Final Report 18-551 (Spring 1999)
Fingerprint Recognition
Group Number 19
(Markus Adhiwiyogo, Samuel Chong, Joseph Huang, Weechoon Teo)

Table of Contents

- Introduction
- Problem Description
- Design
  - Pre-Processing
  - Feature Extraction
  - Classification
- Implementation
- Results
- Conclusion
- Code
- References

I. Introduction

In today's society, the use of electronic commerce, transaction of monetary assets and daily use of email is rapidly increasing. Along with the increased ease of purchasing and selling, there is also an increase in fraud mostly from false identification. Solutions to this problem has been in the field of biometrics, using the person's body as a form of identification. In particular, the uniqueness of fingerprints have made them very popular among law enforcement, banking establishments, and commerce.

The purpose of this project was to design a fingerprint recognition system capable of discerning the identity of fingerprints and able to reject fingerprints for rescanning. The goal was to implement the system in hardware with a DSP applications specific chip. This report describes an implementation of a fingerprint recognition system and contains an analysis of the results.

II. Problem Description

Fingerprints are commonly classified as 5 different types: whorl, left loop, right loop, arch, and tented arch. For most recognition systems, difficulties arise in distinguishing between fingerprints of the same type. Many recognition systems find minutiaes, the ridge stops or breaks, in a fingerprint to identify a fingerprint. Because the number of minutiaes vary among fingerprints, this lends to a variable length of features extracted [1]. This results in difficulties in devising a classification system and database storage. The system we have implemented yields a fixed length of features from which many classification schemes can be applied.

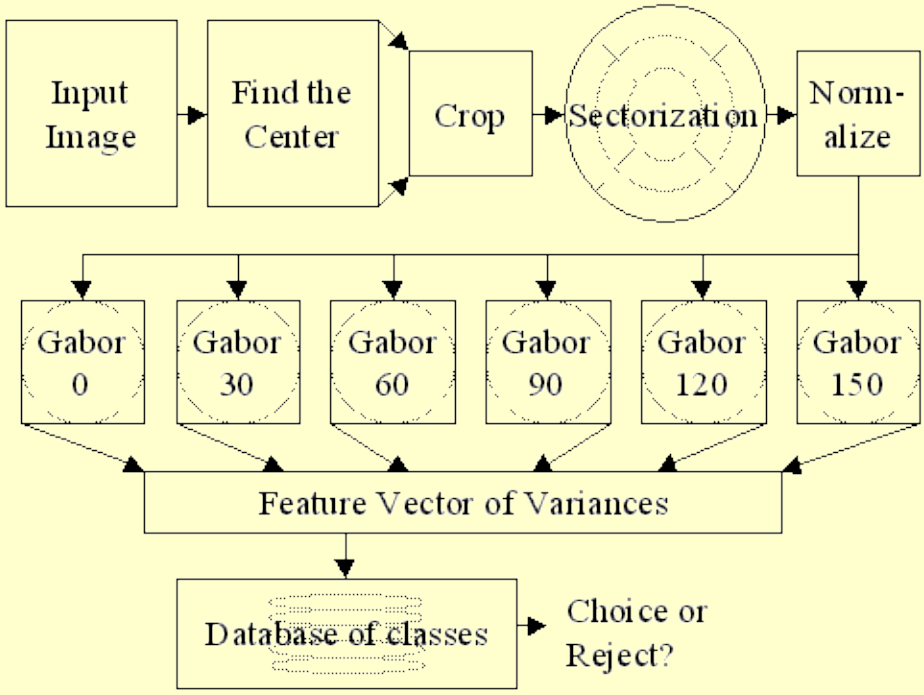
For both our system and the minutiae-based systems, the center point of the fingerprint is important to find for reference in obtaining features. The algorithm for center point determination should be consistent and the system must be "shift-tolerable".

Most of the difficulties for fingerprint recognition systems is in the obtaining of the fingerprint image. Plastic distortion, scanning artifacts, scanning resolution, rotation, and uneven pressure are examples of issues that a system has to resolve in order to maintain consistent results.

