18-551 Fall 2009

Final Report

EYE KNOW YOU

Iris Recognition in Real-Time

Group 5
Andy Lin (aalin@andrew.cmu.edu)
Stacy Lin (stacyl@andrew.cmu.edu)
Vincent Yen (chengchy@andrew.cmu.edu)

Table of Contents

1. Problem

In recent years, security and surveillance has become an increasing issue. Identifying and tracking someone is exceptionally hard in large crowds such as in train stations, subway stations, airports etc. At crucial times, the speed, ease, and accuracy are of utmost importance in real time multiple person tracking and recognition.

The development of biometrics identification systems is a very popular research topic in artificial intelligence. Among all the applications of biometrics identification, face recognition is most often used for automatic visual surveillance systems. However, face recognition is not as accurate as believed. A person's face can appear to be quite different when time, lighting, angle, or emotional state change. In addition, faces of related individuals, or even unrelated ones, can appear to be quite similar. Changes in pose will cause large changes in the appearance of a face, which further complicates the problem of robust face recognition.

On the other hand, the iris of an individual is very stable over time, with the exception of rare disease or trauma. It does not change over time, as faces do, and is well protected from damage and scarring, which is often a problem with fingerprint identification. The iris of each eye is unique for every person. No two irises are alike in their mathematical detail. Even between identical twins and triplets or between one's own left and right eyes, the iris is different. The random patterns of the iris are the equivalent of a complex "human barcode," created by a tangled meshwork of connective tissue and other visible features. [5] The research and use of iris recognition as a form of identification has increased in popularity in recent years.

2. Solution

Our goal is to design an autonomous system to track and identify multiple people through iris recognition in real time. What we are proposing is an iris recognition tracking system that will be able to track multiple people, regardless of their poses to a certain point, in a video. Our system will recognize and identify people and be able to tell if that person is someone we know (in the database) or if they are new (not in the database).

In order to do iris recognition, face detection is a necessary first step to assist and improve the overall performance of recognition. The first step in recognition would be to first locate the face in the video. Initially our plan was to implement Viola-Jones' method in their "Robust Real-Time Face Detection" paper, where they were able to process images extremely rapidly while achieving high detection rates. Their method allows for better results in indoor and outdoor scenarios because it is a feature-based face detection algorithm and independent of the color of the subject and lighting conditions in which many other algorithms are not.

Once the face has been located, we will then locate the eyes and ideally zoom in on the eyes with the camera to perform iris recognition. Iris recognition was chosen instead of previous methods of facial recognition because iris recognition is more accurate. The process of capturing an iris into a biometric template includes segmentation, normalization, unwrapping, and then recognition, which we implement based off of Libor Masek et al algorithm.

3. Novelty

Overall, our process itself is quite novel as past groups who did iris recognition did it directly from still images as it was implemented for verification and identification. We on the other hand implement and simulate a process of identifying from a video, and returning the results in real time. The idea of zooming in on the eyes after finding the face is a new idea. As far as we know, no prior groups in 18-551 or anyone in the field has done anything with it. The main goal of our project was to do iris recognition in real time and with most of the processing on the DSK.

A few groups in the past years have attempted iris recognition. They all followed methods designed around the work of John Daugman. Our segmentation algorithm on the other hand, follows that of Masek's which has never been implemented in this class before. While the actual iris recognition algorithm is a standard method, no past group has done it completely or successfully on the DSK board.

Group 6 of Spring 2003, "The Eyes have it", successfully implemented many of Daugman's published techniques, circular edge detection, Gabor filters for feature extraction and hamming distance for classification. Their database consisted of 48 images, that included 48 classes with only one image in each, that they got from Professor Kumar. All images had the subject's eyes held wide open, as there were no obstructions from eyelids or eyelashes. They experienced significant trouble with the reflections from the flash inside the pupil. To account for their problems with the flash reflections in the pupil, they made assumptions of the location of the center of the iris and the size of every eye. They were able to get code working on the EVM, however, their main shortcoming was their database. [6] We did not have major problems with our database as we accounted for the reflection from the camera flash in our algorithm.

Group 7 of Spring 2004, "Private Eyes: Biometric Identification by Iris Recognition", followed the same method but improved on the previous groups' problems. They implemented all equivalent steps of past groups without EVM trouble, with no assumptions made for the iris location or edge location, the flash reflections in the iris image were not problematic, the database was larger and more detailed (higher resolution), they had a more advanced classifier and a logical testing process. The database they used was supplied by a CMU-ECE doctoral student. The database consisted of 263 iris images, with 23 classes, each of which had between 6 and 13 images; 82% of the classes had between 11-13 images, inclusive. The images were acquired using a high-resolution digital camera. Participants were told they could shift around a bit and "be natural", it was not required that the eyelids be open as much as possible, and often the camera was re-focused between each picture. Within the 263 images there were several blurry pictures, and a few "blinks". They did not remove these images, but instead kept them for testing. [5] Though their project was successful, they only calculated the Discrete Fourier Transform and performed feature extraction on the DSK. We did almost all of our processing and recognition on the DSK except for the segmentation pre-processing of the iris images. In addition, our database differs from all past groups as it not only consists of frontal iris images of our own group members, but angled images as well as we test the degree at which the head can be turned in order for iris recognition to still work.

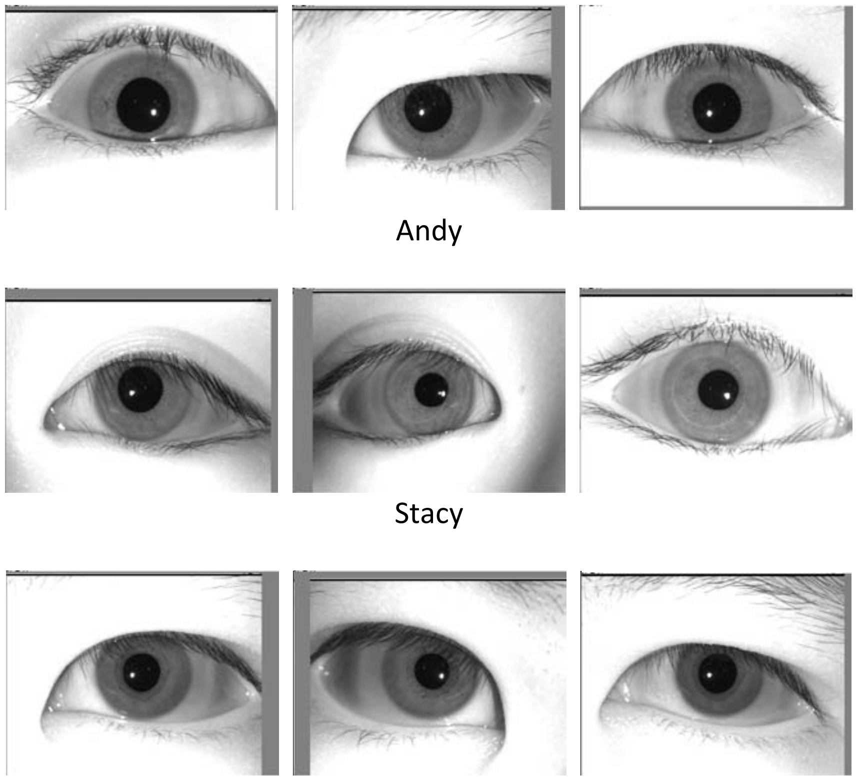
As for face and eye detection, many groups in 18-551 have previously done projects involving face detection, verification, identification and recognition via various methods. Many groups implemented the skin tone detection method. We decided to move forward using similar methods as past groups as we felt this was not the main focus of our project. Thus, we chose to adopt a similar skin tone detection algorithm to that of Group 1 of Spring 2008.

However, because we needed to find the eyes as well as the face, we implemented Rein Lien Hsu's Eye Map algorithm after face detection, which has never been implemented on the DSK. We felt that face detection was not our prominent focus of our project and testing, hence we decided to not further explore other algorithms.

4. Database

4.1 Iris

The training database used consisted of 10 iris images of each group member (3) at different angles, varying up to 10 degrees, for a total of 30 iris images, each of which are 640x480 pixels. These images were taken using a high resolution camera specifically designed to take iris images. This training database is what the testing iris images will be compared to in the final demo (Figure 1).



