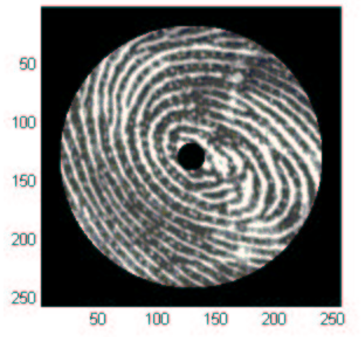
Digital Communication and Signal Processing System Design

Spring 2002

# Fingerprint Verification

# Final Report

Submission Date: May 6 2002



**Group 6**

Cameron Boone (cboone@andrew.cmu.edu)

Faisal Tariq Butt (ftb@andrew.cmu.edu)

Naseer Siddique (nss@andrew.cmu.edu)

Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213

# Table of Contents

## 1.0 Introduction

The field of biometrics, or the identification of people based on physiological or behavioral characteristics [6], has recently received a significant amount of attention in current research as a result of its applicability in the fields of information technology and security. With the increased usage of PIN numbers and passwords in everyday life, the vulnerabilities of these two technologies are becoming more apparent (e.g. password-cracking).

Biometric authorization's strengths are that it requires the user to be present and that it eliminates the hassles of passwords and PIN's (or can be used in parallel with these for added security). Specifically, online identity verification, ATM machines, and building entrance authorization, are all areas where it is very useful to implement an automatic identification system.

Some of the technologies that have been investigated are fingerprinting, iris scanning, facial scanning, voice detection, and palm scanning. Of these, fingerprinting has been in development for the longest, and while the other technologies have attracted more recent attention in journals and research groups, we are focusing on the ability to conduct robust fingerprint verification with minimum overhead. The FBI can easily afford the most fancy new technology, but at the consumer and civilian level, price is a great limiting factor, which is why digital video technologies for biometrics have yet to make much headway in the market.

The purpose of our project was to design a fingerprint verification system that would reliably verify the identity of a person by matching the scanned fingerprint with a database (that contains images of all the people in the system). The goal of the project was to implement the entire computational portion of the fingerprint verification process on the Texas Instruments C67 DSP board and optimize it for eventual implementation in a real-time on-line system. This report describes how we implemented this system, followed by an analysis of the results obtained.

## 1.1 Fingerprints

A fingerprint is the pattern of ridges and valleys on a person's fingers [1]. What makes this a good biometric is that to date, no two people have been identified that have the same fingerprints. The two main applications in fingerprinting are identification and verification [8]. Identification asks the question, "Who is this?" For instance, if fingerprints are found on a stolen object and the objective is to determine whose prints they are, identification must be used to check against all of the entries in the police database. Typically, due to the immense processing needs, identification is not done in real-time. This project, on the other hand, will attempt to build a verification system, which asks "Is this person who he/she claims to be?" For example, if Jane wants to enter a building, she can type her name into a computer console, and then press her finger to the touch pad, and the computer will verify that indeed her fingerprint matches the reference print from the computer.

## 1.2 Distortion

Distortion, or the differences in fingerprints due to pressure and movement of fingers during image capture, is one of the largest obstacles in most fingerprint identification schemes [5]. The problem is demonstrated in an examination of multiple fingerprints from the same individual, each of which is not identical to the others. When more pressure is applied, some features are highlighted while others are eliminated. Similarly, the orientation of the entire fingerprint is different for every sample, depending on the position of the finger, which makes it difficult to find a common reference point (center point) from which to compare. Finally, as with any image capture scheme, there is the issue of noise, which is the data in the signal that does not hold any interpretive value but is rather introduced by the imaging hardware and the medium (e.g. ink, paper). For the above reasons, a bulk of the work that will be done in this project will be pre-processing of the images. Through this pre-processing, we plan to enhance the image details so as to perform better comparison between prints.

## 1.3 Previous Work In The Field

There has been a significant amount of research done in the fingerprint recognition field to date, and it appears that most of the research has been coming out of Michigan State

University under the auspices of a certain Dr. Anil Jain and his former Ph.D. students who are currently working at the T.J. Watson IBM Research Center. We researched and read most of the current published IEEE journal articles on fingerprinting, and while there is much discussion of various methodologies and results, there is little to no published code, presumably a result of patents and intellectual property concerns.

Until now, there have been two different groups that pursued projects in fingerprint recognition. The first project was done in Spring 1999, when a group implemented a fingerprint identification system on a SHARC board (even though it is an older board than the TI C67 that we used, it is still faster). They faced major problems as a result of an inefficient center point determination algorithm; this resulted in problems identifying fingerprints of the same class (i.e. whorls, left loops). This whole project was based in the frequency domain. The problem was very inefficient computationally, and could not be extended to large databases without losing real time processing time.

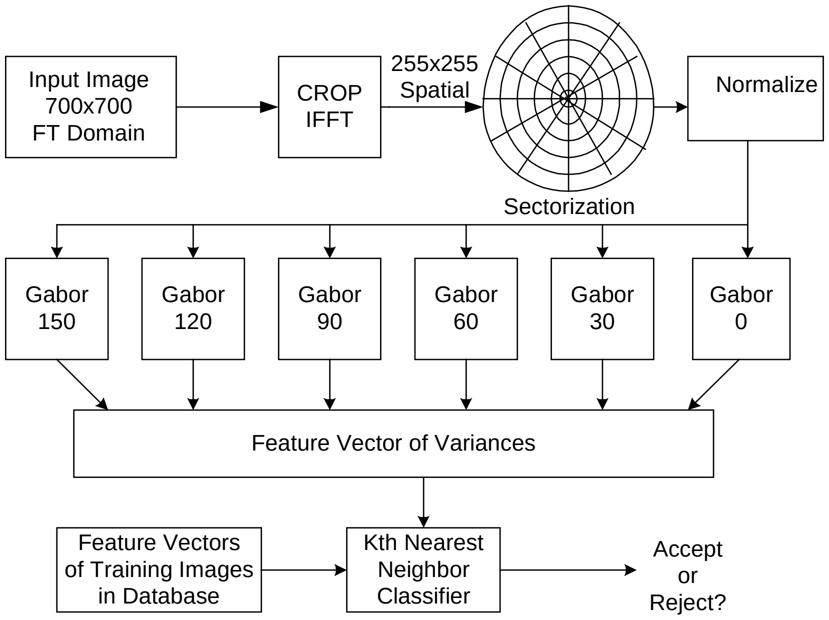
The second project was in Spring 2001, when another fingerprint identification system was implemented. This group used the same center point determination algorithm as the previous group, and faced the exact same problems as the previous group. This project was based on the spatial domain. Also, very limited testing was done to check on the reliability of the system. Finally, the system was very inefficient computationally, and could not be used for databases of a large scale.

There were some major differences between our project and the previous ones. Firstly, center point detection was already done for us (since our images were pre-centered and pre-normalized), though we did verify it by hand. Secondly, we are designing a verification system, as opposed to an identification system. Finally, we aimed to make a scalable system that could handle databases of arbitrary size.

## 2.0 The Algorithm

Our system will be able to verify whether a person is who he/she claims to be with a high rate of success. The system we have designed was divided into three stages: pre-processing of the fingerprint image, extraction of the features representing the fingerprint, and classification of the fingerprint for a final verdict.

