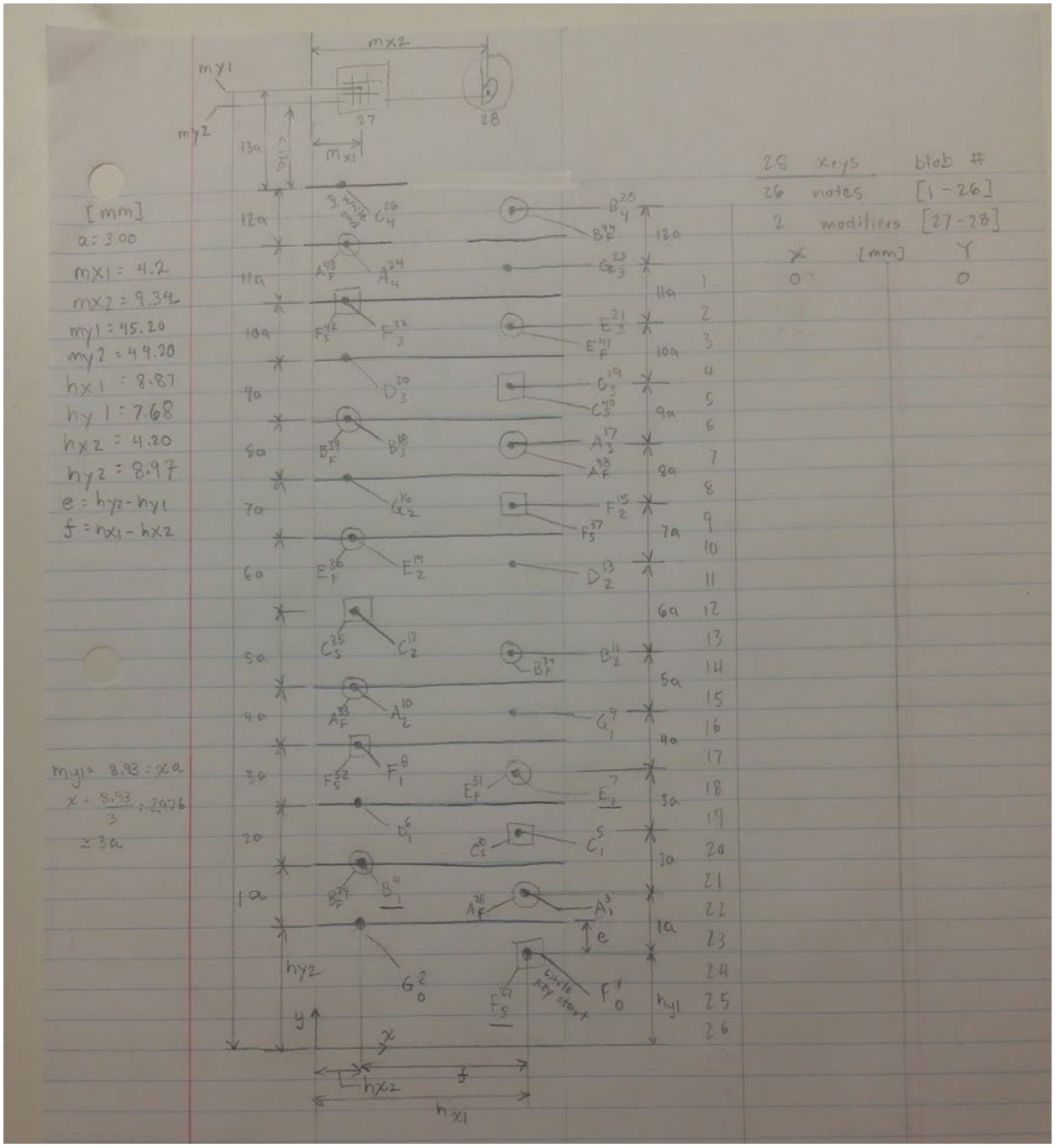
        BND = B{k};                                     %boundary set for object
        BNDx = BND(:,2);                                %Boundary x coord
        BNDy = BND(:,1);                                %Boundary y coord
%           if mean(BNDx)<173 || mean(BNDx)>1095 || mean(BNDy)<538
%               fingertipx{k,1}=[0];
%               fingertipy{k,1}=[0];
%               continue;
%           end
        pkoffset = CEN(:,2)+.5*(CEN(:,2));              %Calculate peak offset point from
centroid
        [pks,locs] = findpeaks(BNDy,'minpeakheight',1);     %find peaks in the boundary in y
axis with a minimum height greater than the peak offset
        [u,v,w]=size(I);
        BNDy=u-BNDy;
        pks=u-pks;
        CEN(:,2)=u-CEN(:,2);
        [u1,v1]=size(pks);
        if u1>1 % just pick the highest peak when there are more than one peaks in the same
boundary
            [pks,minidx]=min(pks);
            locs=locs(minidx,1);
        end
        if pks>430 || pks<272 || BNDx(locs)<170 || BNDx(locs)>1097
            continue;
        end
        fingertipx{k,1}=BNDx(locs);
        fingertipy{k,1}=pks;
        hold on
        plot(BNDx, BNDy, 'b', 'LineWidth', 2);              %plot Boundary
%         plot(CEN(:,1),CEN(:,2), '*');                   %plot centroid
        plot(BNDx(locs),pks,'rv','MarkerFaceColor','r','lineWidth',2);  %plot peaks
        hold off
%           pkNo = u1;                                  %finds the peak Nos
%           pkNo_STR = sprintf('%2.0f',pkNo);               %puts the peakNo in a string
    end
                                                    % roundness is useful, for an object of same
shape ratio, regardless of
```