Vincent

Figure 1. Iris Training Database Example

The testing database (for the demo) consisted of another 12 iris images of each group member under different lighting conditions than that of the training database, and also at different angles, but varying up to 30 degrees. Also included were 4 unknown irises from the Western Virginia University Iris Database to make a total of 40 images to compare with the training database.

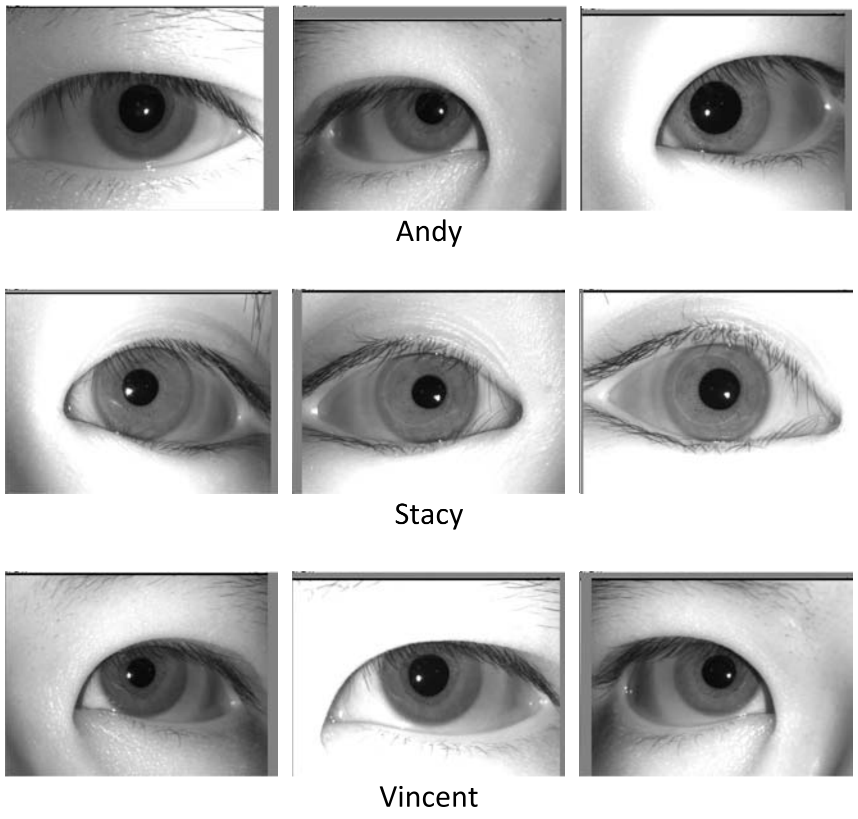


Figure 2. Iris Testing Database Example

In addition to our own training database, we also used one of each 320x280 pixel frontal iris image of 50 people from the Western Virginia University Iris Database (Figure 3), for inter- and intra- class threshold testing of our iris recognition algorithm discussed later.

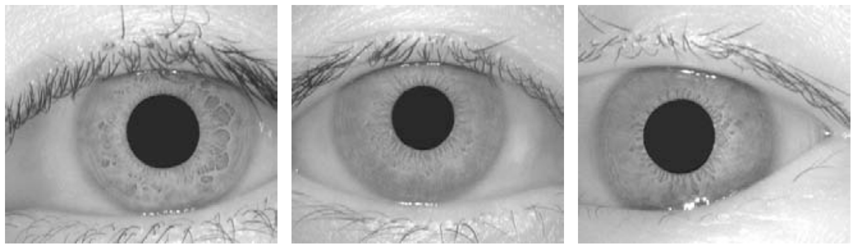


Figure 3. Western Virginia University Iris Database

4.2 Video

We created our own video collection by recording 7 short 160x120 pixel videos of one to two people moving around. This was produced using a Logitech QuickCam Connect which is capable of video capture of up to 640x480 pixels and 30 frames per second. The videos will consist of people moving around, coming in and out, both in the foreground and background. Also, it will primarily consist of only the face of the person and not the entire person. Because of the limited size in video, we can only fit one to at most two people.

5. Algorithm

5.1 Brief Systems Overview

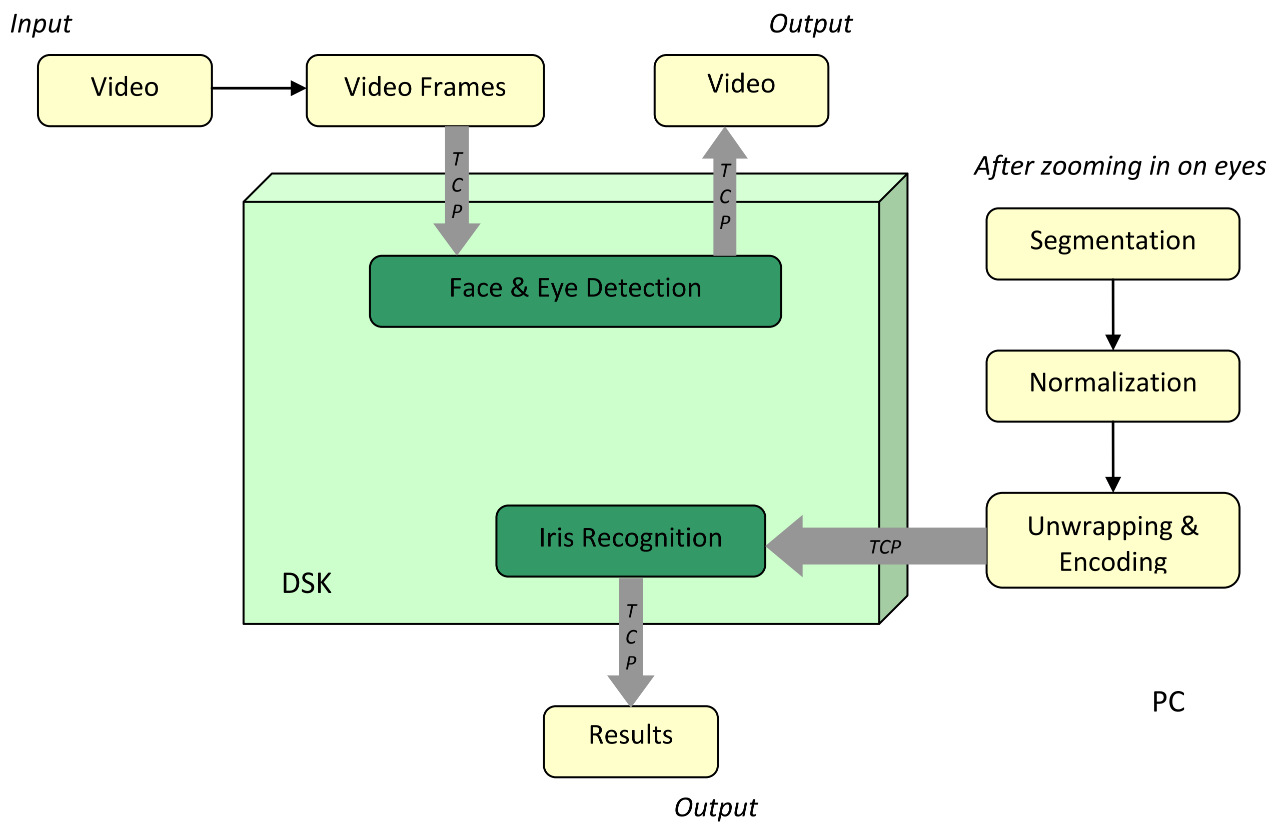


Figure 4. Systems Layout

Above is an overview of our system. The main parts, the face and eye detection and iris recognition (in green), are implemented on the DSK. The parts in yellow, the video and iris segmentation pre-processing, are all done on the PC.

5.2 Initial Detection Algorithm

In the early stages of our research into this problem, we probed into many different areas to find the optimal solution. Initially, we chose to use the Viola-Jones method to implement our real-time face-detector since the Viola-Jones method allows for rapid computation of feature values using integral image and Haar-like features. The face detection procedure classifies images based on the value of simple Haar-like features. Although the process that Viola-Jones established in their paper uses four kinds of Haar-like features, we originally had planned on using only the first three kinds of Haar-like features to reduce the total number of features to train.