Also, fingerprint recognition is a type of image processing which in itself requires memory and computational power. For reasonable performance in speed, a recognition system has to take into consideration these parameters as well.

III. Design

As was stated previously, this system will be able to identify the owner of a fingerprint with reasonable accuracy and will have the ability to reject a fingerprint when the system is "unsure" of it's results. The system presented has been divided into three stages: preprocessing of a fingerprint image, extraction of the features that represent the fingerprint, and the classification of the fingerprint for a decision or a rejection.



The figure above shows our system flow design.

A. Preprocessing

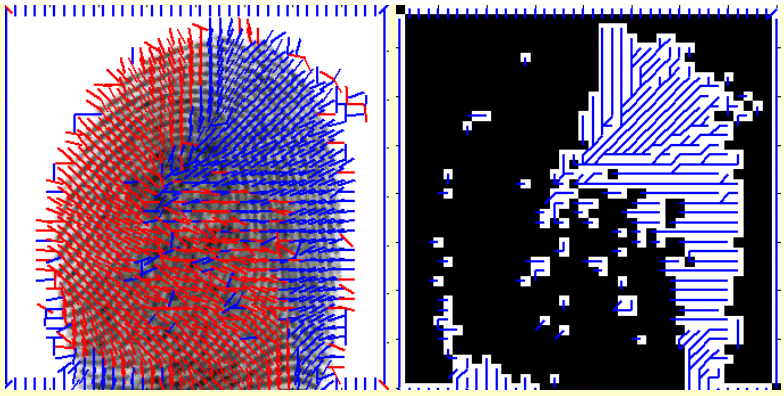
The fingerprint image to be processed needs a resolution of 500dpi so that will at least 10 pixels are between ridges. This is necessary for the feature extraction stage of our design. Because the center point of a fingerprint varies widely amongst individuals, the whole fingerprint must be obtained. We took a 400x400 grayscale fingerprint image for processing which is enough to capture the whole fingerprint.

1. Center Point Determination & Cropping

The first step is to find the center point of a fingerprint. Centerpoint location is done to find the point of most curvature by determining the normals of each fingerprint ridge, and then following them inwards towards the center. The following is the procedure we used [2].

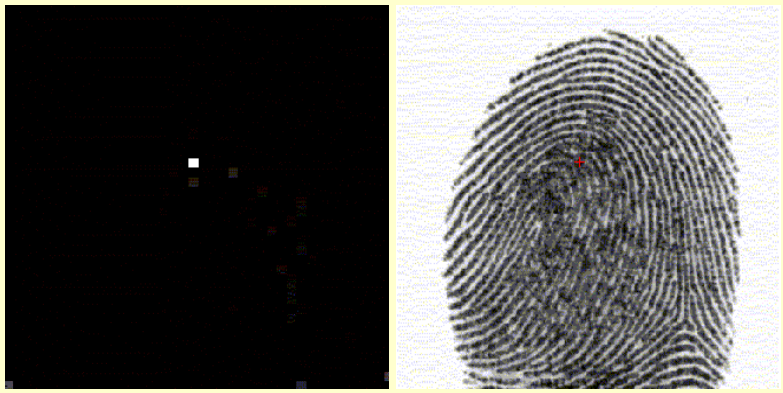
- Apply a pixel-wise adaptive 2-D Gaussian lowpass Wiener filter to the fingerprint. The filter uses neighborhoods of size 5 by 5 to estimate the local gradient mean and standard deviation. This will help in reducing any noise that may cause spurious results in the following gradient calculations.
- Divide the input fingerprint image into non-overlapping blocks of size 10x10.
- Determine the x and y magnitudes of the gradient at each pixel in each block, G_x and G_y. This is done by taking the average of the two neighboring pixels.
- Apply the same 2-D Gaussian lowpass filter on the x and y gradients as above to smooth out the gradients.
- With each block, compute the slope perpendicular to the local orientation of each block using the following formula.

$$\Theta = \frac{1}{2}\tan^{-1}\left(\frac{\sum_{i=1}^{10}\sum_{j=1}^{10} 2G_x(i,j)G_y(i,j)}{\sum_{i=1}^{10}\sum_{j=1}^{10}\left(G_x^2(i,j)-G_y^2(i,j)\right)}\right)+\frac{\pi}{2}$$



- Only looking at blocks with slopes with values ranging from 0 to $\pi/2$, trace a path down until you encounter a slope that is not ranging from 0 to $\pi/2$ and mark that block.

