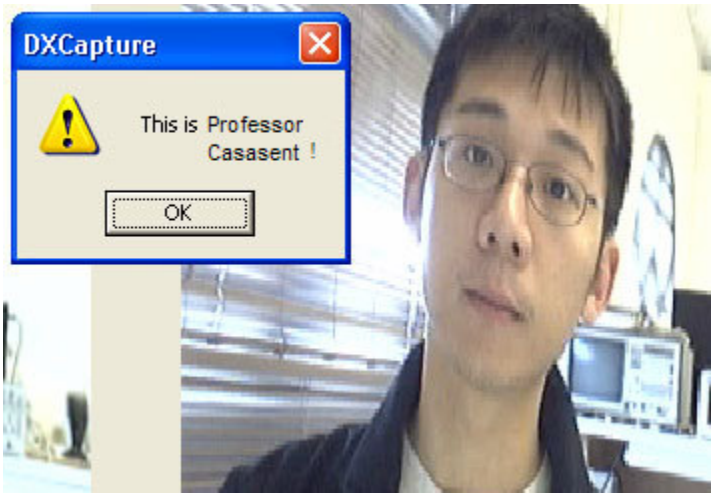
What is your Name?:

A Facial Identification System

Final Report



Digital Communications and Signal Processing Systems Design

18-551 Spring 2005

Group 5

Nancy Chu (nlchu@andrew.cmu.edu)

Dong Keun Lee (dkl@andrew.cmu.edu)

Chee Kiat Ng (ckn@andrew.cmu.edu)

Olivia Tsai (otsai@andrew.cmu.edu)

1. Introduction 4
1.2 The Problem 4
1.3 The Solution 4
1.4 Project Goal 4
2. Prior Work 5
2.1 Previous 18-551 projects 5
2.1.1 ATM Verification System 5
2.1.2 Sleepyhead – Eye Can See You 5
2.2 Uniqueness of our project 5
3. Database 5
3.1 Images from our Database 6
3.2 Preprocessing of the Images 6
3.3 Image Sets 6
3.3.2 The Validation Set 7
3.3.3. The Testing Set 7
3.4 Training and Testing Software 8
make_xxx_norm.m: crops and resizes registered faces to 64 x 64 pixels 8
buildfilter.m: performs the filter synthesis using the filter synthesis algorithm described 8
in the following section 8
pcer.m: calculates the PCER value 8
findpeak.m: finds the peak value to use in the PCER calculation. 8
minace.m: creates the MINACE filter 8
corr.m: performs the correlation 8
recog.m: plots and calculates the true_class, Pc, Pfa, and ROC curves and plots 8
4. Filter Synthesis 9
4.1 Filter Type 9
4.1.1 MINACE filter 9
4.1.2 MACE filter 10
4.3 Peak-to-correlation plane energy ratio (PCER) metric 10
4.4 Automated filter synthesis process 11
5. System Overview 14
5.1 Signal Flow Diagram 14
5.2 Enrollment Stage 14
5.3 Real-time Verification Stage 14
6. Algorithms 15
6.1 Noise Considerations 15
6.2 MINACE Filter algorithm 15
6.3 Eye Detection algorithm 15
6.3.1 Affine Transformations 17
6.4 Determining the PCER thresholds 18
6.4.1 Region of Convergence (ROC) curves 18
7. Software and hardware implementation of the system 22
7.1 Signal Flow (PC & EVM) 22
7.2 Memory Allocation, Data Transfers 22
7.3 Code Optimization 23

7.4 Profiling 23
7.5 Code Usage and References 24
8. Demonstration System (GUI) 24
8.1 GUI Code 25
9. Demo 26
9.1 Results of final demo 26
9.2 Errors which occurred and why 26
9.2.1 Difference of image capturing sources 26
9.2.2 Manual rectangle drawing 26
9.2.3 Optimal face size 26
10. Conclusion 27
11. Future Work 27
12. Acknowledgements 27
13. References 28

1. INTRODUCTION

Our project implements a facial identification system. Specifically, our project is designed to accept an input of a facial image and then output a piece of information about the corresponding person to the user. The use of the single input highlights the difference between our system and a facial verification system, which employs two pieces of information as inputs. The context of the facial identification system would be a classroom setting where a professor would be able to view the entire classroom via a webcam and capture a facial image of a student to be processed by the system.

1.2 The Problem

It is often a daunting task to remember the names of hundreds of students in a lecture hall, even for the sharpest professors. In settings where it is necessary to identify a person, it would be ideal to be able to have a means of obtaining this information without embarrassing oneself by asking for it directly. It is also unnecessary to waste time and energy trying to remember the names of so many people, when this effort could be directed toward more effective lectures and the like. More importantly, this system could be used to ensure that students who are not registered in the class are recognized as imposters and dealt with accordingly. The idea could be generalized to identifying specific people in large crowds provided their information already exists in the system's facial database.

1.3 The Solution

Our project addresses this image processing problem by pausing and capturing an image from the stream of frames collected by a webcam. By using the Minimum Noise and Correlation Energy (MINACE) algorithm, we will create filters that are sensitive to differences in facial expressions, and effectively be able to correlate the real-time facial images with the existing filters to determine whether an identity match exists.

1.4 Project Goal

Our project attempts to be able to correctly identify some students from the Tuesday night lab, who will already exist in our database, as well as correctly reject imposters. In this case imposters are defined as persons not in the database, or in the classroom situation, persons who

are not enrolled in the course. Because there is an expected error rate associated with the effectiveness of the filters, it is our hope that the error rate in correctly rejecting is very low.

2. PRIOR WORK

2.1 Previous 18-551 projects

2.1.1 ATM Verification System

Our project idea is similar to a project done in Spring 2003 by Group 2 [8]. Their group designed and implemented an ATM face verification system that compared whether the claimed identity of a person corresponded with the facial image in the existing database in order to verify or reject the person's identity. The system was designed with a low probability of error in the presence of illumination variation.

2.1.2 Sleepyhead – Eye Can See You

Another project that helped us with our project was the project done by Group 10 [3] in Spring 2004. This group also attempted eye detection; however, instead of using a match filter, we advanced the eye detection a stage further by using a MINACE filter.

2.2 Uniqueness of our project

The main difference between our project and the prior 18-551 project is that our system employs facial identification, not facial verification. Because our system only uses a facial image as an input, the filter selection process is based solely on the image signal. Where the verification system only uses one filter to compare with the input image (selected by the input name), our system needs to compare the input image with all the filters in the database until a match is found. Hence, the processing power is much greater. Also, our project is using manual face selection in real-time as the input image, whereas the previous project input image was not created in real time. Both projects use the MINACE algorithm to create the database filters.