## 2.1 System Diagram



The algorithmic flow for our project is the following: First, a 700x700 test image is input to the PC where it is cropped to 255 x 255 pixels. The cropping is performed such that the middle 255x255 square of pixels centered on the center of the image is retained and the rest of the image is removed. The cropped and normalized image is then fed from the PC to the EVM for the remainder of the preprocessing. This preprocessing consists of sectorizing and normalizing the image, spatially convolving the image with six pre-stored Gabor filters, and finally performing variance calculations on the filtered images to form the feature vector for the image. Once the feature vector is obtained this information gets passed from the EVM back to the PC where it is compared with the pre-stored feature vectors of each fingerprint in the database. The Euclidean distance between each pair of fingerprints is calculated and compared to a pre-defined threshold applicable to the input print. A K-Nearest Neighbor classifier is used to determine which person in the database best matches the input fingerprint, and if this is the same person who claimed to be entering the system, then that person is verified. If not, the subject is rejected and security is notified. A more detailed description of each step involved in the algorithm follows.

## 2.2 Pre-Processing/Distortion Compensation

A) Domain Conversion and Cropping

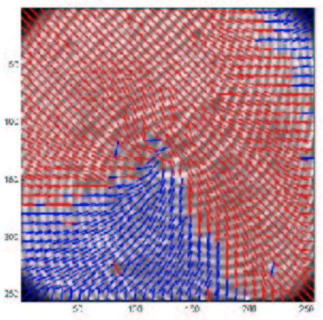
The first part of the pre-processing consisted of converting the files from the FFT domain to the spatial domain and cropping the images from 700x700 pixels to 255x255 pixels. Both of these steps were done in Matlab. The files needed to be converted to the spatial domain so that they could be sectorized and normalized properly, and also for the Gabor filtering and feature extraction in the later stages. By cropping the images to 255x255 pixels and later padding them with one row and one column of zeroes we would also be able to attempt to implement the Gabor filtering stage in the frequency domain using a 256 pt FFT.

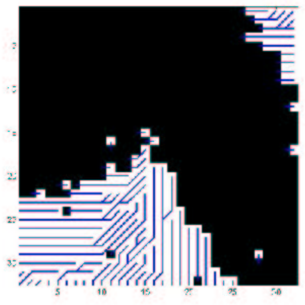
B) Center-point Determination

In pre-processing, our main objective is to reduce the effects of distortion. Even if a good classifying algorithm is used, the results can be easily rendered meaningless if the test image is not oriented identically to the reference images. The center point, once found, is used as a reference point throughout our processing of the image. Specifically, it is needed for accurate sectorization and filtering. The center point is located at the point of maximum curvature of ridge lines. Thankfully, the images in our database were already centered, however at the request of Professor Casasent we ran the Matlab center point code from the 1999 fingerprint identification group on several of our images to evaluate the accuracy of center point algorithms. The center point algorithm, implemented by the 1999 fingerprint group in Matlab, proceeds as follows:

- First a Wiener filter is applied to the fingerprint. This filter performs 2-D adaptive noise-removal by using a pixel-wise adaptive Weiner method. The filter uses neighborhoods of size 5X5 pixels to estimate the local gradient mean and standard deviation. The filter's purpose is to reduce noise that could cause spurious results in the following gradient calculations.
- Next the image is divided into non-overlapping blocks of 10X10 pixels.
- The numerical gradient of the image is determined in the x and y directions, namely $\partial_x$ and $\partial_y$. The values of the gradients are calculated by taking the average of two neighboring pixels.

- A Wiener filter is applied to smooth out the gradients to apply them further.
- Next, for each block, the slope perpendicular to the local orientation of each block is calculated, using the following formula:





$$V_x(i,j) = \sum_{u=0}^{w}\sum_{v=0}^{w} 2\partial_x(u,v) \quad \partial_y(u,v)$$

$$V_y(i,j) = \sum_{u=0}^{w}\sum_{v=0}^{w} [\partial_x^2(u,v) - \partial_y^2(u,v)]$$

$$\theta(i,j) = \frac{1}{2}\tan^{-1}\left(\frac{V_x}{V_y}\right)$$

- For blocks with slopes ranging from 0 to Π/2, we trace down until we reach a slope not ranging from 0 to Π/2. Mark that block.
- The block with the highest number of marks will be used to compute the slope in the negative y direction; it will output an x and y position that will become the center point of the fingerprint.

## C) Sectorization and Normalization

In the sectorization phase, the image is divided into five concentric bands, with each having a radius of 20 pixels that are laid over the center point. The center band has a radius of twelve pixels; hence, the total diameter is (20*5 + 12)*2 – 1 = 223 pixels. Each of the five bands is divided into twelve wedges, and there are therefore a total of 5*12 = 60 total wedges, or sectors for the image.  The center band is ignored because its total area is too small to be any of meaningful use.

Sectorization is important because of the feature extraction component of processing, in which six equi-angular Gabor filters will align with the twelve wedges formed in each of the bands.  Each sector captures information corresponding to each Gabor filter.  The radius values were chosen specifically so as to avoid the effects of circular convolution.

Sectorization is also useful for normalization purposes.  Normalization of the fingerprint is necessary to reduce variations in gray-level values along ridges and valleys throughout the image.  These variations occur when acquiring the image from a scanner, as well as due to the varying pressure applied by the user.  Normalization does not change the clarity of the ridge and valley structures.

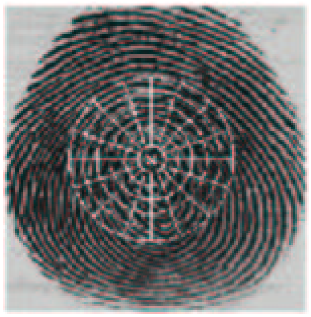
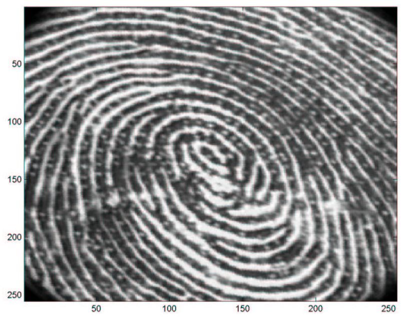


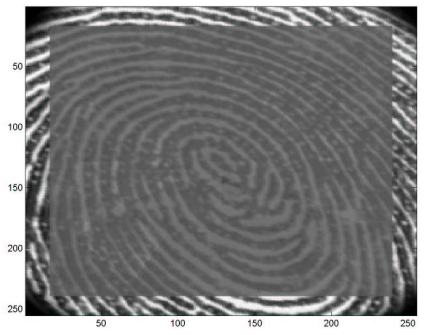
**Figure 1: The sectors**

Each sector is individually normalized to a constant mean and variance of 100, in order to eliminate the variations mentioned above.  All the pixels outside of the sector map are considered to be one giant sector; this will result in an image that is more uniform.  The following formula was used for normalizing each pixel:

$$G(i,j)=\begin{cases} M_0+\sqrt{\dfrac{VAR_0*(I(i,j)-M)^2}{VAR}} & ,If\ I(i,j)>M \\ M_0-\sqrt{\dfrac{VAR_0*(I(i,j)-M)^2}{VAR}} & ,otherwise \end{cases}$$