- The block that has the highest number of marks will compute the slope in the negative y direction and output an x and y position which will be the center point of the fingerprint.



Once this point is determined, we then move it down by 30 pixels because the next step of pre-processing ignores any pixels within a 12 pixel radius of the psuedo-center. If we did not move the center point, that part of the image which we feel has the most information would be lost. The image is then cropped to a 255x255 image centered around this pseudo-center point. This particular size is for radix-2 FFT operations for the feature extraction procedure where we'll pad an extra row and column of zeros. The size has an odd height and width such that the next step of sectorization will have a center pixel.

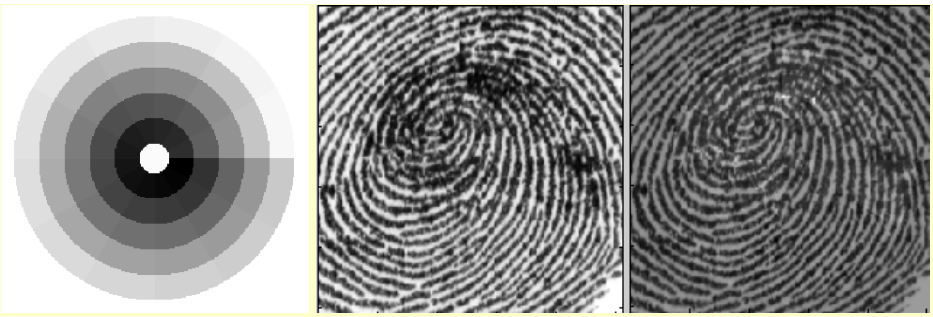
Notice that this algorithm does not determine the orientation of the whole fingerprint. We are assuming that all input images will have the same orientation before processing.

2. Sectorization and Normalization

The cropped fingerprint image is divided into 5 concentric bands centered around the pseudo-center point. Each of these bands has a radius of 20 pixels, and a center hole radius of 12 pixels. Thus, the total radius of the sectorization is 223 pixels. Each band is evenly divided into 12 sectors. The center band is ignored.

This process of sectorization is done because of the feature extraction section. 6 equi-angular Gabor filters will be used which will align with the 12 wedges formed by the bands. In other words, each sector will capture information corresponding to each Gabor filter. The center band is ignored because it has too small an area to be of any use.

The radius of the sectorization was chosen to avoid the effects of circular convolution in applying a Gabor filter. This will be explained in more detail in the next section. All in all, we are left with a total of 60 sectors (12 wedges × 5 bands).



The three figures above show the 60 different sectors, the original fingerprint and the normalized fingerprint respectively.