```
                                        % size the roundess value remains the same.
For instance, a circle with
                                        % radius 5pixels will have the same roundness
as a circle with radius
                                        % 100pixels. It is a measure of how round an
object is.

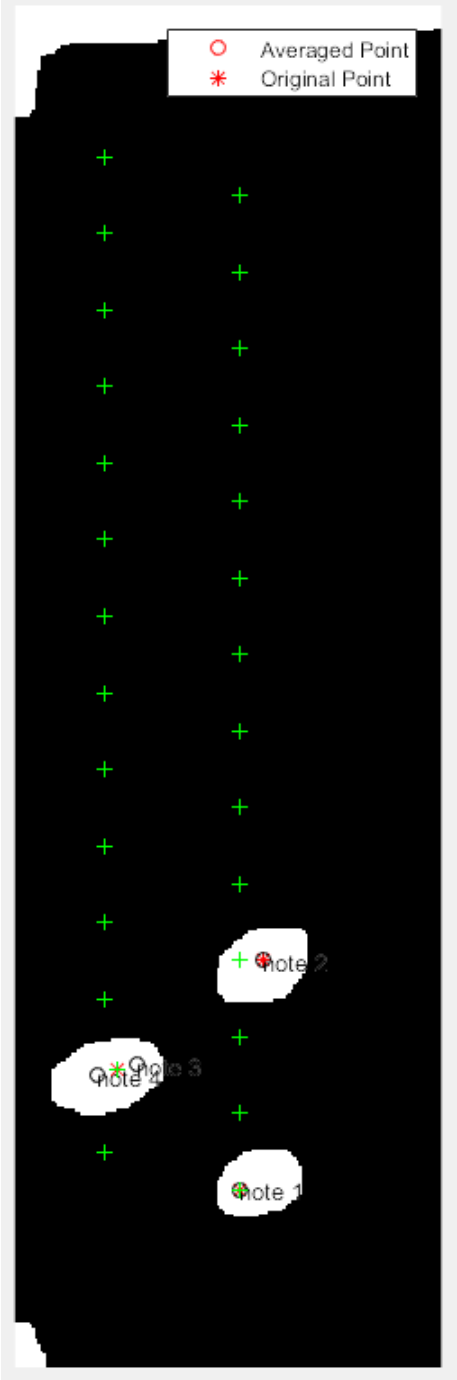
                                        %     CHAR_STR = 'not identified hand';
%sets char_str value to 'not identified'
%     if RND >0.19 && RND < 0.24 && pkNo ==3
%         CHAR_STR = 'W';
%     elseif RND >0.44 && RND < 0.47  && pkNo ==1
%         CHAR_STR = 'O';
%     elseif RND >0.37 && RND < 0.40 && pkNo ==2
%         CHAR_STR = 'R';
%     elseif RND >0.40 && RND < 0.43 && pkNo == 3
%         CHAR_STR = 'D';
%     else
%         CHAR_STR = 'hand';
%     end
%     text(20,20,CHAR_STR,'color','g','Fontsize',14);                    %place text in x=20,y=20 on
the figure with the value of Char_str in redcolour with font size 18
%     text(20,50,['roundness: ' sprintf('%f',RND)],'color','g','Fontsize',14);
%     text(20,80,['finger tips: ' pkNo_STR],'color','g','Fontsize',14);

end
end
```

[LCD MEASUREMENTS] Ground Truth Measurements of the LCD Screen



[LCD NOTES] The locations of the expected notes laid on top of the thresholded LCD screen



[NOTES TABLE] Raw data for figure 25

Model Player (Ideal)				MATLAB Music Reader (Actual)		
Time (sec)	Key	Note		Time (sec)	Key	Note
0	18	B		0	18	B
0.5	19	C		0.5	18	B
1	20	D		1	19	C
1.5	20	D		1.5	20	D
2	19	C		2	20	D
2.5	18	B		2.5	19	C
3	17	A		3		
3.5	16	G		3.5	16	G
4	16	G		4	16	G
4.5	17	A		4.5	17	A
5	18	B		5	18	B
5.5	18	B		5.5	18	B
6	17	A		6	17	A
6.5	18	B		6.5	17	A