*Pre-Normalized Print*

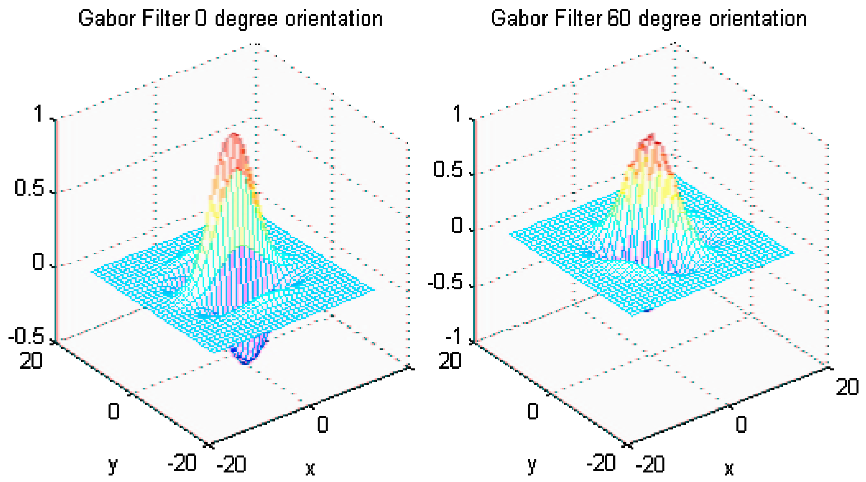


*Same Print Normalized*

## 2.3 Feature Extraction

A) Filtering

The image is next passed through a bank of Gabor filters. Over a local, relatively constant ridge orientation, the frequency and orientation of the ridges and valleys vary slowly. A bandpass filter that is tuned to the correct frequency and orientation can therefore filter out undesired noise, thereby preserving the correct ridges and valleys in that direction [2]. Sectorization can then be used on the filtered images locate ridges in a given sector in a particular direction.



Gabor filters are produced by forming 33x33 filter images for six angles (0, π/6, π/3, π/2, 2π/3 and 5π/6). The filters have an odd height and width to maintain a peak center point.

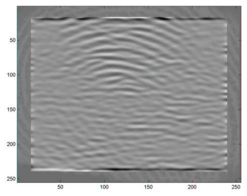
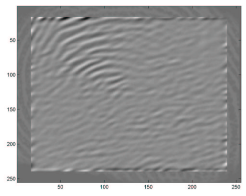
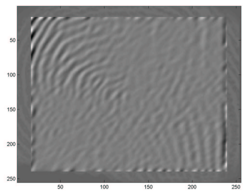
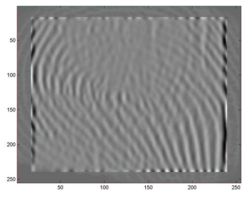
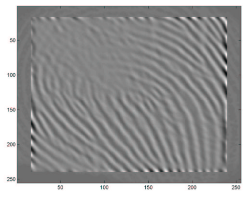
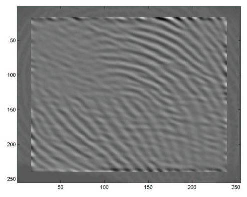
These filters are then convolved with the input image to perform the filtering action. The convolution of the Gabor filters and the image sectors was done spatially.

The Gabor filter used has the general form

$$G(x, y; f, \theta) = \exp\left\{\frac{-1}{2}\left[\frac{x'^2}{\delta_x^2} + \frac{y'^2}{\delta_y^2}\right]\right\}\cos(2\pi f x'),$$

$$x' = x\sin\theta + y\cos\theta$$
$$y' = x\cos\theta - y\sin\theta$$

where θ is the orientation of the Gabor filter, f is the frequency of a sinusoidal plane wave corresponding to the local ridge frequency, and $\delta_x$ and $\delta_y$ are the space constants of the Gaussian envelope along the x and y axes, respectively. The selection of these last two parameters involve a tradeoff between the robustness of the filter to noise, and the creation of spurious ridges and valleys. Larger values result in a more noise-resistant filter while smaller values give a filter less likely to create spurious ridges and valleys. Based on empirical data, these parameters are typically both set to 4.0. The frequency parameter is determined by the inverse of the distance between adjacent ridges in a sector. We determined this value empirically to be 10 pixels, on average. Example output of the Gabor filter bank is the following:

| | | |
|---|---|---|
|  |  |  |
| *0 degrees* | *30 degrees* | *60 degrees* |
|  |  |  |
| *90 degrees* | *120 degrees* | *150 degrees* |

B) Variance Calculation and Feature Vector

After obtaining the six Gabor filtered images, the variance of the pixel values is calculated for each sector using the equation

$$V_{i\theta} = \sqrt{\sum_{K_i} \left(F_{i\theta}(x,y) - P_{i\theta}\right)^2}$$

where $F_{i\theta}$ are the pixel values in the ith sector after passing through a Gabor filter of angle θ, $P_{i\theta}$ is the mean of the pixel values for that sector and angle (after Gabor filtering), and $K_i$ is the number of pixel values in the ith sector. Calculating the variance indicates the approximate concentration of ridges oriented in each direction in the image. If a sector has a high variance at a certain angle it means that a large concentration of ridges are oriented in the direction of the corresponding Gabor filter in that sector. A low variance indicates that the ridges were not moving in that direction and were thus filtered out. The variance results for the 60 sectors in each of the six filter directions form the 360-component (6 angles x 60 sectors) *feature vector* of the fingerprint scan. The feature vector is a way of compactly representing the received fingerprint image.

## 2.4 Classification

The feature vector of the input image is next compared with the feature vectors of the images in the database. We were initially uncertain as to which type of classifier we would use. We considered classification methods such as Mean Nearest Neighbor (MNN), Kth- Nearest Neighbor (KNN) and Mahalanobis Distance, a direction-sensitive distance classifier. The Mean Nearest Neighbor Classifier was ruled out after noting that the mean values for each person's images in the database would be quite divergent from the input image of the same person if even slight rotation or translation of the database images was present, which they were. After collecting data, we determined that the Kth-Nearest Neighbor classifier would be an excellent classification scheme for our system.

A) Kth Nearest Neighbor (KNN) Classifier

In this classifier, the feature vectors of all of the images in the training set are calculated and stored in a database. The feature vector of the input image is then compared to the feature vectors of every image in the database. The Euclidean distance from the input image to each of the database images is calculated and compared to a pre-determined threshold set specifically for the input image. If the Euclidean distance between the input image and a database image is below this threshold, then a counter linked to the person corresponding to that database image is incremented. After this comparison has been performed for every image in the database, the person in the database with the highest counter value is selected as the match.