3. DATABASE

Unlike previous projects, our group did not use images from an existing database. Although we considered using the images from the Pose, Illumination, and Expression (PIE) Database [7], we

were not interested in variations of illumination and poses, so we decided to collect our own images with varying facial expressions. Initially, the images were saved in JPEG formats, but after we used MATLAB® to pre-process the faces, the data was saved in .MAT formats.

3.1 Images from our Database

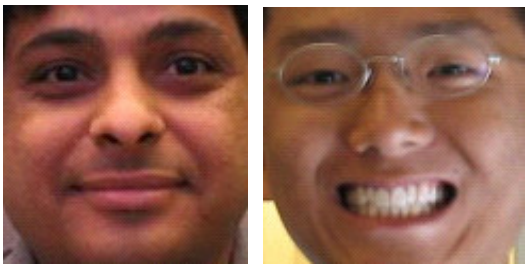


Figure 1: Example of how we cropped the facial images

The images in Figure 1are an example of the images that we used in our database. The images are 64 x 64 pixels in size, 24-bits per pixel.

3.2 Preprocessing of the Images

The images loaded into MATLAB were first converted to grayscale. They were then manually cropped to include just above the eyes and below the mouth and centered based on the eye coordinates, which were manually located. The faces were then down-sampled to 64x64 pixels and used for the MINACE filters.

3.3 Image Sets

We used three different sets of images: the training set, the validation and the testing set. A separate MINACE filter was synthesized for each person.

3.3.1 The Training Set

Our training set consists of 10 different people. For our purposes, these 10 people include our 4 group members and 6 other students from our lab demo section. Each person's filter contains 10 images with the following varying facial expressions, totaling in 100 images in the training set: straight-face (mouth-closed), straight-face (mouth slightly open), smiling (no teeth), smiling (teeth), smile (huge), slightly angry (no

extreme distortion of face), surprised/excited, sleepy, confused, half-smile. These facial expressions are shown below in Figure 2.

Figure 2: The 10 different facial expressions used for the training and testing sets

3.3.2 The Validation Set

Our validation set contains 3 people who are not in the training set. Since we want to be able to detect differences in people over difference in expressions, we minimized expression variation by only using 2 different facial variations, as opposed to our test set, which contains 10 different facial expressions. There are a total of 6 images in our validation set.

3.3.3. The Testing Set

The test set contains images that will be classified as either true-class images or imposter images. There are 10 people in the true-class set which correspond to the same people used in the training set. 10 images were taken of each person with the same facial expressions as the training set, totaling in 100 images in our true-class set.

The imposter set includes 5 people neither in our database or from our validation set. There will be 10 pictures of each of these imposters with expressions as listed in the true-class description, totaling in 50 imposter images.

3.4 Training and Testing Software

We used MATLAB to perform the training and testing on the filters. Listed below are the main files we used to create these image sets and a short description of what each function's purpose. All the MATLAB code has been commented and included in Group5_matlab_code.zip.

get_xxx_eyes.m: gets the eye coordinates by allowing someone to manually click on the images where the pupils should be. The results are stored in a .MAT file be used in **make_xxx_norm.m**. "xxx" represents the subject's name

facenorm.m: uses the eye coordinates from **get_xxx_eyes.m** to register faces

make_xxx_norm.m: crops and resizes registered faces to 64 x 64 pixels

buildfilter.m: performs the filter synthesis using the filter synthesis algorithm described in the following section

pcer.m: calculates the PCER value

findpeak.m: finds the peak value to use in the PCER calculation.

minace.m: creates the MINACE filter

corr.m: performs the correlation

recog.m: plots and calculates the true_class, Pc, Pfa, and ROC curves and plots

4. FILTER SYNTHESIS

A filter was created for each person in the database. Facial recognition was achieved by correlating the input image with each of the filters to obtain peak correlation values.

4.1 Filter Type

We evaluated two types of distortion-invariant filters: the minimum noise and average correlation plane energy (MINACE) and the minimum average correlation energy (MACE) filter before determining which filter would produce the most effective results for our problem.

4.1.1 MINACE filter

We modeled our MINACE formulation based on those described in prior papers [7]. The MINACE filter maximizes the ratio of correlation peak to the correlation plane energy and focuses on separating the noise from the actual signal by minimizing a combination of correlation plane signal energy (Es) and correlation plane noise energy (En). Hence, the filter is designed to give a certain correlation peak value for each training set image included in the filter based on the chosen parameters and constraints of the filter.

The peak constraints are defined by

$$\mathbf{X^H h = u = [1\ 1 \ldots 1]^T}$$

where X is defined as the 2D-FT of the training image set, H represents the transposition of X, and h represents the noise model. Specifically, the energy function is minimized as

$$\mathbf{E = h^H\ Th}$$

where T is defined as

$$\mathbf{T(k,k) = max[S(k,k),\ cN(k,k)]}$$

and

$$\mathbf{S(k,k) = max[S_1(k,k), S_2(k,k), \ldots S_{NT}(k,k)]}$$

The following function is a weighted combination of the two objective functions, where the parameter c is used to determine the emphasis of Es or Ed:

$$\mathbf{E = (1 - c)\, E_s + cE_D}$$

The presence of the control parameter c, introduces flexibility to the filter. The closed form equation of the filter is

$$\mathbf{h = T^{-1}X(X^{H}T^{-1}X)^{-1}u}$$

where **h** is the Fourier transform of the filter, **X** is the Fourier transform of the training set images, **N** is variance of the noise model, and **T** which is the maximum of S1(u,v), S2(u,v),…,cN and the envelope at each frequency (u,v). The parameter **c** is the dc of **N** divided by dc of **S** and **u** is the specified correlation peak values for the distorted object views.

4.1.2 MACE filter

The difference between the MINACE and MACE filter is the value of the c-parameter. The MACE filter is simply a MINACE filter when the c-value is equivalent to zero; hence, eliminating the weighting of the noise component. In this case, only the peak energy is maximized, without minimizing the noise energy of the signal.

4.3 Peak-to-correlation plane energy ratio (PCER) metric

We used the PCER as the correlation match-score metric. The value of the largest peak in a correlation output is used for recognition in many correlation filter approaches. The test image is assigned to the person whose filter produces the largest correlation peak, located in the central 11x11 pixel region of the correlation plane.

$$PCER = \frac{correlation\_peak\_value}{\sqrt{average\_correlation\_plane\_energy}}$$

The PCER, as defined above, is then computed knowing the correlation peak value and the average correlation plane energy.

4.4 Automated filter synthesis process

The first step of the filter synthesis process was to choose the appropriate parameters for our filter. These included the c-value (to determine the weights of minimizing or maximizing the correlation peak energy or the noise energy), the minimum true PCER value (*min_true_pcer)*, the maximum false PCER value (*max_false_pcer*), and the number of images to use in the filters [7].