By analyzing corresponding weights, we can select around 200 most important features to be our final classifier that can classify any 24x24 image to be either a face or non-face. The classifier can also be scaled to detect larger faces. We were originally using the training database used in another CMU course, Machine Learning for Signal Processing, to train our classifier. In order to detect faces tilted at an angle, we had to train our classifier not only on frontal view face images, but also on the tilted face images.

After looking more closely at the complexity of the algorithm, we concluded that it would not be our desired algorithm to implement, since Viola-Jones method requires substantial amount of time for training the classifiers. The actual working system discussed in Viola-Jones paper requires weeks of training time with 5K faces and 9.5K nonfaces for detecting only frontal faces. Given a relatively tight time constraint for our project, we decided it would be better to switch to a face detection algorithm based on skin tone detection like some previous groups had done before. It wasn't until after we changed algorithms and was well into our project that we found out that there were already trained classifiers on OpenCV. Also, Viola-Jones method can only locate the positions of human faces but not the eyes, the new algorithm we use can detect eye locations directly and thus gives us information we are more interested in.

5.3 Final Face & Eye Detection Algorithm

As a result, our new face detection algorithm was be based on Rein-Lien Hsu's paper, "Face Detection in Color Images", which introduces the use of skin color detection and eye map to detect both face and eyes in a image.

5.3.1 Video Pre-processing

The original recorded video, stored in AVI file format, is 160x120 pixels, 24-bit color and runs at a speed of 30 frames per second. Preprocessing the video on the PC first is necessary because processing such a large amount of data all at once would render the system incredibly slow. Thus we need to cut down the amount of data sent to the DSK for processing. The video needs to be broken up into individual frames first so that the DSK board can effectively process one frame of the video at a time and return its results.

The video preprocessing is done with MATLAB. First, we use *mmreader()* to construct a multimedia reader object that can read in video data from a multimedia file. Then we read in all the video frames and get the total number of frames. We then select every sixth frame of the original video and rescale each chosen frame to a smaller resolution of 80x100 pixels in order to reduce the data needed to be processed on the DSK. Through trial and error, we found the size 80x100 pixels works best because it is the lowest resolution our face detection algorithm can tolerate to function correctly. This preprocessing procedure can reduce our processing time significantly since less data needs to be transferred to the DSK for processing. We then create a Matlab movie struct from the video frames. And using *imshow()* we can see any particular frame of the video.

We assume our video frame to be in YCbCr format, and there could be at most two people present in the video frame to ensure qualitative feedback from our system. Mentioned previously, many groups in previous years have implemented face detection based on skin tone detectors, but our algorithm takes one step further to detect eyes within the face region. The reason for our EyeMap is to remove false alarm faces due to the robustness of skin tone detection.

Based on the Hsu's algorithm, our approach to face detection within each frame of video consists of the following steps: creating a binary image via YCbCr thresholding, opening and closing via erosion and dilation, constructing eye map via gray-scale erosion and dilation, and masking the eye map with the skin blob image to identify eyes. [1]

5.3.2. YCbCR Thresholding

First, we use YCbCr color space to threshold our video frame into a binary image. YCbCr is a color space that is often used in video and digital photography systems. The difference between YCbCr and RGB is that YCbCr represents color as brightness and two color difference signals, while RGB represents color as red, green and blue. In YCbCr, the Y represents the brightness component (luma), Cb represents the blue chroma component (blue minus luma) and Cr represents the red chroma component (red minus luma) of the image.

Because we felt that face detection was not the focus of our research, we simply adopted Group 1 of Fall 2008's method in which they created a binary image from a color image by setting pixels in the range of $100<Cb<133$ and $140<Cr<165$ to be 1 and other pixels to be 0. [4] In fact, many research studies found that the chrominance components of the skin tone color are independent of the luminance component [7, 8].

5.3.3. Image Opening and Closing

After the frames are pre-processed, the data is sent to the DSK for processing. Once we have the binary image, we then perform image opening and image closing on our binary image to create blobs of skin regions. Dilation adds pixels to the boundaries of objects in an image while erosion removes pixels on object boundaries. The number of pixels added or removed from the objects in an image depends on the size and shape of the structuring element used to process the image. This basically means that in image opening and closing, we remove noise and artifacts and fill holes that are smaller than a square of 3x3 pixels if a structuring element of size 3x3 pixels is chosen. Through trial and error, we choose our structuring element to be a square of 3x3 pixels in image opening. In image closing, we first dilate it by size 3x3 and then erode it by size 9x9. The reason we would like to do more erosion than dilation in image closing will be explained in the later section. The result of these operations should give us a set of skin blobs as face candidates. We call this processed binary image the face mask.

5.3.4 Holes Filling

The set of skin blobs we obtain after image opening and image closing usually have holes around the eyes and mouths because eyes and mouths do not contain skin color. These holes can be automatically filled by performing image closing if they are small enough. However, large holes can still remain after image closing and make our face mask unable to detect eyes during the final stage of masking. Therefore, we propose a method to fill these holes in a recursive manner. First, we will create a complement image of the original binary image. Since the original binary image has holes around eyes, these holes become blobs in the complement image. We can then label these blobs and calculate the areas inside them using flood fill algorithm. Finally, we can fill out the holes in original binary image if any of them has an area less than the threshold value we set. In our implementation, we choose to fill holes that are less than 50 pixels in the sense that faces that appear in our video shall not be larger than a certain size given the screen range of our camera. Therefore, we assume that eyes that are present on our video shall not be larger than 50 pixels. As the result, we implement three major functions, *bwlabel()*, *regionprops()* and *imfill()*. Function *bwlabel()* labels individual blobs in a binary image, and function *regionprops()* measures the area of each blobs using the labeling outputted by function *bwlabel()*. Finally, function *imfill()* will fill the holes based on the labeling and area returned by the previous toe functions. After this stage, we shall have all skin blobs in the binary image filled, and have completed our face mask.

5.3.5 Eye Map

At this point, we should have sets of blobs. We now need to interpret which blob represents a face and which blob is a false detection. By using Hsu's Eye Map to localize eyes within the image, we can easily determine whether the blob is a face if eyes are located in it.

Although two eye maps, one based on chroma components and another based on luma components, are used in the original paper to create a combined eye map [1], we only use the second eye map, called "Eye Map L" in the original paper, as our source to locate eyes inside an image. Eye Map L is designed to emphasize brighter and darker pixels in the luma components. It is constructed by

EyeMapL = g_ero(Y(x,y)) / {g_dil(Y(x,y))+1},

where functions g_ero() and g_dil() are gray-scale erosion and gray-scale dilation. The gray scale dilation is performed by using the following equation:

$$(f \oplus b)(x,y) = \max\{f(x-x', y-y') + b(x',y') | (x',y') \in D_b\},$$

where Db is the domain of b, a hemispheric structuring element that can be generated using Matlab function *strel('ball',R,H,N)*. We can obtain the value of the structuring element in

Matlab and use it in our C-code. Gray-scale erosion is basically the same process as gray-scale dilation, but it finds the minimum instead of the maximum. We normalize the calculated eye map to the range from 0 to 255. Looking at the eye map in Figure 2, we can see that values around eye regions are highly enhanced, but the values around face boundaries also get enhanced.

5.3.6 Masking

Having both the eye map and skin blob image calculated, we can exclude uninterested parts of our eye map by masking our eye map image with our skin blob image. The eye map we created not only enhanced the regions around the eyes but also the regions around the face boundaries. We prefer that our face mask only mask out regions around the face boundaries to avoid falsely detecting face boundaries as eyes. Thus, back in the stage of image closing, we perform more erosion than dilation. With our eye map properly masked, we can perform thresholding on it to obtain the resultant binary image containing blobs that represent only enhanced part of our eye map, and with further erosion and dilation, we can achieve an image with only the blobs that represent eyes. If there are faces present in our video frame, the final masked eye map should contain small blobs representing the location of human eyes, or else it won’t contain any blobs. We choose a threshold value of 90 to apply to our masked eye map through trial and error. The result is the blob and location of the eyes. Figure 5 shows images of the completed steps as described above.