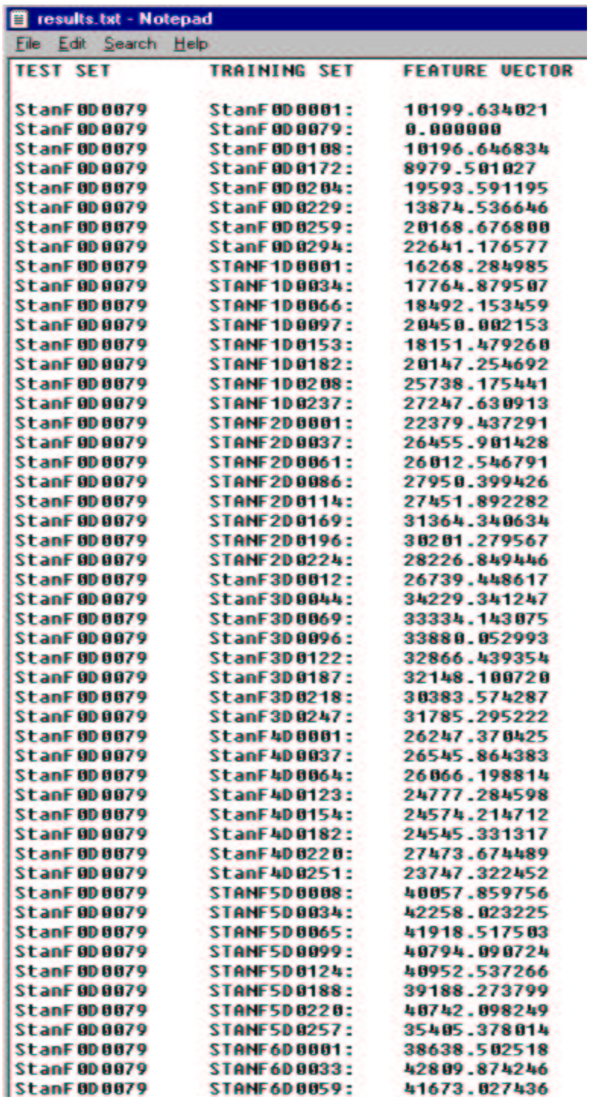
This classifier is useful in that it can take into account variations between different training images from the same person, something that the Mean Nearest Neighbor (MNN) classifier cannot do. Although the KNN classifier requires more computations than the MNN classifier (360 E.D. calculations and comparisons rather than 45) it did not hurt us much in terms of computation time. The KNN classifier takes about five seconds to run, and when compared with the overall system time of about eight minutes, is insignificant.

### B) Threshold Determination

When deciding upon which method to choose for this threshold determination, we had to choose between two different alternatives: the first option was to calculate a general threshold that could be used for all the different users. The disadvantage with this method was that different images generally have different magnitudes of variances (for some images, the magnitude is in the range of 20000.00, while for other images, it is in the range of 45000.00), hence coming up with a simple threshold that would work for all 45 users in the database would be very challenging, if not impossible. The second method was to calculate a different threshold for every user. This method was computationally more demanding (since it involved computing 45 different thresholds), but would lead to more accurate results. Initially, we were undecided as to which method to use. There was a tradeoff to be made between the time spent on computability versus the accuracy of our final results. However, upon consultation with Professor Vijaya Kumar, we decided to choose the second method, due to the higher chance of overall success, and the fact that it is commonly used in industry today.

As mentioned before, we had 8 training set images for each individual user. The threshold values were calculated by using each of these 8 images (one at a time) as a test image, and finding out the feature vector values for the entire training set image using that ‘test image’.

We would output these feature vector calculations to a file, so that they could be analyzed further, and a threshold could be determined.  The screenshot below illustrates the method explained above:



```
TEST SET          TRAINING SET      FEATURE VECTOR

StanF0D0079       StanF0D0001:      10199.634021
StanF0D0079       StanF0D0079:      0.000000
StanF0D0079       StanF0D0108:      10196.646834
StanF0D0079       StanF0D0172:      8979.501027
StanF0D0079       StanF0D0204:      19593.591195
StanF0D0079       StanF0D0229:      13874.536646
StanF0D0079       StanF0D0259:      20168.676800
StanF0D0079       StanF0D0294:      22641.176577
StanF0D0079       STANF1D0001:      16268.284985
StanF0D0079       STANF1D0034:      17764.879507
StanF0D0079       STANF1D0066:      18492.153459
StanF0D0079       STANF1D0097:      20450.002153
StanF0D0079       STANF1D0153:      18151.479260
StanF0D0079       STANF1D0182:      20147.254692
StanF0D0079       STANF1D0208:      25738.175441
StanF0D0079       STANF1D0237:      27247.630913
StanF0D0079       STANF2D0001:      22379.437291
StanF0D0079       STANF2D0037:      26455.901428
StanF0D0079       STANF2D0061:      26012.546791
StanF0D0079       STANF2D0086:      27950.399426
StanF0D0079       STANF2D0114:      27451.892282
StanF0D0079       STANF2D0169:      31364.340634
StanF0D0079       STANF2D0196:      30201.279567
StanF0D0079       STANF2D0224:      28226.849446
StanF0D0079       StanF3D0012:      26739.448617
StanF0D0079       StanF3D0044:      34229.341247
StanF0D0079       StanF3D0069:      33334.143075
StanF0D0079       StanF3D0096:      33880.052993
StanF0D0079       StanF3D0122:      32866.439354
StanF0D0079       StanF3D0187:      32148.100720
StanF0D0079       StanF3D0218:      30383.574287
StanF0D0079       StanF3D0247:      31785.295222
StanF0D0079       StanF4D0001:      26247.370425
StanF0D0079       StanF4D0037:      26545.864383
StanF0D0079       StanF4D0064:      26066.198814
StanF0D0079       StanF4D0123:      24777.284598
StanF0D0079       StanF4D0154:      24574.214712
StanF0D0079       StanF4D0182:      24545.331317
StanF0D0079       StanF4D0220:      27473.674489
StanF0D0079       StanF4D0251:      23747.322452
StanF0D0079       STANF5D0008:      40057.859756
StanF0D0079       STANF5D0034:      42258.023225
StanF0D0079       STANF5D0065:      41918.517503
StanF0D0079       STANF5D0099:      40794.090724
StanF0D0079       STANF5D0124:      40952.537266
StanF0D0079       STANF5D0188:      39188.273799
StanF0D0079       STANF5D0220:      40742.098249
StanF0D0079       STANF5D0257:      35405.378014
StanF0D0079       STANF6D0001:      38638.502518
StanF0D0079       STANF6D0033:      42809.874246
StanF0D0079       STANF6D0059:      41673.027436
```

In the above print screen, the image ‘StanF0D0079’ is one of the 8 images that comprise the training set for the user Stan_F0.  However, in the tests above, we are using ‘StanF0D0079’ as a test image, and have computed the feature vector calculations for all the images in the training set.  The second column consists of all the members of the training set, while the third column consists of the feature vector calculation (between the ‘test’ image and the ‘training’ image).  Note that the feature vector calculation when both the test and training image is

‘StanD0D0079’ is 0, as expected; when both files are the same, the variance between the two images should be 0.

Another observation clearly noticeable from the diagram above is the fact that the feature vector calculation values are so much smaller when there is a match than when two users are falsely put together. It can be seen that the feature vector calculation between the test set (StanF0D0079) and any training set ‘StanF0DXXXX’ has a relatively small number. All the other feature vector calculations are much larger magnitude wise. The threshold for the user Stan_F0 was 14000.00. From the diagram above, it can be seen that 5 of the 8 training sets have values less than 14000.00. The other training set images (for other users) have values that are significantly larger (> 20000.00). This illustrates the clear match between ‘StanF00D0079’ and the user Stan_F0.