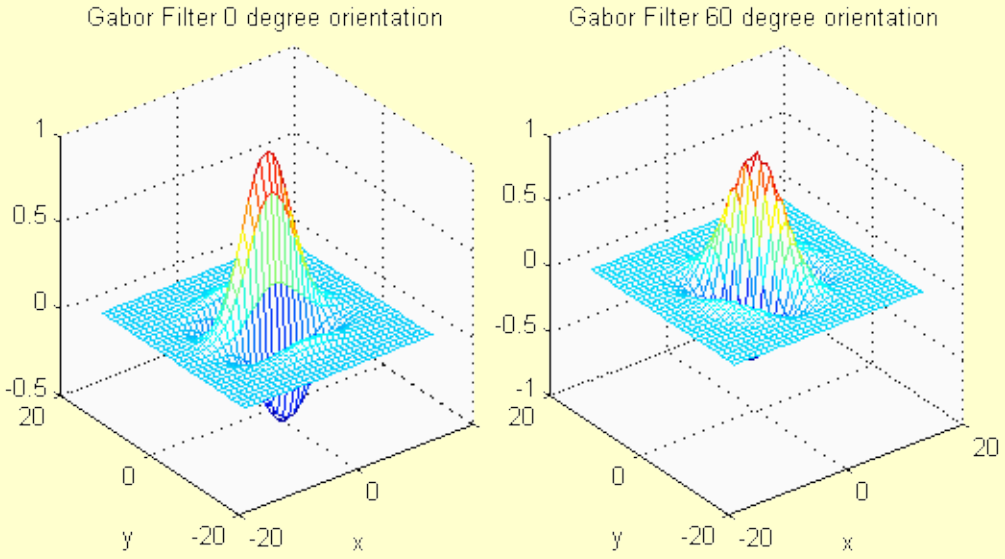
Another reason for sectorization is for normalization purposes. Each sector is individually normalized to a constant mean and variance to eliminate variations in darkness in the fingerprint pattern, due to scanning noise and pressure variations. And all the pixels outside of the sectormap is considered to be one giant sector. This will yield in an image that is more uniform. The following is the equation which we used for normalization of each pixel. We used a constant mean M_0 and variance V_0 of 100. i is the sector number, M_i is the mean of the sector, and V_i is the variance of the sector.

$$N_i(x,y) = \begin{cases} M_0 + \sqrt{\dfrac{V_0 \times (I(x,y) - M_i^2)}{V_i}}, & if\ I(x,y) > M_i \\ M_0 - \sqrt{\dfrac{V_0 \times (I(x,y) - M_i^2)}{V_i}}, & otherwise \end{cases}$$

B. Feature Extraction

1. Gabor Filterization

We then pass the normalized image through a bank of Gabor filters. Each filter is performed by producing a 33x33 filter image for 6 angles (0, π/6, π/3, π/2, 2π/3 and 5π/6), and convolving it with the fingerprint image. Spatial domain convolving is rather slow, so multiplication in the frequency domain is done; however, this involves more memory to store real and imaginary coefficients.



The purpose of applying Gabor filters is to remove noise while preserving ridge structures and providing information contained in a particular direction in the image. The sectorization will then detect the presence of ridges in that direction.

As was discussed before, the sectorization size of 223 was derived to avoid the effects of circular convolution. Because the size of each Gabor filter is 33x33, we only have to worry about 16 pixels aliasing over each side of the image. The Gabor filter also has an odd height and width to maintain its peak center.

point. The following is the definition of the Gabor filter we are using [3].

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

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

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

The parameters, δ_x and δ_y were empirically determined and were both set to 4.0. Too small a value will make the filter ineffective in removing noise while too large a value will destroy ridge and furrow details. The frequency of the cosine envelope is determined by the inverse of the distance of two ridges. We found that the distance to be on average 10 pixels.

2. Sectorization and Variance Calculation

After we get the 6 filtered images, we calculate the variance of the pixel values in each sector. This will tell us the concentration of fingerprint ridges going in each direction in that part of the fingerprint. A higher variance in a sector means that the ridges in that image were going in the same direction as is the Gabor filter. A low variance indicates that the ridges were not, so the filtering smoothed them out. The resulting 360 variance values (6 × 60) as the feature vector of the fingerprint scan.

The following is the equation for variance calculation. $F_{i\theta}$ are the pixel values in the *i*th sector after a Gabor filter with angle θ has been applied. $P_{i\theta}$ is the mean of the pixel values. K_i is the number of pixels in the *i*th sector.

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

C. Classification

The feature vector received from feature extraction is a compact representation of the fingerprint image we received. The next step is to classify the fingerprint to an owner or reject the fingerprint for rescanning.

We have thought of various classification methods such as finding the Maximum Likelihood using Gaussian pdf approximation, Kth-Nearest Neighbor and Mean Nearest Neighbor (MNN). We finally decided to use the MNN method after considering the memory and computation issues.