We first initialized the c-value to 0.001. By performing some initial tests and adjusting the number of images used in the filters we were able to determine an appropriate *min_true_pcer* to be 25. And by testing a false-class validation set against the filters, we were able to determine a *max_false_pcer* constraint to be 12.5.

To determine the final number of images to use in the filter, we first used the first image from the training set to be used to create the filter. We then correlated the remaining images in the training set with the newly created filter to obtain PCER values. If any of the PCER values were below *min_true_pcer*, we took the image with the lowest PCER value and added that image to create the filter again. This process was repeated until all the PCER values were above *min_true_pcer*.

Similarly, to determine the c-value, we correlated the filter against the validation set images. If any of the PCER values obtained from these correlations were above *max_false_pcer*, then we increased the c-value by 1e-5 and repeated until all PCER values were below *max_false_pcer*. Our c values ended up being in the range of 9.8000e-004 to 9.9000e-004, this suggests a MINACE filter worked better for our purposes than a MACE filter whose c-value would have been zero.

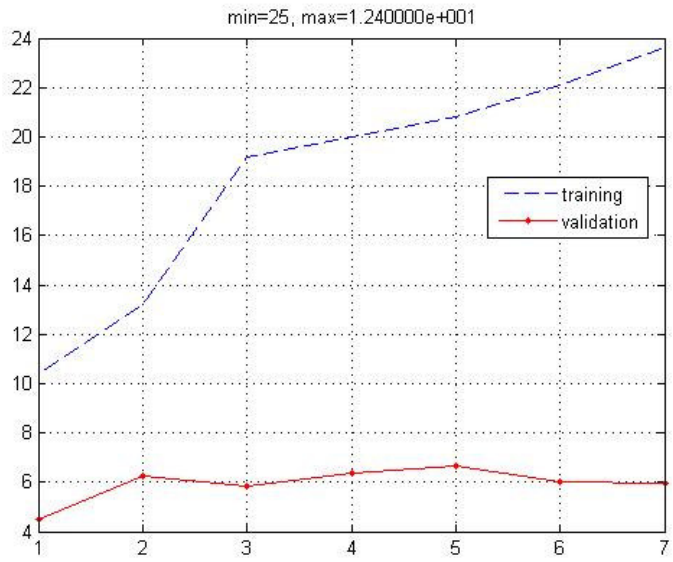


Figure 3: PCER values plotted as a function of the number of images used in the filter

The above Figure 3 shows the PCER values of two different image sets: Professor Rajeev Gandhi's training set and the validation set when correlated with Rajeev's filter. The PCER values are plotted as a function of the number of images used in the filter as determined by the previously mentioned filter synthesis procedure.

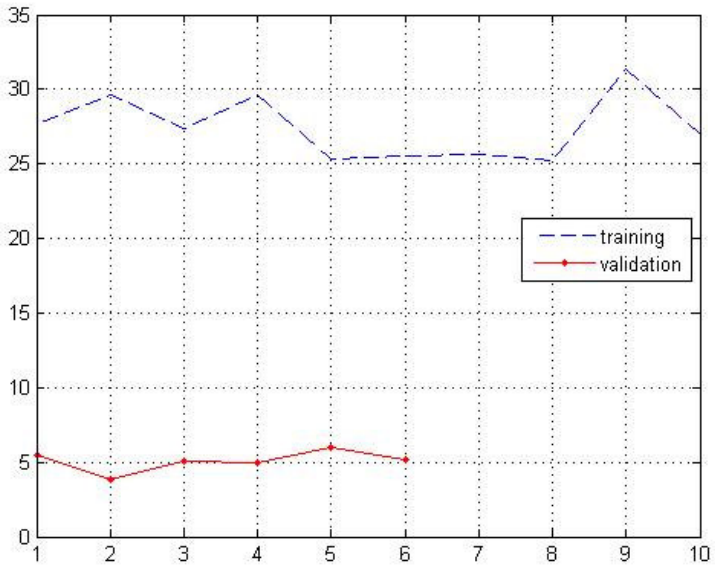


Figure 4: PCER values as a function of the image index number

Figure 4 plots the PCER values obtained when correlating the filter created from filter synthesis with all the images in the training set and the validation set. As shown in this plot, all the PCER values for training set is above *min_true_pcer* and all the PCER values for validation set is below *max_false_pcer.* This filter was created by using 7 images from the training set.

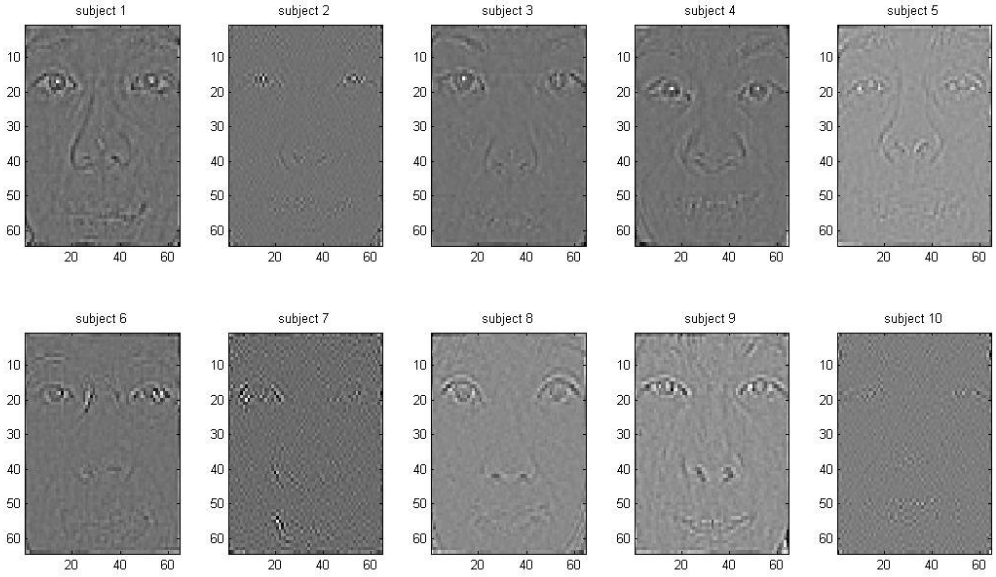


Figure 5: IFFTs of all 10 filters

The above figure shows all the IFFTs for our filters. Note that filters for subject 2 and subject 10 are not as discriminant as compared to other filters. In later sections, we will compare the ROC curve obtained when using all 10 filters vs the ROC curve obtained by excluding Filter 2 and Filter 10.

5. System Overview

Our system will be based on using correlation filtering to determine the name of a student or an imposter, which will represent anyone that does not exist within the current student database.