Using the method described above, we traversed through our whole database and calculated thresholds for every one of our users. The table on the following page shows the threshold values obtained:

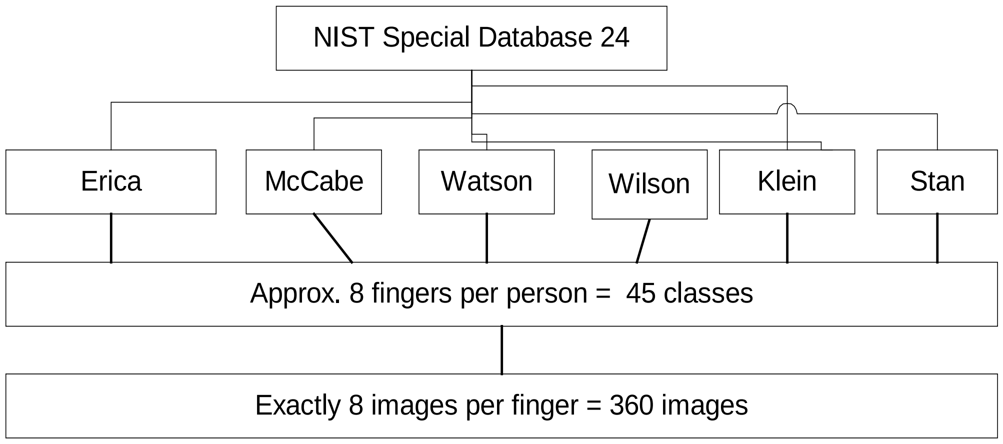
| Class | User Name | Threshold Value | Simulation Name |
|---|---|---|---|
| 0 | Stan_F0 | 14000 | Bain |
| 1 | Stan_F1 | 13000 | Kumar |
| 2 | Stan_F2 | 13000 | Blanton |
| 3 | Stan_F3 | 12000 | Bryant |
| 4 | Stan_F4 | 11400 | Carley |
| 5 | Stan_F5 | 13600 | Casasent |
| 6 | Stan_F6 | 14500 | Cendes |
| 7 | Stan_F7 | 13300 | Charap |
| 8 | Stan_F8 | 14280 | Chen |
| 9 | Stan_F9 | 10900 | Choset |
| 10 | Wat_F0 | 9500 | Davidson |
| 11 | Wat_F1 | 7800 | Faloutsos |
| 12 | Wat_F2 | 8400 | Falsafi |
| 13 | Wat_F3 | 6700 | Fedder |
| 14 | Wat_F4 | 10000 | Fennstra |
| 15 | Wat_F5 | 7800 | Fisher |
| 16 | Wat_F6 | 15000 | Gabriel |
| 17 | Wat_F7 | 9200 | Ganger |
| 18 | Wat_F8 | 11600 | Gibson |
| 19 | Wil_F0 | 14000 | Goldstein |
| 20 | Wil_F1 | 9600 | Greve |
| 21 | Wil_F6 | 12200 | Hoburg |
| 22 | McCabe_F0 | 6850 | Hoe |
| 23 | McCabe_F1 | 9000 | Johnson |

|  |  |  |  |
|---|---|---|---|
| 24 | McCabe_F2 | 9800 | Jordan |
| 25 | McCabe_F3 | 7800 | Kanade |
| 26 | McCabe_F5 | 9200 | Khosla |
| 27 | McCabe_F7 | 10400 | Kim |
| 28 | Eric_F0 | 7900 | Koopman |
| 29 | Eric_F1 | 13200 | Krogh |
| 30 | Eric_F2 | 12900 | Lambeth |
| 31 | Eric_F3 | 10950 | Maly |
| 32 | Eric_F4 | 14200 | Marculescu |
| 33 | Eric_F5 | 11100 | Messner |
| 34 | Eric_F7 | 9700 | Moura |
| 35 | Klein_F0 | 9000 | Mukherjee |
| 36 | Klein_F1 | 17600 | Nagle |
| 37 | Klein_F2 | 8900 | Negi |
| 38 | Klein_F3 | 10000 | Neuman |
| 39 | Klein_F4 | 12050 | Paul |
| 40 | Klein_F5 | 11450 | Peha |
| 41 | Klein_F6 | 9500 | Pileggi |
| 42 | Klein_F7 | 16300 | Rajkumar |
| 43 | Klein_F8 | 18400 | Rutenbar |
| 44 | Klein_F9 | 15500 | Schelsinger |

# 3.0 Implementation

## 3.1 The Data

We used fingerprints obtained from Craig Watson, a research scientist at the National Institute of Standards and Technology (NIST). He gave us images from fingerprint database 24; these images were both pre-centered and pre-normalized. This contained fingerprints from six people: Watson, McCabe, Erika, Wilson, Klein and Stan. For each person, there were fingerprint images available for 8 fingers. For each finger, we used exactly 8 prints. For our purposes, we assumed that each separate finger could constitute a different class (individual), since it is unrelated to the images of other fingers from that person. Under this method, we had a total of 45 different classes in our database. For each user, we had eight distinct training images, and one test image.  To summarize, we had 45 test images, and a training set consisting of 360


NIST Special Database 24
Erica
McCabe
Watson
Wilson
Klein
Stan
Approx. 8 fingers per person =  45 classes
Exactly 8 images per finger = 360 images

images.

Original image size:

700x700 pixels * 4 Bytes Float/value* (1 Real + 1 Imaginary value) = 3.92 MB

Cropped image size:

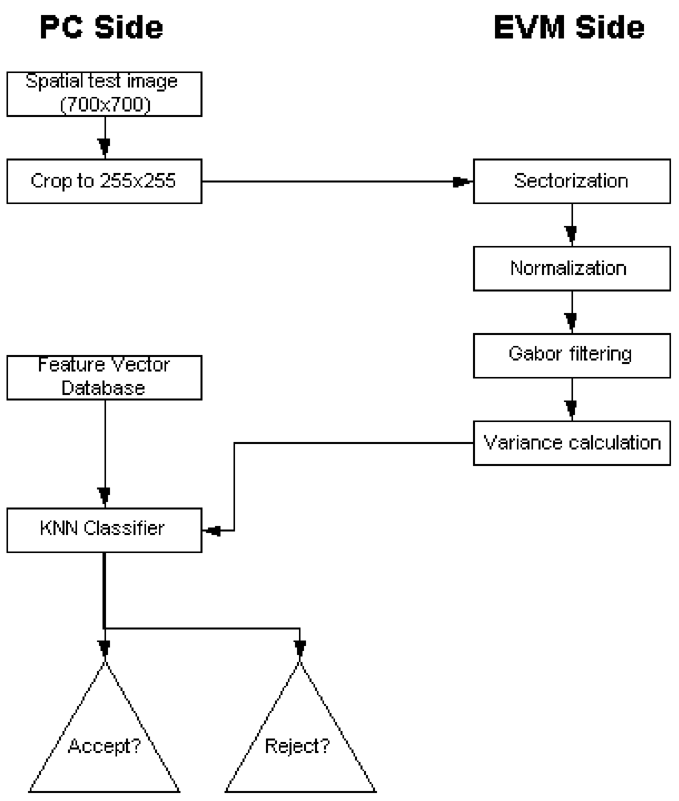
255x255 pixels * 4 Bytes = 255 KB

Total space needed:

255 KB * 405 images = 107.4 MB

## 3.2  Signal Flow (PC vs. EVM)

In the final demonstration, the database has already been created from the training images and the user enters an image in the spatial domain.  Since the images in the original database are already pre-FFT'd, we simulate a live demonstration by running an IFFT on the images outside of the demonstration and then submitting them to the PC and EVM, respectively.  The only processing that is immediately done on the PC is that the image is cropped to 255x255 pixels. The PC then requests a transfer from the EVM, and subsequently sends the cropped 255x255 pixel test image to the C67 board through the Host Port Interface (HPI).  Although we know that the FIFO transfers tend to be faster than the HPI, the HPI transfers have been tested to be more reliable for larger transfers.  On the EVM side, the sectorization, normalization, Gabor filtering, and feature vector generation all takes place.  Next, the PC requests a transfer and receives the feature vector of the test image.  Finally, it takes the feature vector of the test image and compares it against the feature vectors of all the training images, as per the classifier algorithm.