1. Mean Nearest Neighbor

In this classifier, the mean of all of the feature vectors from each class (each individual person) is calculated from the training set and then saved in our database for future matching. When given a fingerpint, we find the Euclidean distance between the target feature vector and each class in our database, and choose the class that yields the lowest distance as the match.

Compared to the other classification schemes we looked into, the mean nearest neighbor required the least number of calculations to yield a result. It also required the least storage since it is a sufficient statistic of the training set (only one feature vector for each class needed). This classifier does not take into consideration the variations within the same class of fingerprints. But after some tests we ran, which will be commented on later, the mean nearest neighbor classifier is sufficient enough for this project.

2. Threshold Determination

A fingerprint may be rejected due to two reasons. First, the fingerprint may not be in the database and thus the classifier should not classify it to any of the classes in the database. Second, the fingerprint may be in the database but its quality is so bad that the classifier is not able to confidently classify it to the right class. In this case, it is better for the classifier to reject the fingerprint and request another print. By "bad" qulity, we refer to prints that have too much distortion in it or their center points were determined incorrectly. To do this, we

have two threshold values that both must fail in order for a rejection to occur.

One check is if the minimum distance (best match) is too high. We will reject this fingerprint as not being in our database, since the feature is not close enough to be anything. The threshold value we use is determined by eyeballing some actual values received from our training set.

The second check is the difference in distance between the best match and the second best. If this value is not high enough (i.e. the best match does not stick out enough from the rest), we are better off not choosing anything. As can be seen, tuning of these thresholds should depend on exactly how large a variation in Euclidean distance can occur between scans from the same fingerprint.

By studying the distances of our training set we obtained the following thresholds for rejection. If the euclidean distance for the best match is above 330,000 AND if the difference between the best match and the next best match is below 50,000, we will reject the fingerprint.

IV. Implementation

A. Fingerprint Database Collection

For this project, we gathered from 31 individuals, 35 fingerprints each by using an inkpad and paper. We asked each person to press their right thumb on the paper without rolling the finger. The reason why we did not roll the finger was because of a warning from Professor Casasent telling us that rolled prints have plastic distortion. Upon examination of a few test prints, we were pleased with the resulting inkprints.

We then scanned each fingerprint taking into consideration the orientation so that our system will not have to deal with rotation. Each image was saved as a 400x400 pixel 8-bit grayscale image scanned at 500dpi. A record of what types of fingerprints we collected is shown in the table below. The reject class is an extra set of fingerprints that was collected to train and test the system's ability to reject a fingerprint that is not part of the database.

class0 - Whorl	class10 - Twin Loop (Whorl)	class20 - Arch
class1 - Left Loop	class11 - Whorl	class21 - Whorl
class2 - Whorl	class12 - Whorl	class22 - Whorl
class3 - Whorl	class13 - Whorl	class23 - Left Loop
class4 - Whorl	class14 - Whorl	class24 - Whorl
class5 - Twin Loop (Whorl)	class15 - Left Loop	class25 - Right Loop
class6 - Whorl	class16 - Right Loop	class26 - Left Loop
class7 - Whorl	class17 - Left Loop	class27 - Left Loop
class8 - Left Loop	class18 - Left Loop	class28 - Whorl
class9 - Whorl	class19 - Whorl	class29 - Left Loop
reject - Left Loop		

The set of 35 fingerprints from each class is partitioned into a training set and testing set. The training set consists of 30 prints that is further divided into two sets of 25 prints and 5 prints. The 25 prints are used to form the database and the other 5 prints are used to determine the threshold for rejection and tuning of the algorithm's parameters. We also used the 30 fingerprints from the reject class to help determine the threshold to use. Based on the output from our matcher, we noticed that a fingerprint is usually misclassified when the fingerprint is too far away from best match and the distance between the best and next best match are too close. By studying the output of the matcher based on the 30 training fingerprints, we obtained the thresholds for rejection.

The remaining 5 unused prints are then used to compile a confusion matrix to determine the performance of the system. The matrix is shown in the results section.