5.1 Signal Flow Diagram

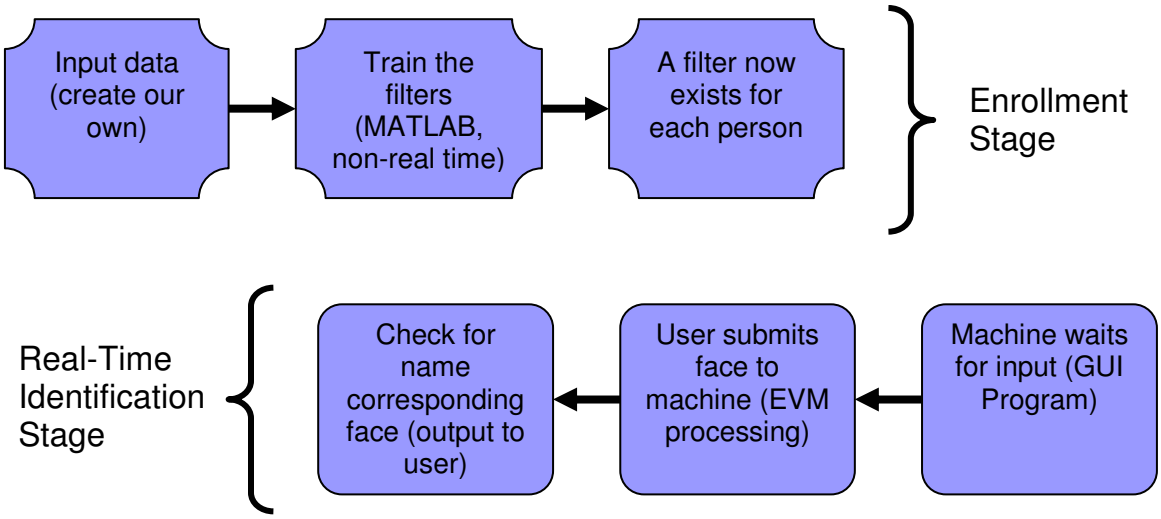


Figure 6: System Overview

5.2 Enrollment Stage

The enrollment stage of our system consists of creating our own input data, training the filters in MATLAB, and creating a filter for each person in our desired database. None of the stages of enrollment are done in real-time.

5.3 Real-time Verification Stage

The real-time identification stage consists of the machine waiting for the input from the user, which is chosen via the GUI program. The user then submits the face to be identified to the machine where the EVM does the processing. The closest match (as determined by the correlation values) is then determined, and the corresponding name is outputted to the user.

6. ALGORITHMS

The following subsections will describe in further detail the algorithms and assumptions we used to formulate the model for our system.

6.1 Noise Considerations

For our expected distortion power spectrum model, we assumed a zero-mean white Gaussian noise, as described in the MINACE filter synthesis section.

6.2 MINACE Filter algorithm

The MINACE filter was found to work better than the MACE filter so we synthesized a MINACE filter for each subject.

6.3 Eye Detection algorithm

In order to have the most successful identification system, it is ideal that we have the testing images sized and oriented the same way as the training images. Hence, our system should be able to automatically resize and rotate the input facial image of interest to be comparable to the training images. Our approach is to use MINACE filters to detect the eye coordinates and then perform affine transformations accordingly. The following figure shows an input image and the resulting IFFT whose peaks represent the locations of the eyes and nostrils.

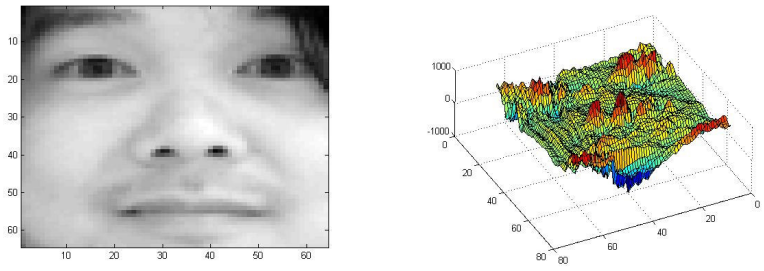


Figure 7: Input image and its corresponding IFFT identifying the location of the eyes

In general, the eyes can be located by creating a MINACE eye filter. The MINACE filter is then correlated with the full image of the face. After doing so, the result will show peaks in regions

where it is most similar to the filter, which are the eyes. To reduce the possibility of error, and the amount of computation needed, we assume that the face can only tilt a maximum of 45 degrees clockwise or anticlockwise. So we need only look at the upper half of the result to locate the peaks so as not to unintentionally identify the location of the nostrils

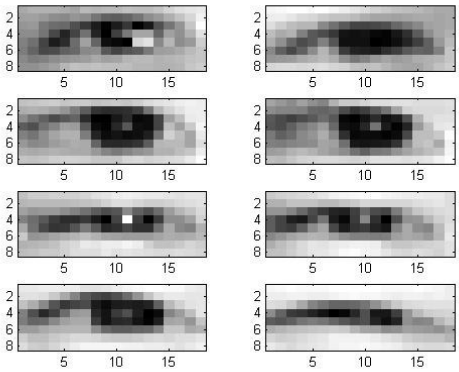


Figure 8: Training images used in the eye filter

We created the eye filter by synthesizing training images that have the most distinct eye features. These features ranged from fully open eyes to half open eyes, to slits. Next we determined the proper c value to use in creating our eye filter. To do so, we conducted the following experiment.

Figure 9: Three test images with different head tilts

Using 3 testing images, as shown in Figure 9, we cropped out the left eye of each image and correlated the extracted left eye with our MINACE filter using a range of different c values. The 3 testing images exploited differences in head tilt: one level, another tilted 45 degrees to the right, and another tilted 45 degrees to the left. The c values used range from 0.999 to 0.5 to 0.001. Figure 10 below shows the cropped left eyes before correlating with the MINACE filters.

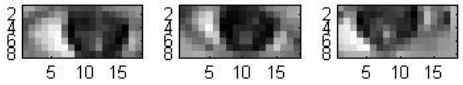


Figure 10: The corresponding left eye of the three different test images

We found that the optimum c value to use was 0.001, which gives us the maximum PCER values of 2.0659, 2.2461 and 1.9916 respectively.

6.3.1 Affine Transformations

Once the eyes are detected, the eye coordinates are used to transform the testing images to correspond with the training images. The following equation is used for affine transformation:

$$\begin{vmatrix} x_2 \\ y_2 \end{vmatrix} = A \times \begin{vmatrix} x_1 \\ y_1 \end{vmatrix} + B$$

Pure translation can be carried out by defining only the B matrix, where

$$A = \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix}, B = \begin{vmatrix} b_1 \\ b_2 \end{vmatrix}$$

Pure rotation can be done by:

$$A = \begin{vmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{vmatrix}, B = \begin{vmatrix} 0 \\ 0 \end{vmatrix}$$

And finally, pure scaling is:

$$A = \begin{vmatrix} a_{11} & 0 \\ 0 & a_{22} \end{vmatrix}, B = \begin{vmatrix} 0 \\ 0 \end{vmatrix}$$

Based on this idea, we use a set of basic transformations that CxImage Library [5] has to perform an equivalent transformation. First, we rotated the entire image so that the eyes were level. We then scaled it such that the distance between the eyes was the same as those in the training image. Finally, we offset the image so that the left eye was aligned in the same coordinates as the training images. All the transformations were done using bilinear transformation.