## 3.3 Computation

Approximate calculations:

Gabor kernel generation (computed once):

(33*33 pixels)*(5 multiplies + 3 trigonometric operations) = 52,272 operations

Sector Means (per image):

(223*223 pixels)*(1 add per pixel) + (62 sectors * $\frac{1\, divide}{\sec tor}$) = 49,791 arithmetic operations

Variance (per image):

(223*223 pixels)*(4 adds + 1 mult + 1 square root) = 298,374 arithmetic operations

Normalization (per image):

(223*223 pixels)*(3 mults + 1 sqare root) = 198,916 arithmetic operations

Gabor filtering (total):

(223*223 pixels)* (33*33 pixels in kernel*3 cycles)*6 Gabor angles = 324,929,286 multiplies and adds

Classification (Euclidean distance from training database):

(720 multiplies + 360 adds + 360 square roots)*360 training images = 518,400 arithmetic operations

There are two levels of data processing within this verification system: off-line and on-line. The off-line processes are those that are involved in creating the image database, the feature vector database, and all of the computation that can be done without an actual test print. The only part of the system that is on-line is that which occurs when the test print arrives (sectorization, normalization, gabor filtering, creation of feature vector for a single print, and classification of the print). In a physical implementation of this system, optimization of the off-line processing is unimportant as it occurs before the user's arrival. However, as soon as the user arrives, all of the processing from that point onward is optimized.

As can be seen in the equations above, the largest computational load in this system is created by the Gabor filtering component. 2-dimensional image filtering grows linearly in computation time with the size of the kernel. Unlike the 3x3 kernels used in simple edge detection algorithms, the 33x33 Gabor kernels result in an enormous amount of computation and space requirements. The generation of the kernels themselves is somewhat expensive, but since it is a process that is done off-line, it is not much of a concern (consider a system which generates the Gabor kernels during boot time so that when the user arrives, they are already resident in memory).

## 3.4 Optimizations

A) FFT Algorithm

Initially, this group proposed an alternative method for doing the Gabor filtering which could, in theory, have resulted in significant performance improvements in the current bottleneck. The overall scheme is to save FFT'd versions of the Gabor filters on the hard drive (off-line), and when the test print arrives from the PC, immediately take the FFT and subsequently perform the filtering with the 6 filters and finally perform an IFFT of each image. Ultimately, the memory requirements for this scheme were prohibitively large and when we tried to allocate memory to complete it, the EVM would consistently return with incomprehensible errors. Even though we were able to easily allocate the same amount of space and perform the optimization on the PC side, the board would hiccup with each allocation of 512KB, thus rendering the board-size FFT impossible within our time constraints. Even though the TA's made every attempt to assist the debugging process, the errors were not even from the compiler level, but rather from the linker and the assembler, so in the final stages we reverted the design to a spatial-domain filtering scheme. Nonetheless, it is debatable whether the transform-domain would have actually been computationally cheaper since 6 IFFT's would have been necessary, and those come at a cost.

B) Memory Paging

In most cases, it is very beneficial to copy small amounts of data from the SDRAM to the SBSRAM or the on-chip memory before actually doing the processing. In previous experiments in this class (using a 3x3 filter kernel), it was ideal to page 16 KB at a time into the board's on-chip memory for processing, but in this case, the smallest "chunk" would be a 33x33x255 slice, which takes up 278,784 bytes (more than 256 KB). Since the on-chip memory contains 128 KB total and the SBSRAM contains 256 KB of memory total, the only space on the board where the processing could be done is on the SDRAM, which takes 15 cycles per access as compared to the 1 cycle per access for on-chip memory. When we tried to break it up and place it onto smaller, faster portions of the memory hierarchy, the same errors would result that occurred during the FFT experimentation.

C) Code Size

Due to the limited space in on-chip memory, it was necessary to allocate the .text portion of the program in the SBSRAM. This is a standard practice and is purely a function of code size, which we attempted to minimize by using the highest level of optimization in the compiler (linker option –o3) and by removing all of the extraneous code from the final release code (mostly debug statements).

D) Loop Unrolling

There is one place in the code that easily lends itself to loop unrolling, and that is the gabor filter kernel generation. The reason why it is convenient in this portion is that there are two levels of nested loops, 33 computations wide each. However, after unrolling the bottom for loop so that all 33 additions and multiplications were done in one statement, there was no performance improvement (in rote time taken for computation). It is quite likely that the optimizing compiler already optimized that loop for maximum performance. In the Gabor 2-dimensional spatial filtering, everything is nested loops but due to the size of the kernel (1089 elements), the memory requirements nullify any chance of unrolling.

E) Profiling

When our group attempted to run the profiler in Code Composer Studio the program would slow to a crawl and eventually crash (possibly due to too long a wait). However, we selected the most computationally expensive portion of the algorithm and hand-profiled it in the ideal case.

$$(255*255\text{ pixels})*(33*33\text{ pixels }*\frac{3\,cycles}{(6\,mults\ and\ adds)})*6\text{ angles}*\frac{31\,cycles}{access} = 6{,}585{,}536{,}925\text{ cycles}$$

(A) (B) (C) (D) (E)

Explanation of the various parts of the above equation:

(A) The number of pixels in the total image

(B) The number of pixels in the kernel

(C) 2 cycles for a multiplication, 1 cycle for an add, 6 multiplies and adds in parallel

(D) The number of Gabor angles

(E) 31 cycles to access the input print pixel (from SDRAM), 15 cycles to access the output print pixel, and 1 cycle to access the Gabor kernel value (off of on-chip memory).

Since the chip can be approximated to run at 100 MHz, or 100 million cycles per second, then it should do the Gabor filtering above 65.8 seconds as computed below:

$$6{,}585{,}536{,}925\text{ cycles }*\frac{100\,million\ cycles}{\sec ond} = 65.8\text{ seconds}$$

This is still not as extreme as the 4 minutes that the system actually takes, but there are certainly non-idealities that are introduced in the compiling process and in other parts of the algorithm that have not been accounted for in that calculation. It is provided mostly to suggest the order of the computational needs.

## 3.5 Code used from previous projects

Most of the final implementation from 1999 was not useful because they were using another DSP board from the one being used these days in 18-551. As for the 2001 project, the code left behind was not integrated and did not take the same approach, so we ended up reworking most of it for our database (a different database), and writing our own code from scratch for data analysis and classification. As for the Texas Instruments web page, we could not find any of the algorithms which were implemented in this project. There was one piece of code from lab 2 that we attempted to integrate (the in-place 1-D FFT) during our foray into the transform domain, but it was never implemented in the final project due to the nightmares of memory allocation (see Implementation section).

Gabor filtering code: In 2001, they did this in the spatial domain as well, so we were able to use the code. However, their work was actually implemented on the PC Side so we ported it over to the EVM. There is no web link to this code, but it can be obtained from Professor Casasent.