B. Hardware

The hardware we used for this recognition system was a PC and an Analog Devices Sharc (Super Harvard Architecture Computer) DSP 21062 EZ-LAB Development board [7] developed by Bittware, Inc. We also acquired the use of scanners that were publicly available as well as the traditional inkpad and paper for printing.

Memory is also an important component in our design because of the intensive image processing. The following is the breakdown of major memory usage needed for the whole algorithm.

Size	Words	Purpose
2×256×256	= 131072	Padded Fingerprint(Real & Imag)
2×256×256	= 131072	Padded Gabor Filter(Real & Imag)
6×60	= 360	Features
Total	262504 floats × 4 bytes each ~ 1 MB	

We attempted to purchase a CHUM 16Meg Memory Expansion card from Bittware, Inc. but later found out that our development board was not compatible with it [8]. In interest of time and monetary funds, we fell back to using the PC for memory storage and have the Sharc board do all the computations. In doing so, speed was sacrificed.

The advantage in using the SHARC development board over the TI C30 EVM boards is in the amount of memory on-chip (64K words) and in the user-friendliness of documentation and software compilers. The learning curve in using the SHARC is very steep.

C. Software

The C compiler libraries for the Sharc board that we have were compiled with a 16-bit instruction set. Therefore, we hit the 640K conventional memory limit very quickly and the usage of XMS (Extended memory) is needed. This further slows down memory access due to memory page swapping. The major bottleneck in this system is the memory swapping required between the Sharc and the PC.

Borland C++ v5.0 was obtained for a C compiler that is compatible with 16-bit DOS compiling and has libraries for XMS memory allocation. The Sharc board comes with its own compiler and linker. C code was written and compiled from scratch except for the code for XMS which was obtained from Borland's web site [9].

Due to memory constraints on the SHARC board, not all of the code are loaded all at once. The algorithm is split into 4 parts: Sectorization and Normalization, Gabor Filter Generation, FFT Algorithms, and Convolution with FFT Shifting. Each of these parts are loaded onto the Sharc board when needed. The center point determination is done in MATLAB and the classification is done in C on the PC.

Not only is this algorithm implemented on the Sharc board, it is also implemented in MATLAB and in C++. Code can be obtained below.

V. Results

The time to extract the features of one fingerprint takes on average 4 minutes 30 seconds on a Pentium 166 non-MMX with 128Megs of EDO RAM. The following confusion matrix was obtained on an equivalent system developed purely in C on a Linux machine without the use of the Sharc board. For that system, it takes less than 2 seconds to extract the features. For the equivalent system in MATLAB, it takes 40 seconds.

Summary of Results

With thresholds, there are 12 rejected and 7 misclassified fingerprints out of a total of 155 prints. Thus we achieved an accuracy of 95.1% without including those rejected fingerprints.

Without thresholds, there are 13 misclassified fingerprints out of a total of 155 prints. Thus we will only achieve an accuracy of 91.6% if the system doesn't allow for rejection of fingerprints.

Confusion matrix - Best Match

| Classified as Class

class#											1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	R
	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	J
class0	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
class1	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
class2	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
class3	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1
class4	-	-	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
class5	-	-	-	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
class6	-	-	-	-	-	-	1	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	3
class7	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1
class8	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
class9	-	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
class10	-	-	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
class11	-	-	-	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
class12	-	-	-	-	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
class13	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1
class14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1
class15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
class16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-
class17	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	-
class18	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-
class19	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-	-	-	-	-	-
class20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-	-	-	-	-
class21	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	3
class22	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-	-	-
class23	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-	-
class24	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-
class25	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-	-	-	-	-
class26	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-	-	-	-
class27	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-
class28	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-
class29	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	-
reject	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	2	-	-	-	-	-	-	2

NOTE: The reject class is not in the database.