6.4 Determining the PCER thresholds

The appropriate PCER threshold is set in order to determine the point where any PCER lower than the threshold would belong to imposters and any PCER value above the threshold would belong to those in our database.

6.4.1 Region of Convergence (ROC) curves

Because a maximum PCER value was calculated after correlating the test image with each of the 10 filters, we can use this information to calculate the percentage of correct classifications and the percentage of false acceptances.

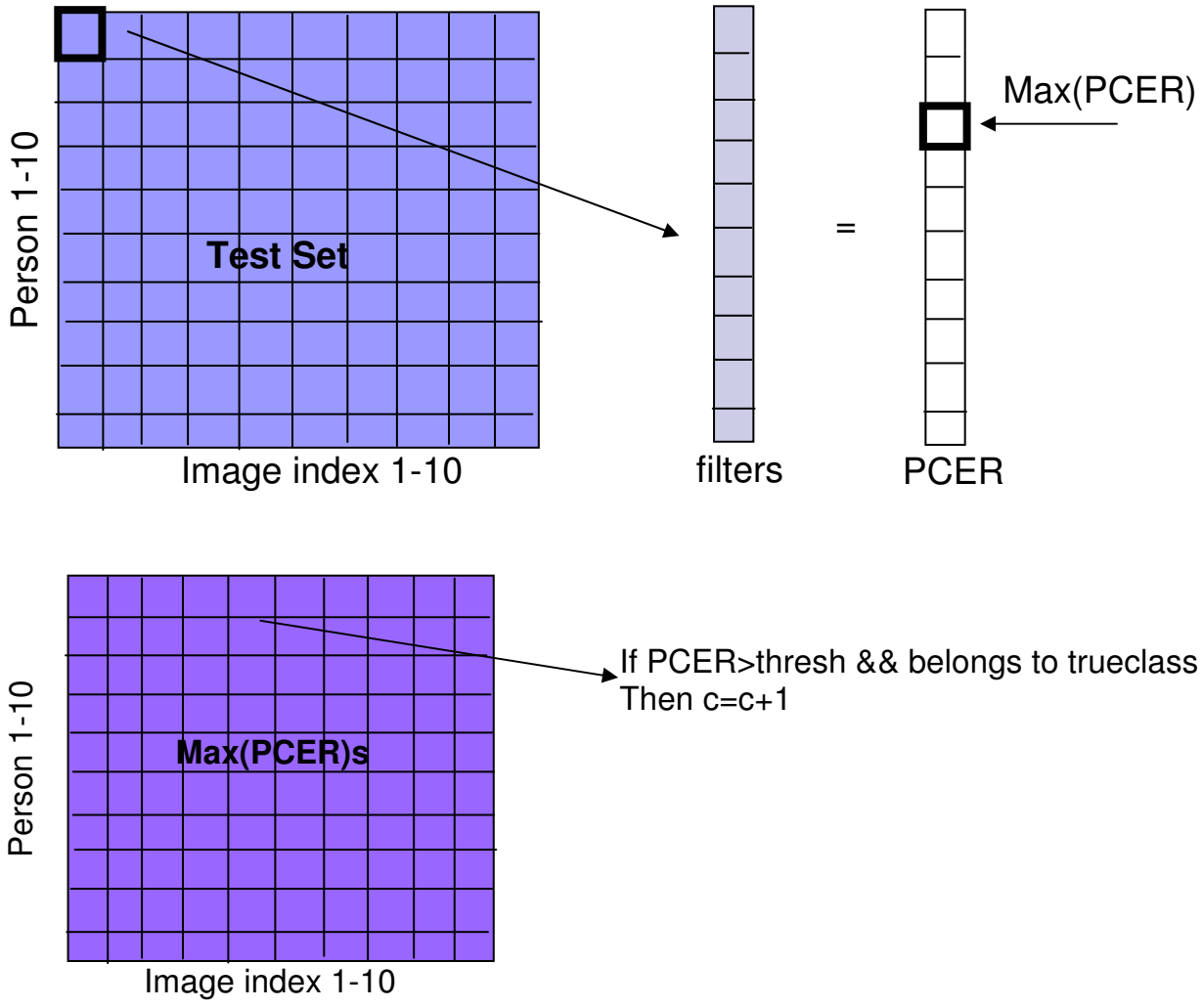


Figure 11: Calculating the Minimum True PCER

$$Pc = \frac{c}{total\_\#\_of\_images\_in\_test\_set}$$

The percentage of correct classifications, Pc, as defined and illustrated above in Figure 11, is obtained by correlating each image in the Test Set with all 10 filters to determine the corresponding maximum PCER value. We are then left with 100 maximum PCER values from each of the 100 images in the Test Set. From these 100 maximum PCER values, we determine the number of correct classification by testing if that PCER value is above the threshold and belongs to the true class. Since the Test Set and the filters are organized in the same order, if their indices are equal, then that image with the maximum PCER belongs to the true class.