Sectorization: There were array-indexing errors in the sectorization code used from 1999 and 2001 (which could have affected their results), so we repaired and used it. The web link for this code is:

http://www.ece.cmu.edu/~ee551/Old_projects/projects/s99_19/fp/sectnorm.c

## 3.6 Code Documentation

| | |
|---|---|
| readiles.m | Takes a 700X700 FFT'ed image and converts it to a 255X255 IFFT'ed image |
| readgabs.m | Displays the Gabor images that are created on the EVM side |
| gabor_pc.c | PC side of the first part of our demo (normalization, sectorization, gabor filtering, variance and feature vector calculation). This file is run on Microsoft Visual C++ |
| gabor_evm.c | EVM side of the first part of our demo. The EVM does the normalization, sectorization, gabor filtering, variance and feature vector calculation. This file runs on Visual Studio, with gabor_pc.c running concurrently on Microsoft Visual C++ |
| working_demo.c | The second part of our demo, that is all PC based. This is where the verification of the user takes place |
| pc_side.c | Our whole demo, all done on the PC side. This was our preliminary file, before starting to shift some of the processes onto the EVM. It is also much quicker than our actual demo, since the EVM is pretty slow |

## 4.0 Results

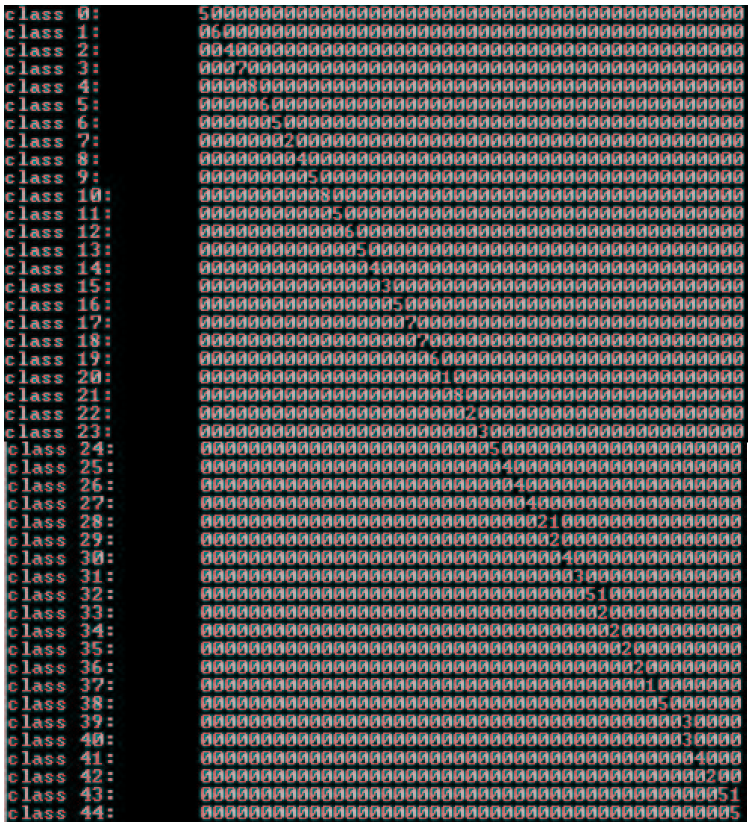
Overall, our algorithm worked splendidly, as our 100% success rate demonstrates. A major reason we had such a high success rate was due to the fact that our images were both pre-normalized and pre-centered. Generally in the past, trying to find an adequate centering algorithm has proved to spell the downfall of many groups; there simply isn't any algorithm out there right now that reliably calculates the center point of an image. Hence, the fact that this step had already been done for us made our task a little more straightforward.

We had a total of 45 test set images, with a training set database of 360 images (8 per user). Our algorithm was able to successfully verify that each user was who he/she claimed to be. We tested our algorithm very vigorously, and it passed all the tests successfully.

.

## 4.1 Confusion Matrix

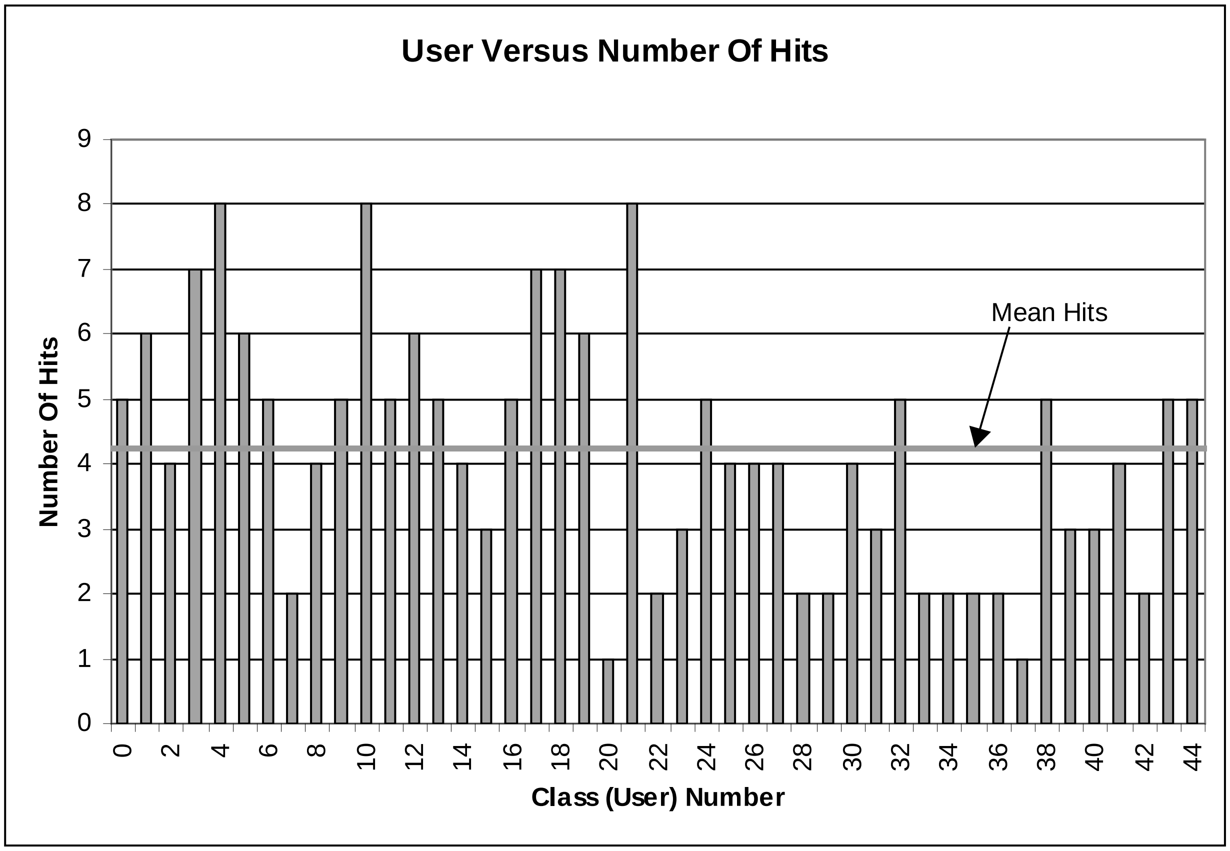
The following is a screenshot of our confusion matrix. ‘Class’ represents the user (we had a total of 45 users for our database), whilst the corresponding array of numbers represents the number of training set images for each particular class (for all the 45 classes) that have Euclidean distance calculations that are below the threshold. As an example, for Class ‘0’, the 0th index (which corresponds to user ‘0’) has 5 training set images that match to class ‘0’. No other user has any training set images that match to class ‘0’. This intuitively makes sense, since only user ‘0’ should have any images that match with class ‘0’. This was generally the case for all the classes, demonstrating that the verification system does indeed work with unerring accuracy.



```
class 0:        500000000000000000000000000000000000000000000
class 1:        060000000000000000000000000000000000000000000
class 2:        004000000000000000000000000000000000000000000
class 3:        000700000000000000000000000000000000000000000
class 4:        000080000000000000000000000000000000000000000
class 5:        000006000000000000000000000000000000000000000
class 6:        000000500000000000000000000000000000000000000
class 7:        000000020000000000000000000000000000000000000
class 8:        000000004000000000000000000000000000000000000
class 9:        000000000500000000000000000000000000000000000
class 10:       000000000080000000000000000000000000000000000
class 11:       000000000005000000000000000000000000000000000
class 12:       000000000000600000000000000000000000000000000
class 13:       000000000000050000000000000000000000000000000
class 14:       000000000000004000000000000000000000000000000
class 15:       000000000000000300000000000000000000000000000
class 16:       000000000000000050000000000000000000000000000
class 17:       000000000000000007000000000000000000000000000
class 18:       000000000000000000700000000000000000000000000
class 19:       000000000000000000060000000000000000000000000
class 20:       000000000000000000001000000000000000000000000
class 21:       000000000000000000000800000000000000000000000
class 22:       000000000000000000000020000000000000000000000
class 23:       000000000000000000000003000000000000000000000
class 24:       000000000000000000000000500000000000000000000
class 25:       000000000000000000000000040000000000000000000
class 26:       000000000000000000000000004000000000000000000
class 27:       000000000000000000000000000400000000000000000
class 28:       000000000000000000000000000021000000000000000
class 29:       000000000000000000000000000002000000000000000
class 30:       000000000000000000000000000000400000000000000
class 31:       000000000000000000000000000000030000000000000
class 32:       000000000000000000000000000000005100000000000
class 33:       000000000000000000000000000000000200000000000
class 34:       000000000000000000000000000000000020000000000
class 35:       000000000000000000000000000000000002000000000
class 36:       000000000000000000000000000000000000200000000
class 37:       000000000000000000000000000000000000010000000
class 38:       000000000000000000000000000000000000005000000
class 39:       000000000000000000000000000000000000000030000
class 40:       000000000000000000000000000000000000000030000
class 41:       000000000000000000000000000000000000000004000
class 42:       000000000000000000000000000000000000000000200
class 43:       000000000000000000000000000000000000000000051
class 44:       000000000000000000000000000000000000000000005
```