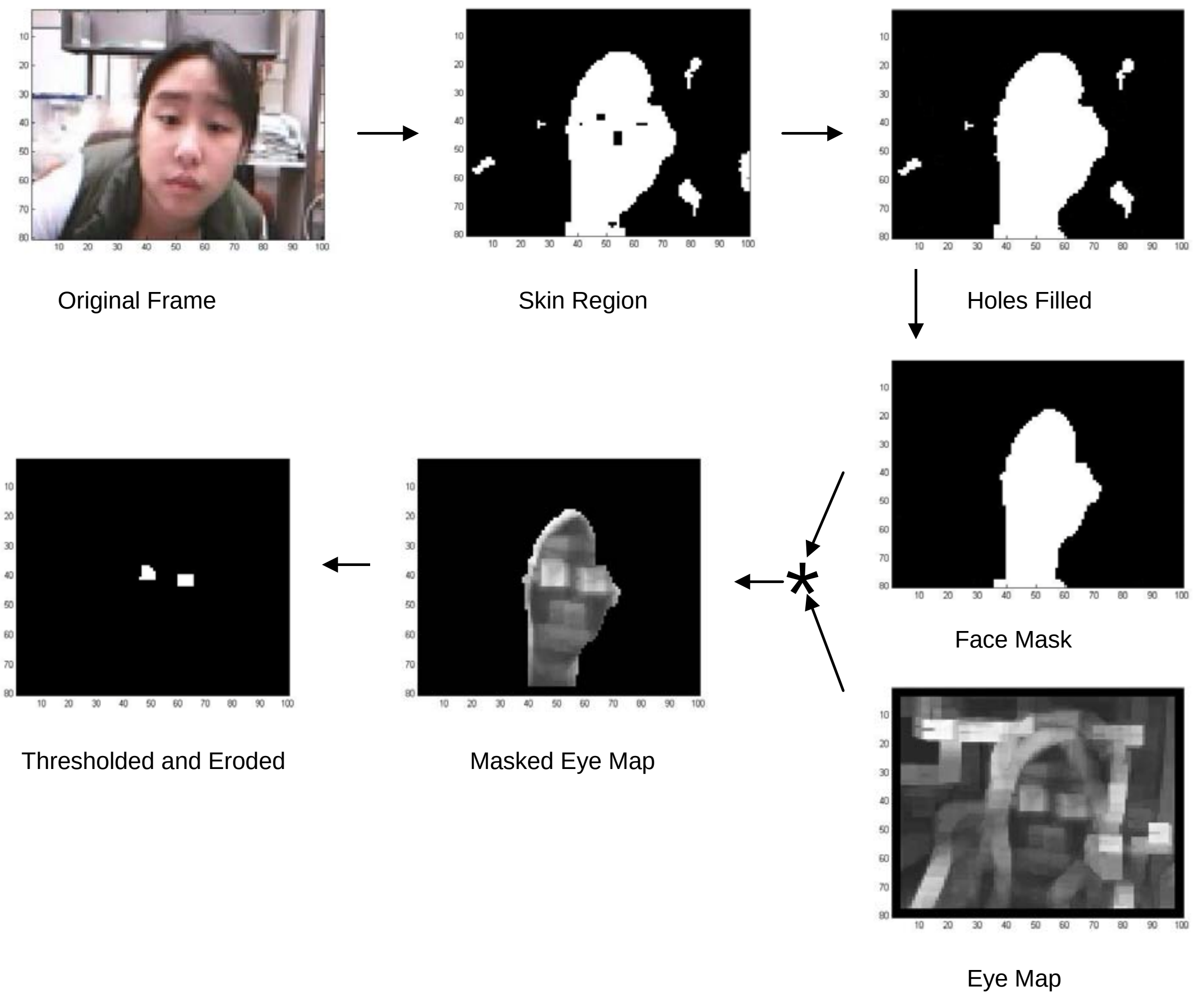


Figure 5. Face & Eye Detection Process

5.4 Iris Segmentation

After locating the faces and eyes of our target, ideally we would then zoom in to the region of the eye to get a picture that has a more detailed image of the iris pattern and proceed to the stage of iris segmentation. Due to the time constraint, we will skip the step of zooming in and pass in the eye images of each of the three group members during the demonstration. Since iris segmentation will be performed on the PC side, we can use Matlab to process the image. We have no intention to optimize this segmentation process as it is slow and we are under time constraint.

5.4.1 Confirm existence of the pupil

The iris segmentation method we used was based on Libor Masek's theory of iris recognition. We slightly modified his open source Matlab code to perform the following steps. First, we would like to confirm the existence of the pupil. To do this, we first need to fill up the color in the specular reflection area by convolving the image with a Gaussian blur filter and then dilating it. Afterward, we will provide a threshold value that represents the color of the pupil and loops through the entire image matrix to label each pixel that has a value less than the threshold (usually less than 60 from the range of 0 to 255) with 1. Eventually, the previous step will result in a binary map, and we can use this binary map to determine the area of the separate image by using *bwlabel()* followed by *regionprops()*. At last, we would like to make sure that the largest object in this picture is greater than a certain size, and if it is, we can confirm that the pupil exists in the picture.

5.4.2 Determining edge and boundary

The second step is determining the edge of the pupil and the iris boundary, that is finding the values of the coordinates. To achieve this, we need to look at the coordinates of each labeled pixel in the binary map created in the previous step and duplicate another one to dilate. After the dilation of the duplicate binary map, we can use the function *edge()* to determine the edge of the objects in the picture (Figure 6). In this case, it will be the edge of pupil and iris. We would like to start from the outer boundary since the pupil and iris somewhat forms concentric circles.

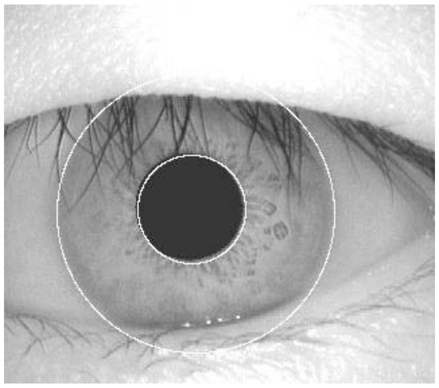


Figure 6. Determining Edge and Boundary

5.4.3 Segmentation

For each of the points that form the edge of the outer boundary of the concentric circles, we would like to know its coordinate, together with the radius from the center of the concentric circle to that specific point. When this process is finished, we can start drawing multiple lines from the center of the pupil to the limbus region and repeat this for the inner boundary by looking at the points that form the inner circle. When obtaining the information, we can convolve the image with a median filter to decrease the intensity of the curvey. The last step is to find the point that has the greatest change after decreasing the curvey intensity and apply Gaussian to model the probability of the found points. Finally we have two sets of coordinates with two radii: one for the inner boundary and one for the outer boundary.

5.4.4 Normalization

When the coordinates of the boundary are found, we can proceed to normalizing the iris region to create a standard template. Normalization is used to compensate the pose of the eyes. This step is part of iris segmentation, so it will still be run on the PC. During normalization, we first calculate the displacement of the pupil center from iris center (we have acquired the boundaries of both, so this step can be easily done). Then, we calculate the radius around the iris as a function of the phase from the center. Note that we would like to exclude the outside rings of the iris as part of the data as it will potentially generate noise. The next step is to find the Cartesian location of each data point around the circular iris region, and extract the intensity of the data point into a normalized polar form using interpolation, or upsampling. Finally, we would again exclude the out-ring points (further reduce the noise) in order to write out the circular pattern, and get the coordinates for circles around both the iris and pupil.