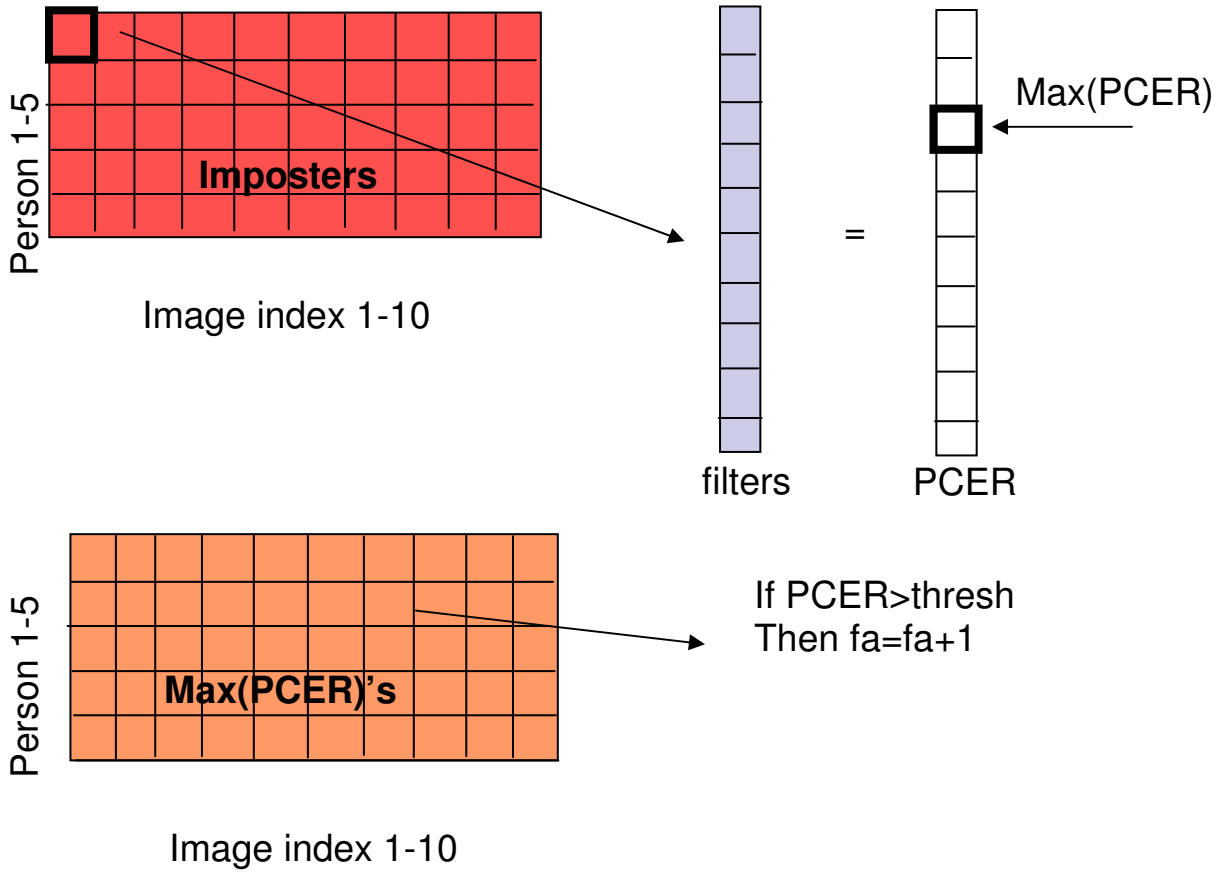


Figure 12: Calculating the Maximum False PCER

$$Pfa = \frac{fa}{total\_\#\_of\_images\_of\_imposters}$$

Similarly, the percentage of false acceptances, P_{fa}, is determined by the number of total false acceptances, *fa*. This number is determined by correlating each image from the Imposters Set with the 10 filters to determine the maximum PCER value for that image, as shown in Figure 12. This would then result in 50 maximum PCERs from the 50 images in the Imposter Set. P_{fa} is then determined by counting the number of imposters who result in maximum PCER values greater than our determined threshold. Note, when calculating Pfa, it is not necessary to determine if the image with the maximum PCER belongs to the true class.

6.4.2 ROC Curve

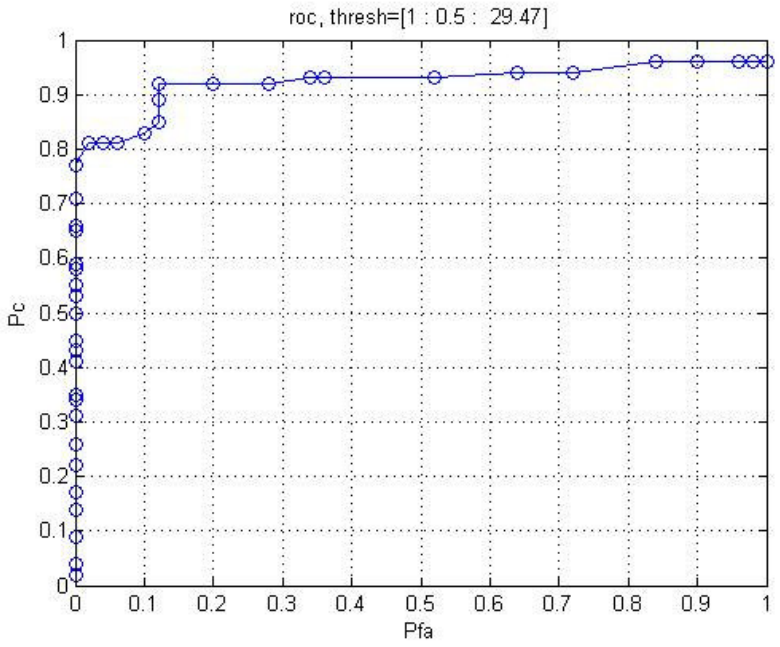


Figure 13: ROC Curve

By choosing the threshold of 14.5, there is an 89% probability of returning the right name, but only a 12% probability of a false acceptance.

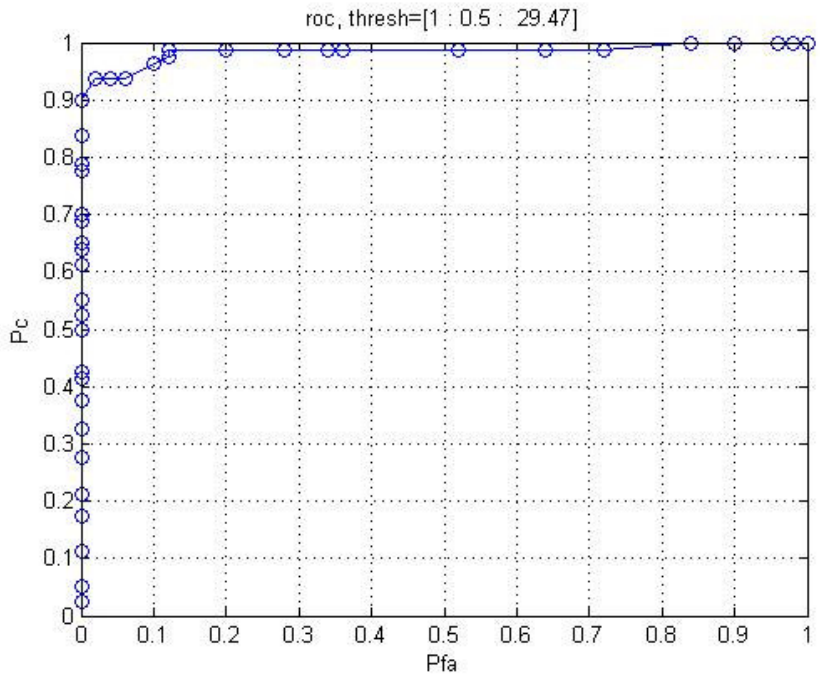


Figure 14: Improved ROC Curve

The above ROC curve was obtained after excluding Filter 2 and Filter 10 from our database. Just as expected, the ROC curve shifts up, giving higher values for Pc values. We can then choose a threshold of 16, which will produce a 96% probability of returning the right name, and only a 10% probability of a false acceptance.