The following graph provides a simpler interpretation of the results seen in the confusion matrix:

An interesting statistic is that the average number of hits per user was around 4.2.  This means that for every user, approximately 4 of the 8 training images had feature vector values that were below the threshold value (hence a hit was recorded for each one of these training set images).



The higher the average number of hits, the more reliable the results, due to a higher number of training set images agreeing with the verification.

## 4.2  Center Point Determination

As mentioned in section 2.2 (B), we attempted to run the Center Point Determination algorithm, written by the 1999 Fingerprint Identification group in Matlab, on several of our images.  Some editing of the Matlab file was needed initially in order to read in the file to the appropriate format and set the image dimensions correctly.  Running the algorithm on a few of our images gave bogus results, an indication that the center point algorithm does not work well, as predicted by Professor Casasent.  Since the images in our database were already centered we did not have to worry about trying to implement our own centering algorithm, and we achieved excellent results without one.

## 4.3 Shift Invariance

As previously mentioned, the images in our database came to us pre-normalized and pre-centered. In order to test the robustness of our verification system we ran tests to determine how resilient the system was to shifts in the input image. A shift in the input image would cause sectors no longer to overlap properly and would slightly alter variance calculations, thereby changing the Euclidean distances calculated, and possibly effecting a proper choice or reject decision. We ran our algorithm on input images shifted by 2, 6 and 10 pixels. We tried this with five different prints and in all cases the print was not properly verified. From this testing, it appears that our system is very sensitive to shifts in the input image and strongly relies upon centered images for successful verification. The primary contributing factor in this occurrence was probably the thresholds set for each person. The thresholds were determined given the current alignment of the images and when that alignment is changed the Euclidean distances will change, and a new, more accurate threshold must be set.

## 5.0 Further Work

Considering that the greatest deficiency in the current implementation of this project is the speed, that would be the first priority of any improvements. Even if simply the paging to memory worked, it would result in approximately 10x improvement in the Gabor portion, which would reduce the runtime of a single experiment from six minutes to under one minute. After that, we would attempt to do DMA transfers from the board to the on-chip memory, using a threaded scheme so that there is always data being processed and being relocated.

## 6.0 Acknowledgements

We gratefully acknowledge all of the assistance provided by Professors David Casasent and Vijaya Kumar, both of whom have offered all of their knowledge about fingerprinting and pattern recognition to a group that initially knew nothing about either. In addition, the previous groups who have worked on this topic for 18-551 projects (in 1999 and 2001) provided a good reference point and a base of information from which to build. Craig Watson's database of images was also extremely helpful in fulfilling our goals of a robust, accurate system. Finally, we'd like to thank the Teaching Assistants of the class for their help throughout the semester.

## 7.0 References

[1] L. Ern, G. Sulong. "Fingerprint Classification Approaches: An Overview," International Symposium on Signal Processing and its Applications, pp. 347-349, 2001.

[2] L. Hong, Y. Wan, A. Jain. "Fingerprint Image Enhancement: Algorithm and Performance Evaluation." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 777-789, 1998.

[3] M. Adhiwiyogo, S. Chong, J. Huang, W. Teo. "Fingerprint Recognition." 18-551 Final Project, Spring 1999.

[4] "Fingerprint Recognition." 18-551 Final Project, Spring 2001.

[5] C. Dorai, N. Ratha, R. Bolle. "Reliable Distortion Detection in Compressed Fingerprint Videos." IBM T.J. Watson Research Center. November 2, 1999.

[6] A. Jain and A. Ross. "An Overview of Biometrics." Michigan State University Pattern Recognition and Image Processing Lab.

[7] National Institute of Standards and Technologies. NIST Special Database 4: NIST 8-Bit Gray Scale Images of Fingerprint Image Groups (FIGS).
http://www.nist.gov/srd/nistsd4.htm.

[8] R. McDowall. "Biometrics: The Password You'll Never Forget." McDowall Consulting, Kent, UK, 2000.
http://www.21cfr11.com/files/library/compliance/lcgc10_00.pdf

[9] "1999 Glossary for Biometric Terms." Association for Biometrics (AfB) and International Computer Security Association (ICSA).
http://www.afb.org.uk/downloads/glossuk1.html

[10] A. Jain, S. Pankanti, S. Prabhakar, A. Ross. "Recent Advances in Fingerprint Verification."
http://citeseer.nj.nec.com/455299.html

[11] A. Jain, S. Pankanti. "Automated Fingerprint Identification and Imaging Systems." Dept. of Comp. Science & Eng, Michigan State University, and TJW Research Center, IBM Research.
http://citeseer.nj.nec.com/453622.html

[12] S. Greenberg, M. Aladjem, D. Kogan, I. Dimitrov. "Fingerprint Image Enhancement Using Filtering Techniques." Electrical and Computer Engineering Department, Ben-Gurion University of the Negev.
http://www.ee.bgu.ac.il/~aladjem/pdf/C27_icpr2000.pdf