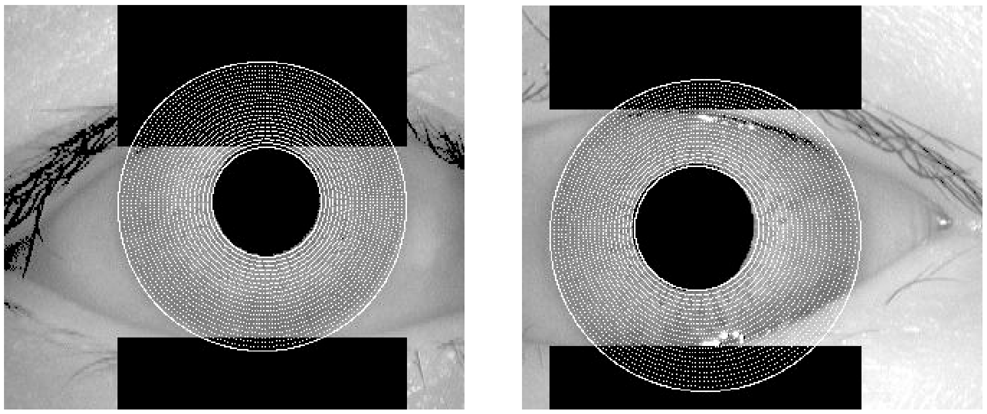

a. Frontal b. Angled

Figure 7. Iris Segmentation and Normalization

5.4.5 Unwrapping

Once normalization is completed, we then unwrap the curved region between the inner bound and outer bound to get a rectangular shape as seen in Figure 8.

Figure 8. Unwrapped

After unwrapping the iris data, it is then convolved with the Gabor filter to encode the template as seen in Figure 9. The size of the template is 20x480 pixels after segmentation, normalization, and encoding. This will be the template we will use for the actual iris recognition.

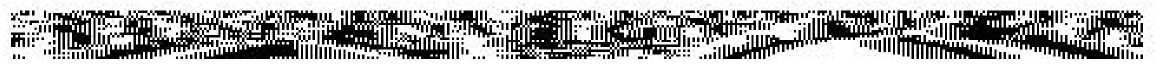


Figure 9. Iris Template

5.5 Iris Recognition

The preprocessing of the iris image mentioned above will be performed on the PC. After the template is successfully created, we pass this template to the DSK immediately. The DSK side contains the function that does the template matching. In order to do this, the iris database must be transferred to the DSK as well. We currently incorporate 30 irises that are a combination of both those released by West Virginia University for research purposes and those taken by ourselves (our own eyes). The basic matching process is to loop through the database and find the hamming distance between the template that is newly created and the ones stored in the database. We would also need a threshold to determine whether this hamming distance represents a match between two templates. As a result, we would like to look at the pairs that have hamming distances less than the threshold, and return the one that has the smallest value of hamming distance.

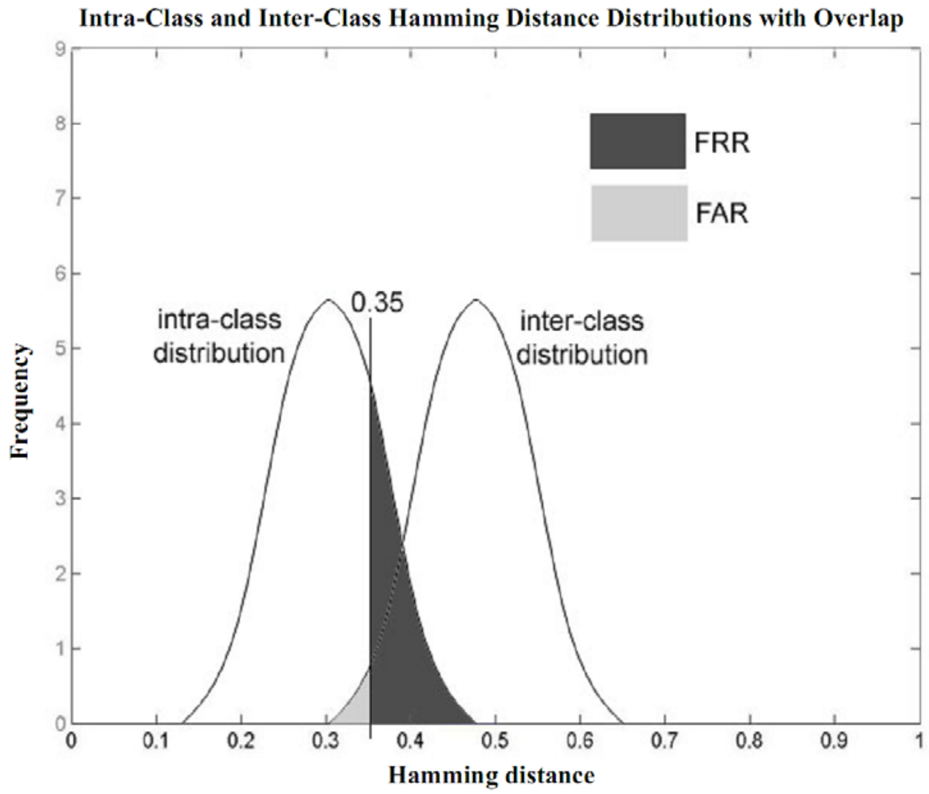


Figure 10. Intra-Class and Inter-Class Hamming Distance Distributions with Overlap

The hamming distance calculates the ratios of different bits and the total bits between two images that have the same size. To count the different number of bits, we utilize the "shift and compare" method. The shifting process performs the switching of the right and left columns of the picture frame. For the first switch, we reverse the two left most columns with the two right most columns, and then we gradually increase the number of column switched to 4, 6, and 8, etc… After each shift, we compare each element of the two frames and divide the total number of different bits with the total number of bits in one frame to get the temporary hamming distance in the current shift, and if the hamming distance found in the next shift is smaller the current one, we replace the current hamming distance with that smaller number. Overall, according to Libor Masek's experiment result, we'd like to perform 16 column-shifting to obtain the best comparison result.

The hamming distances of comparisons between two images belonging to the same eyes and different eyes form two normal distributions as shown above. The hamming distance between two images containing the same eyes will have mostly lower hamming distance than those that contain different eyes. The success of iris recognition depends on the interpretation of the boundary between the hamming distance of frames containing the same eyes and those containing different eyes. We'd like to choose the point that includes the most area of the same iris and the least area of the different irises. In the graph shown above, this boundary has been estimated to be 0.377 (Figure 10). This threshold of 0.377 was determined from trial and error. For a threshold greater than 0.4, everything will be recognized, for anything less than 0.35 there will be too many unrecognized. The possible errors for recognition are the overlapping regions of the two classes in the diagram. The shaded region is the combination of the correctly recognized and every other region. For the threshold we chose, we wanted to make sure that there was enough correctly recognized data and not too many incorrect ones. There is a tradeoff between having enough recognized but not too many incorrectly recognized images. Through trial and error, we found the best balance of correct and incorrect was with the hamming distance of 0.377.

6. Test Results

6.1 Face and Eye Detection

Despite our expectation of our system only detecting eyes in the video, the face detection algorithm we implemented has limited performance. First, because it is based on color thresholding, it is highly dependent on the lighting conditions of the image. The algorithm we use would require appropriate illumination to be robust, or otherwise the face mask that gets created in the first three stages of our system will not be complete. Particularly, we have noticed that it works best for indoors environments and often fails in outdoor scenarios. We found that fluorescent white lighting gives us problems as the skin loses its pinkness and becomes whiter.

The seven video clips we used to test our system were all recorded under appropriate lighting, but our system still suffered from high false positives as the below table display the accuracy rate of our system. Analysis of eye detection result is done on every frame of the testing video to derive the detection rate and number of failed detections are shown in Table 1.

Table 1. Face & Eye Detection Accuracy Rate

Video	Number of people in video	All eyes detected	Detected at least one person	Failed Detection
34	1	87%		0%
37	1	15%		62%
44	1	47%		15%
45	1	52%		47%
36	2	35%	40%	14%
39	2	10%	43%	18%
43	2	27%	34%	4%

As we can see, the detection rate of eyes for two people is a lot lower than that for just one person. Particularly, when there are two people present in the video, our system tends to detect eyes on one face but fails to detect eyes on another face. This behavior can be explained by our choice to apply fixed threshold value instead of applying dynamic thresholding on each skin blob individually. Although the eye map we create enhances eyes on both faces, eyes on different faces usually have different values. This difference in value can be even widened as we normalize our eye map to range from 0 to 255, and thus if we apply fixed point thresholding on the masked eye map, eyes with the lower score are often thresholed out. We once thought of applying dynamic thresholding but eventually gave up because dynamic thresholding requires labeling the face mask to calculate the mean value within each skin blobs. These individual blob analysis are costly in the sense that the overall processing time is proportional to the number of skin blobs in the video frame. Given that we implement our *bwlabel()* function in recursive manner, the processing time of our system will increase even significantly if we apply dynamic thsholding to our algorithm. Since we are designing a real-time system, dynamic thresholding simply kills the processing speed of our system. Possible improvement can be done if we can rewrite our *bwlable()* function in the iterative format.