7. SOFTWARE AND HARDWARE IMPLEMENTATION OF THE SYSTEM

7.1 Signal Flow (PC & EVM)

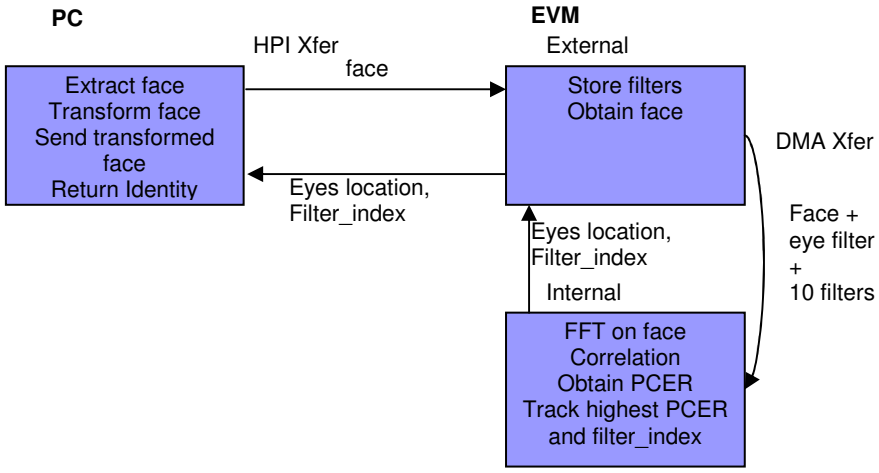


Figure 15: PC to EVM System Diagram

Figure 15 depicts the interaction between the PC side of the system and the EVM side. The PC performs the functions of extracting, transforming, and sending the face and returning the identity of the person. The EVM stores the filters, obtains the faces and performs the following functions: FFT on the face, correlations, PCER calculations, maximum PCER tracking, and filter index tracking.

The EVM performs the following steps. First, a two-dimensional Fast Fourier Transform (2D-FFT) is performed on the input image. The result from the 2D-FFT is then correlated with the image filters. Next, a two-dimensional inverse Fast Fourier Transform (2D-IFFT) is performed on the correlated matrix, and a FFT-Shift is performed. Finally, the PCER value is calculated and the filter number which has the highest PCER value is sent back to the PC.

7.2 Memory Allocation, Data Transfers

The following table shows the various memory usage on the EVM.

Table 1: Memory Requirements

Input images	64 x 128 bytes = 8 kB
Each filter	64 x 128 x 4 = 32 kB
2 images	16 kB
10 slots for filters	320 kB
Twiddle factor	64 x 2x 4 = 512 bytes
Digit reverse	64 x 4 = 256 bytes
IFFT result	64 x 64 x 4 = 16 kB
Output file size	128 kB

Since the data type of the image points are of type unsigned char with size 1byte, the memory requirement for the input image is 8 KB (64 pixels x 128 pixels x 1byte). The data type of other data elements are float of size is 4 byte, so we need to multiply the number of elements by the float size.

In order to minimize the memory usage, we performed the FFT, correlation, and transpose functions in place. Because the EVM code contains several versions of FFT and correlation functions, the total EVM code size is 16 kB.

7.3 Code Optimization

To optimize our code, we unrolled the code of the 2D-FFT and the correlation functions by a factor of 2 and we used a DMA transfer for the 2D-FFT function. The following section demonstrates the performance of our system.

7.4 Profiling

The following table shows the results of profiling the functions used on the EVM.

Table 2: Profiling of Functions

	SBSRAM	On-Chip
2D-FFT	12,112,714	5,992,495
Unrolled FFT	7,058,083	4,875,125
Unroll & DMA FFT	6,425,421	N/A
2D-FFT(asm FFT)	6,607,736	3,964,337
Correlation	1,895,070	1,321,418
Unrolled Correlation	1,359,600	1,012,094

We found that the results of profiling different functions varied widely in the number of cycles taken, depending on which computer was used at the time of profiling. The above profiling data are the best results that we have obtained from our trials. The table shows that most function performances ONCHIP were better than those on SBSRAM. In the case of the 2D-FFT, we used unrolling and DMA transfers to obtain better performance. However, we also realized that the DMA transfer on ONCHIP made the performance worse in trying to profile 2D-FFT function after the demo, while the DMA transfer on SBSRAM resulted in a better performance. As Professor Casasent recommended, we changed the 1D-FFT function "cfftr4" from the C-equivalent of the assembly code to solely the assembly code to get optimal performance. We tried to optimize the 2D-FFT using 1D-FFT assembly code by utilizing unrolling and DMA transfer, but we could not obtain better performance. The reason why the performance of the uncontrolled Correlation is approximately three or four times faster than that from 2D-FFT is that the 2D-FFT function requires more memory access than the correlation function. The 2D-FFT has a lot of "for" loops because the function transforms the row of input matrix, transposes the result, transforms the transposed matrix again, and then transposes the matrix, whereas the correlation function has only double nested loops.

7.5 Code Usage and References

We made use of the 1D-FFT function from the Spring 2005 Lab 2 to perform the 2-dimensional Fast Fourier Transform.

8. DEMONSTRATION SYSTEM (GUI)

A graphical interface was necessary in order to allow the user to capture the image of the face to be processed. This GUI was designed using Microsoft Visual C++ [4], allowing the user to view the classroom through a webcam, pause when necessary, and extract a face by drawing a rectangle around the desired face using the mouse. Using the Microsoft DirectShow SDK [6], one window showed the continuous live stream of the classroom through the webcam and another window displayed the captured frame. The extracted face was then cropped and resized to a 64x64 image and sent to the EVM to locate the eyes and then transform the image correctly by eliminating tilt and rotation of the face before sending the signal to the EVM for identification.

Once the EVM completed its task, the program would go through the correlation results and return the most likely identity of the person. If the program could not detect a decent identification of the person, it notifies the user that this is an unknown person in the class. A Logitech QuickCam Pro 4000 was the webcam purchased and used in our system.

8.1 GUI Code

The GUI template was obtained from "The Code Project". Other useful references were the CxImage library [4] and Microsoft DirectShow. The code to draw the rectangle was written from scratch.