Another problem our algorithm suffered from is that the eye map not only enhances eye regions, but also partially enhances the regions around nostrils, mouths, and eyebrows. We noticed one group members' nostrils were mistaken for eyes. Usually the eye regions have much higher values than values in those regions, but occasionally those regions have equal or higher values when light is shed on faces differently. As a result, our system also detects those facial features as eyes. This kind of problem can be solved if, again, individual blob analysis can be done to verify the positions of all potential eye blobs in each single skin blob, but we are forced to gave up on this approach when the processing speed of our system comes as our first priority.

The video 37 in our testing data base serves as our major failure case of our algorithm. In the video, Vincent had his hand in a fist and his fist is closer to the camera than his face. Most of the time, the boundaries between his fingers was detected as eyes while the eyes on his face are lost. Such failure explains that objects closer to the camera are usually assigned higher values in the eye map. Again, if dynamic thrsholding can be done on each skin blob individually, we expect the output result can be much improved.

Despite the above problems, our algorithm works pretty well for pose invariance. Even if only one eye is present in a video frame, our system can still detect the eye to determine the presence of a face.

6.2 Testing for Iris Recognition Hamming Distance

The following is an example of the real hamming distances between the images containing the same iris and different irises. We simulate the database we will use in the final demo with 30 iris frames from the West Virginia University database. The database contains images of 7 different irises. The first six irises have four frames each, and the last two images serve as the irises that cannot be found. Figure 11a is an image of the testing iris template (third iris of the database) to be matched.

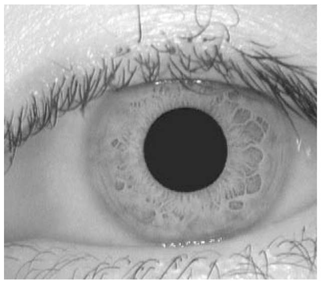
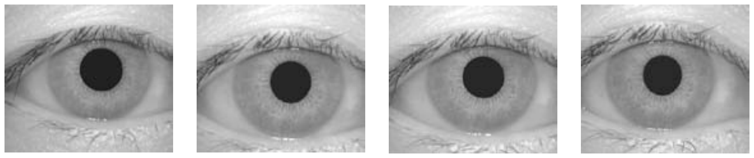


Figure 11a. “Unknown” Iris to be Matched

The hamming distance between the tester image and the database can be seen in Figure 11b.

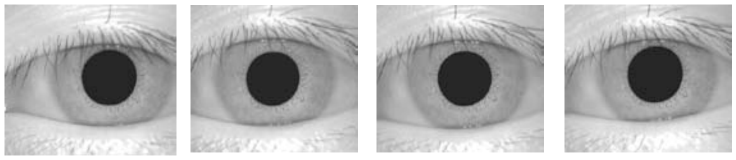
With the first template:

HD
0.43479
0.41687
0.43927
0.44187



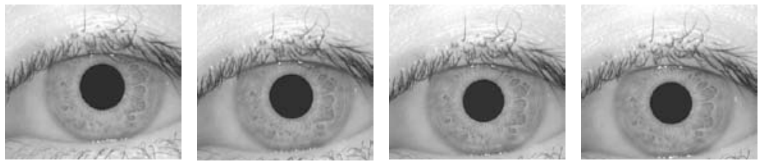
With the second template:

HD
0.4352
0.43395
0.44010
0.44812



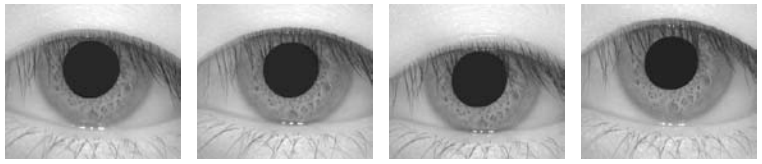
With the third template:

HD
0.26208
0.22885
0.21239
0.23395



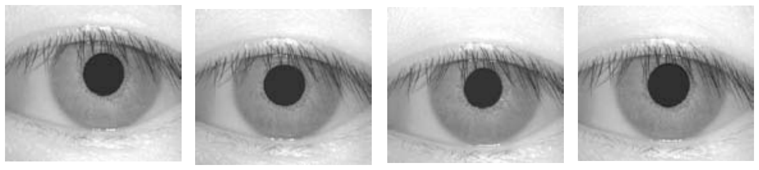
With the fourth template:

HD
0.47312
0.47729
0.48354
0.44802



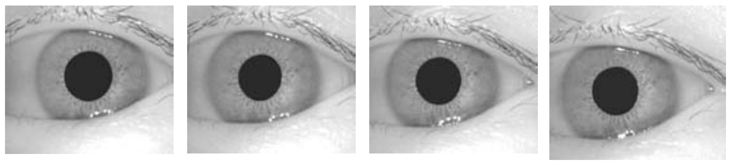
With the fifth template:

HD
0.45020
0.45291
0.45750
0.43343



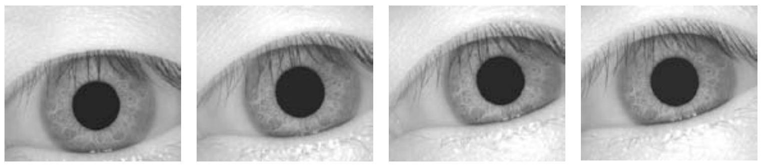
With the sixth template:

HD
0.48583
0.47968
0.47395
0.48135



With the seventh template:

HD
0.48062
0.46468
0.48635
0.47385



With the eighth template:

HD
0.48322
0.47822

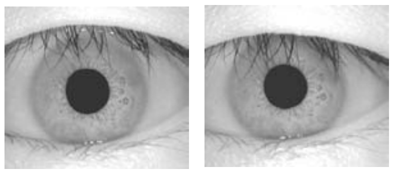


Figure 11b. Matching in Iris Database

The third template class, of the 8, is the correct iris. As can be seen in Figure 11a and Figure 11b, it is obvious that if the templates contain the same iris, despite the angle and a little rotation, the hamming distance will still be below 0.377, whereas the hamming distance of the frames having different irises contain values greater than 0.377.

The calculation time of all these hamming distances ranges from 4.7s to 5.2s. This time includes the file transferring from PC to DSK and the recognition result transferring back from DSK to PC. This number was obtained using timers.h.

6.3 Iris Recognition

After trying 50 images, we found one set of irises that failed (Figure 12). An image is considered to be a failure when the testing image cannot be appointed to its right owner (has hamming distance above 0.377). We are uncertain of the reason for this, but it may be that the database made a mistake. From the two images that did not test correctly, we found that the intra-class hamming distance was around 0.45, which is not normal. Theoretically, an intra class frontal image comparison should result in a hamming distance of less than 0.4. However, among the failed intra class image set, the hamming distance has been calculated to be above 0.45. As a result, we naturally suspect that West Virginia University actually made a mistake when putting up all the images together into the same set. Other than the one image, all inter- and intra- class testing was correct.

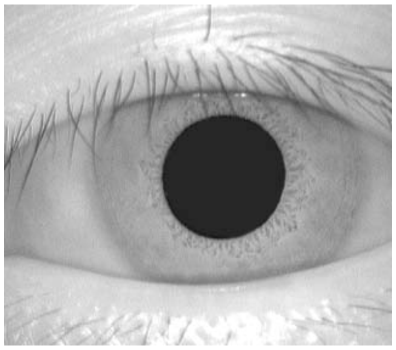
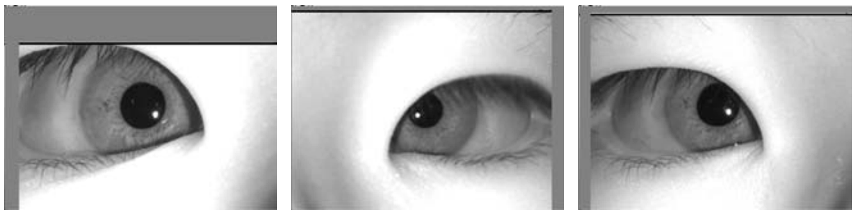
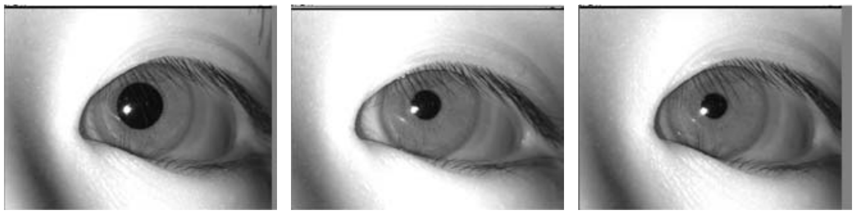


Figure 12. Incorrectly Recognized

After testing our training database, we found 8 out of the 30 images failed. Examples of these images can be found in Figure 13. For Andy and Vincent's eyes, it is evident that the reason those images were unrecognized was because the inner iris pattern was compromised due to the angle of the head. That is, the degree in which the head was turned caused the iris pattern to be incomplete and thus unsuccessfully matched with the training database. As for Stacy's eyes, because the angle turned of the head does not seem as severe as those for Andy and Vincent's, it is speculated that the poor quality of the image (iris pattern unclear) was the reason for unsuccessful matching. Overall, our iris recognition algorithm is fairly accurate.


Andy


Stacy

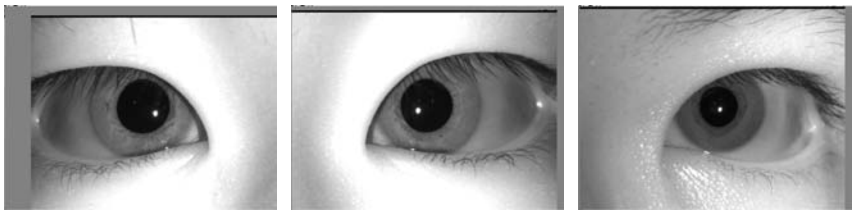

Vincent

Figure 13. Failed Images in Testing Database Example

7. Processing, speeds, data rates

Below is the data flow from PC to DSK. Once again, the yellow is what is being done on the PC and being sent to the DSK. The green is what is being implemented on the DSK and then being sent back to the PC.

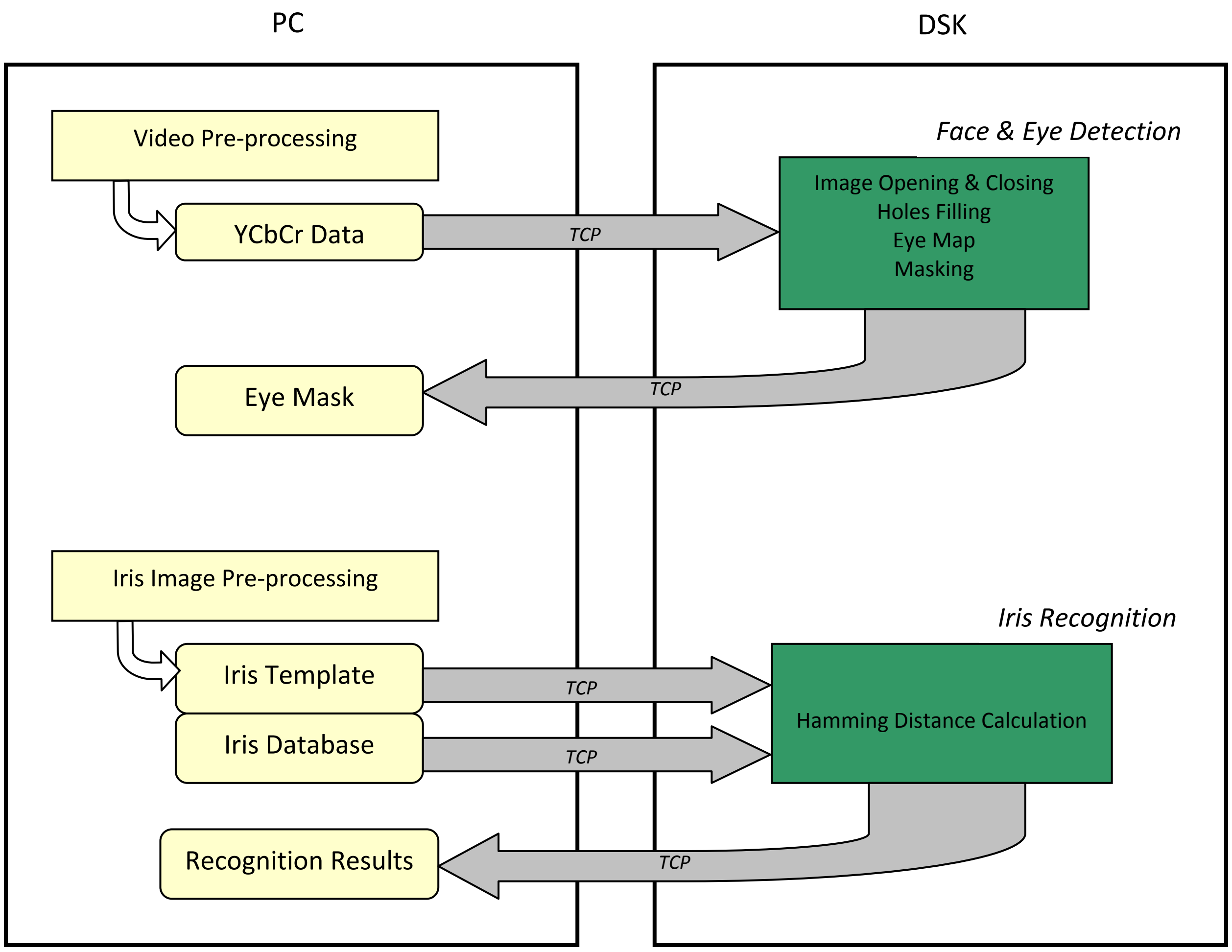


Figure 14. Data Flow

7.1 Face & Eye Detection

The video pre-processing and YCbCr data acquisition were all done in Matlab. The rest of the C code for the face and eye detection processing we wrote on our own based on Rein Lien Hsu's algorithm and was implemented on the DSK.

While our face detection code (in C) utilizes recursion to implement the *bwlabel()* function, which is a function used in Matlab to label all the separate blobs, we found that we had difficulty profiling the entire function. This is a new problem that never appeared in the past groups, and Code Composer does not warn the issue of using recursion on DSK as well. Generally, it is not a good idea to use recursion on embedded board since it is possible that all the stacks will be occupied by that particular function. Although the recursion does not cause direct problem in our final demo but only profiling the number of cycles, it is still worth address this problem to future group. Due to the reason stated above, we can only include the profiling cycles of the iris recognition part.

Because we implement our bwlabel() function in the recursive format, we encounter an unexpected problem while trying to profile our overall system function. Although our system runs normally without profiling, it enters a long hang when we try to do profiling the bwlabel() function. Therefore, we only did profiling on the functions that is irrelevant to the stage of filling the holes. The following is the table of profiled function in clock cycles (Table 2).

Table 2. Profiled Speed for Face & Eye Detection

Function (assumed structuring element of size 3x3 pixels)	Profiled Speed (cycles)
imerode()	505159
imdilate()	598585
g_ero()	35737426
g_dil()	35451217
normalize()	1762302
createEyeMap()	127481178

Both *imerode()* and *imdilate()* are functions that perform morphological operations on the binary image. The algorithm of *imerode()* requires scanning the binary image with the chosen structuring element throughout each pixel locations. If all cells enclosed inside the structuring element are 1, then the center pixel location of the structuring element is set to 0 on the input binary image. Function *imdilate()* is just the opposite operation, where the pixel at the center location of the structuring element is checked, and if the center pixel is 1, all pixels enclosed by the structuring element are set to1. In our system, we use EDMA to page in the binary image we would like to process from external memory to internal memory. Since the overhead for an EDMA request is only 30 cycles, we can ignore it in our estimation. Now, using the estimation time we obtained from lab 3, we know that an average read operation and an average write operation in internal memory takes about 0.69 cycles and 0.63 cycles respectively. We also know that the DSK is cable of doing two data fetches, additions, and multiplications in parallel; therefore, we can simply estimate the function by measuring the time spent on reading and writing data. If we assume to use a structuring element of size 3x3 pixels, the *imerode()* functions requires reading data 9 times and writing data once. Given that our frame is 80x100 pixels, the total time spent on reading data is 9*0.69*8000 = 49680 cycles, and the total time spent on writing data is 1*0.69*8000 = 5040 cycles. Thus the overall estimated speed of *imerode()* is 49680+5040 = 54720 cycles.