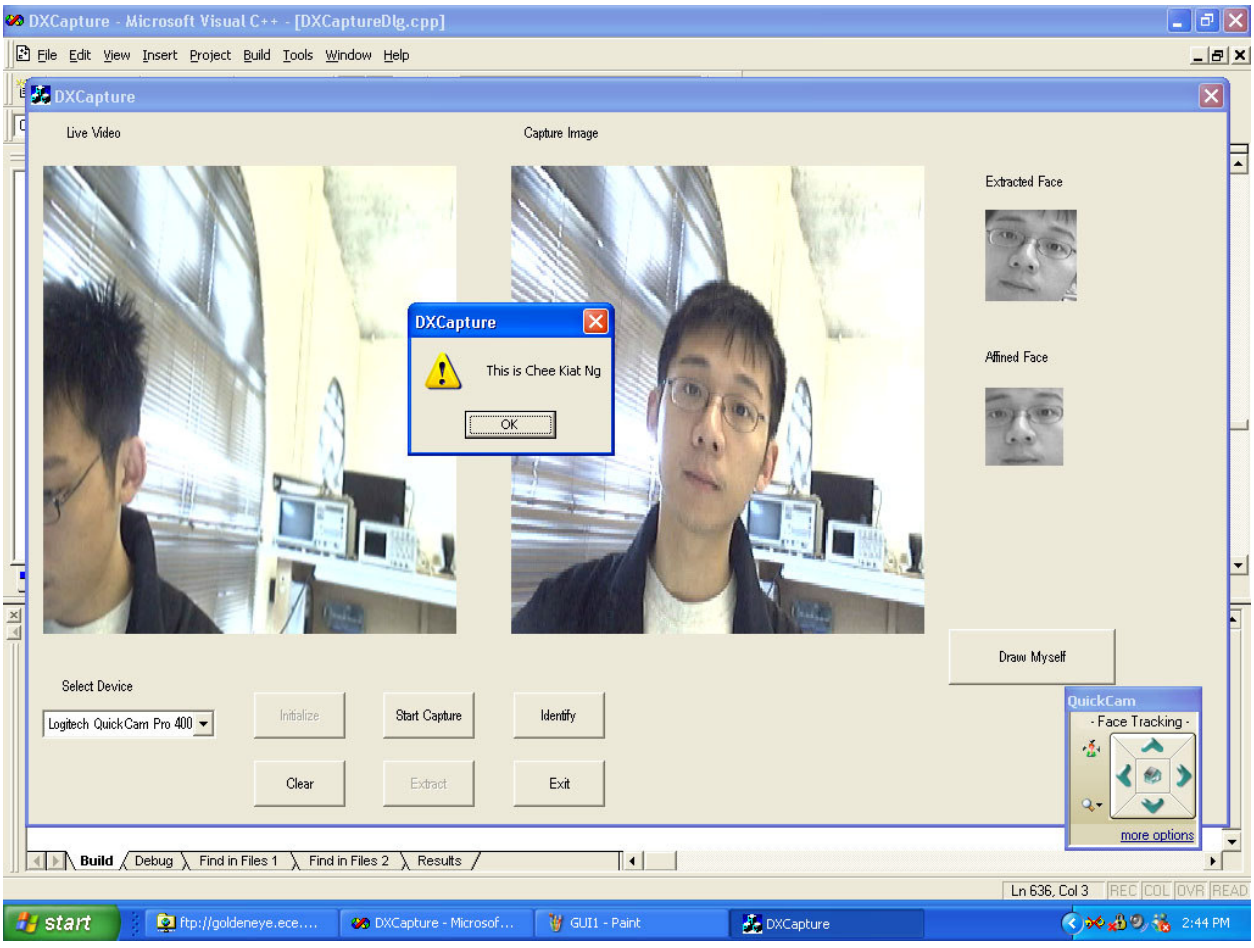


Figure 16: Graphical User Interface

9. DEMO

9.1 Results of final demo

The system resulted in numerous false acceptances and wrongly rejected true-class subjects. In many instances male imposters were identified as females from our database.

9.2 Errors which occurred and why

The following section discusses some errors that occurred during the demo and the sources of some of these errors. Some possible solutions are also suggested.

9.2.1 Difference of image capturing sources

We can attribute many of our problems to the difference in sources of the training set images and the testing images. The training set images used for the filters were taken with a Sony CyberShot DSC-T1 5.0 mega-pixel digital camera. The testing set images captured in real-time were obtained via a Logitech QuickCam 4000 webcam. The difference in quality and image resolution of the devices affected our system's ability to effectively correlate a testing image with its corresponding filter and output a high PCER value.

9.2.2 Manual rectangle drawing

Other problems stemmed from the drawing of the rectangle and the fact that the manual cropping of the face in real-time would affect how well the testing image correlates with the existing filters.

9.2.3 Optimal face size

Optimal face identification resulted when the face size captured by the webcam was approximately 1/3 the size of the screen. Since the screen is 320x220 pixels, a face 1/3 the size of the screen would be about 64 pixels wide, which is the resultant size of cropping and resizing.

10. CONCLUSION

Although there were a few mishaps during our demo regarding the quality of one or two filters, which resulted in identification problems, once we removed these "problem" filters, we were able to achieve our expected results. We are also very optimistic that if the suggestions and minor changes discussed in the following section are implemented, the system would be more effective.

11. FUTURE WORK

There are several improvements that could be implemented to make our system more successful. For one, when creating the training set, the different expression variations should be minimized and replaced by normal, less extreme facial expressions. Also, all training and test images should be normalized to have unit energy before filter synthesis and prior to performing correlations. Because our project did not do this, future work could explore this possibility.

Also, our system used a single frame capture to obtain the input image. Future projects can utilize video sequencing capture to handle multiple frames instead of just one.

12. ACKNOWLEDGEMENTS

We would like to thank Rohit Patnaik, Professor Rajeev Gandhi, and Professor David Casasent for their assistance and feedback regarding our progress. We would also like to thank the people who volunteered to be in our database. Without your images, we could not have created our filters!

13. REFERENCES

Affine Transformation Theory (reference just used for theory purposes, no code provided):

[1] - Jain, Fundamentals of Digital Image Processing, Prentice-Hall, 1986, p 321.

[2] - Geometric Operations - Affine Transformation (http://www.cee.hw.ac.uk/hipr/html/affine.html), last accessed May 2, 2005.

Eye Detection Theory:

[3] - Chaudhry, A., Jayakumar, M., and Maniar, J. *Sleepyhead - EYE can see you.* 18-551 Spring 2004 Report, Group 2.

Graphical User Interface:

[4] - GUI template from "The Code Project" (http://www.codeproject.com)

[5] - CxImage library (http://www.xdp.it/cximage.htm)

[6] - Microsoft DirectShow (http://c2.com/cgi/wiki?DirectShow)

MINACE filter (MINACE code written was based on the equations and theory from the following references):

[7] - Patnaik, R., and Casasent, D. Illumination invariant face recognition and imposter rejection using different MINACE filter algorithms.

[8] - Edmondson, Z., Patnaik, R., and Zelinski, A. *Face Verification For ATM Access.* 18-551 Spring 2003, Group 10.