The function *imdilate()* is just the opposite operation which requires reading data once and writing data 9 times. Thus the two functions shall have similar processing speed. Notice that we use logical OR in *imdilate()* and use logical AND in *imerode()* to write data. We also do loop unrolling to allow parallel processing of data. The two tricks above combined make our processing speed improves from 1504462 cycles to 505159 cycles. Functions *g_ero()* and *g_dil()* output data as float type in order to compute eye map L in later stage. Since we only perform *g_ero()* and *g_dil()* once for each video frame, we use our internal memory for storing unsigned char only and decide that g_ero() and *g_dil()* can be done in external memory. The algorithm involves finding the maximum (or minimum) neighbor inside the structuring element and thus requires usage of if statements, which slows down the processing speed significantly as we can see from the huge clock cycles in the table above. The trade off we made here was to sacrifice the speed of *g_dil()* and *g_ero()* to improve the speed of *imerode()* and *imdilate()*, which we call them for a total of five times with various structuring element sizes. The total profiled time for our overall system function, *createEyeMap()*, is 127481178 cycles(0.53 sec) not including the cycles required for our *bwlabel()* function. Given the transfer rate we measure at about 0.85 seconds between PC and DSK, we can roughly estimate that the processing time for us to do the whole stage of filling holes in the face mask takes about 0.32 seconds.

7.2 Iris Segmentation

Because we used Libor Masek's open source Matlab code for pre-processing the iris image on the PC with minor changes to suit our project, we did not intend to optimize the section. For the iris picture size of 320x280, the time required to generate a template is less than two seconds.

7.3 Iris Recognition

All of the C code used for recognition, we completely wrote on our own. All of the calculations are done using the on chip memory (L1 cache), thus no EDMA was required for optimization.

The summary of the profiling speed for Iris Recognition can be seen in Table 3. The first total is the total number of cycles required to compare two images. The second total is the number of cycles required to transfer the data from the PC to the DSK for the comparison. For each comparison, we transfer 30 database images, one test image, and compute the hamming distance 30 times. As a result, the total number of transfer is 2550*30 + 60*31, for a final total of 78360 cycles.

Table 3. Profiled Speed for Iris Recognition

Function	Profiled Speed (cycles)
shiftbits()	2232
gethammingdistance()	318
Total to compare two images	2550
requestData()	12
recvDataTransfer()	48
Total to transfer from PC to DSK	60
Total to transfer 30 images	78360

8. Final Demo

For our final demo, we created two new videos of our group members on the spot and performed our face and eye detection on it. Then we chose images in the iris testing database and compared it to the iris training database for iris recognition.

9. Conclusion and Future Work

The main issue we found with our face and eye detection algorithm was its accuracy and ability to detect two pairs of eyes. In the future, perhaps other means such as using dynamic thresholding instead of fixed thresholding can be used. Fixed thresholding works best for one person. If there are two people, the value around one person's eye is different than that around another's eye. So in fixed thresholding, we may lose the other person. However, if we use dynamic thresholding and analyze each blob separately, then scan and calculate the mean and apply different thresholding values, the results should be improved. Due to time limit and trade off between the speed of results (trying to do real-time), this was not further explored in this project.

The main limitation with iris recognition in our project and in the field is the ability to capture good quality iris images. Though we had good iris image data, the camera we used which was specifically designed to capture iris images for iris recognition only allowed us to turn our heads and keep the eyes in front (to ensure good quality iris images). We realized that this is not ideal for real-life situations. Further testing should be in the angle in which the head and eyes together can be turned for iris recognition to still be successful. It would be ideal if a better camera in resolution and ability (if there is one) that could take a video and zoom in, to try and connect the entire process of our project as opposed to the split between face and eye detection and iris recognition. In addition, in our recognition, we transfer each set of iris images for comparison to and from the PC and DSK separately, calculate the hamming distance, then send the data back to the PC for one comparison in the database and start the process all over again for the next image in the database to be compared to. The entire process could possibly be reduced and optimized if in the future we transfer the images in parallel while processing (finding the hamming distance) in parallel. Due to the time constraint, this was not further explored.

In addition, we found that our database was limiting. The training and testing database were perhaps a bit too similar. Though lighting conditions and angles were different, it was not very evident. Further expansion in the iris and video database, such as trying more people, different backgrounds, proportions of people to video etc, and its conditions could greatly increase the real-life application of our project.

Overall, our project was relatively successful, even though we had to make a trade-off between accuracy and speed, as we did accomplish face and eye detection and iris recognition in real-time.

10. Schedule

For our final schedule, we found that we closely followed it for the iris segmentation and recognition part. The face and eye detection part of our project was the more problematic of the two as we struggled to find an algorithm that we could implement successfully and would suit our project for the time and accuracy we wanted.

Table 3. List of Tasks & Schedule

Week	Task	Who
10/12	Viola-Jones implementation with Adaboost.	Vincent, Stacy
10/19	Iris segmentation Matlab construction	Andy
	Continued Viola-Jones and Adaboost Matlab code.	Vincent, Stacy
10/26	Combined face and eye detection using skin-tone detector.	Vincent
	Continued iris segmentation Matlab construction	Andy
11/2	Continued face and eye detection skin-tone and eyemap code.	Vincent
	Iris segmentation Matlab testing.	Stacy
	Iris recognition C code implementation including PC and DSK transferring	Andy
11/9	Translation of face and eye detection Matlab code to C code	Vincent, Andy
	Iris database construction	Stacy
11/16	Implement transferring of face and eye detection to DSK	Andy, Vincent
	Optimization	Everyone
11/23	Finalize all details for demo. Write final report, oral presentation.	Stacy, Everyone
	Continue Optimization	Vincent, Andy
11/30	Oral Presentation, Final Demo, Final Report	Everyone

11. References

[1] Hsu, Rein-Lien and Abdel-Mottaleb, Mohamed and Jain, Anil K. “Face Detection in Color Images”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 696-706, May 2002.

-Face and Eye detection algorithm

[2] Masek, Libor. “Recognition of Human Iris Patterns for Biometric Identification”. Thesis for the Bachelor of Engineering of The School of Computer Science and Software Engineering, The University of Western Australia. 2003.

-Iris Segmentation and Recognition algorithm

[3] Libor Masek, Peter Kovesi. “MATLAB Source Code for a Biometric Identification System Based on Iris Patterns”. The School of Computer Science and Software Engineering at The University of Western Australia. 2003.

- Open source Matlab code for iris segmentation

[4] Aneeb Quereshi, Gregory Tress, David Xiang. “C.A.V.E.S – Content Aware Video Expansion and Scaling”. 18-551 Fall 2008 Group1.

-Used their YCbCr threshold value

[5] Randy Atai, Matthew Brockmeyer, Darren Shultz and Frank Tompkins. “Private Eyes: Biometric Identification by Iris Recognition”. 18-551 Spring 2004 Group 7.

-Reference of past 551 project

[6] Steve Lee, Jeff Mrenak and Lynna Quandt. “The Eyes Have It”. 18-551 Spring 2003 Group 6.

-Reference of past 551 project

[7] J.Yang, W.Lu, and A. Waibel,“Skin Color modeling and Adaptation ”, ACCV, pp.687-694, 1998.

[8] M.Hu, S.Worrall, A.H.Sadka, A.M.kondoz, "Face Feature Detection And Model Design For 2-D Scalable Model-Based Video Coding", ***VIE***, Guildford- London, UK July 2003.