Misclassification between types of Prints

1 fingerprint in class6 is misclassified as class14. Both are whorls.
1 fingerprint in class18 is misclassified as class11. They are left loop and whorl respectively.
1 fingerprint in class27 is misclassified as class8. Both are left loops.
1 fingerprint in class28 is misclassified as class11. Both are whorls.
1 fingerprint in class30 is misclassified as class19. Both are left loops.
1 fingerprint in class30 is misclassified as class19. They are left loop and whorl respectively.
2 fingerprint in class30 is misclassified as class24. They are left loop and whorl respectively.

As seen above, misclassification tends to happen within the same type of fingerprints. The reason for the cross-type misclassification in class30 is that the fingerprints in that class are of very bad quality. The centerpoints are off and the ridges are smeared by ink. Class 6's center point determination was varying a lot (greater than 50 pixels) which also accounts for the high number of rejects.

However, a test was run by calculating the distance of a fingerprint with itself with its center point shifted by 10 pixels. The result is a distance of 160000 which is significantly below most best match distances which are on the order of 300000.

Whitespaces in cropped images

Because we obtained fingerprints by just pressing the finger down instead of rolling it, many of our fingerprints are incomplete and contain whitespaces due to the center point being close to the edge of the finger. Some of the subjects have small thumb size and thus whitespaces are used to fill up the 256x256 image. However, these whitespaces do not affect the performance of the system since they are consistently present. Gabor filtering will just not detect any ridges in those whitespaces which will accurately represent the fingerprint we obtained.

Comments on 2nd Best Match

NOTE: Each of the 5 fingerprints not rejected by the system has a 2nd best match. This is a list of 2nd best matches and which type they correspond to.
W = Whorl, L = Left loop, R = Right loop, A = Arch, NM = No Match

		Print#					Print type				
class#	type	1	2	3	4	5	1	2	3	4	5
class0	W	21	5	3	11	5	W	W	W	W	W
class1	L	19	8	8	8	23	W	L	L	L	L **
class2	W	13	13	13	13	13	W	W	W	W	W
class3	W	28	28	NM	28	21	W	W	NM	W	W
class4	W	11	11	19	11	11	W	W	W	W	W
class5	W	0	0	11	0	0	W	W	W	W	W
class6	W	21	NM	13	NM	NM	W	NM	W	NM	NM
class7	W	14	14	NM	14	14	W	W	NM	W	W
class8	L	1	27	27	27	1	L	L	L	L	L
class9	W	22	3	0	3	22	W	W	W	W	W
class10	W	12	12	12	12	12	W	W	W	W	W
class11	W	0	0	4	0	4	W	W	W	W	W
class12	W	10	28	11	10	11	W	W	W	W	W
class13	W	2	2	NM	2	19	W	W	NM	W	W
class14	W	NM	21	22	7	2	NM	W	W	W	W
class15	L	26	27	26	26	26	L	L	L	L	L
class16	R	25	25	25	25	25	R	R	R	R	R *
class17	L	15	29	15	29	29	L	L	L	L	L
class18	L	12	23	19	19	19	W	L	W	W	W **
class19	W	13	13	13	13	13	W	W	W	W	W
class20	A	18	18	18	18	18	L	L	L	L	L ***
class21	W	22	NM	22	NM	NM	W	NM	W	NM	NM
class22	W	14	2	14	14	14	W	W	W	W	W
class23	L	19	19	19	19	19	W	W	W	W	W **
class24	W	21	2	21	21	21	W	W	W	W	W
class25	R	16	16	16	16	16	R	R	R	R	R *
class26	L	15	27	1	27	27	L	L	L	L	L
class27	L	26	26	27	15	26	L	L	L	L	L
class28	W	3	3	3	3	0	W	W	W	W	W
class29	L	17	17	17	1	17	L	L	L	L	L
reject	L	NM	25	19	NM	19	NM	R	W	NM	W

NOTE: Reject class is not in the database

The 2nd best match of most classes belongs to the same type of print. This is particularly obvious for class16 and class25. Since they are the only right loops in the whole database, we expect them to be the next closest match of each other. See (*).
Class19 is ambiguous in the sense that it looks both like a whorl and a left loop. This accounts for the inconsistency that is observed in (**).
Class20 is the only arch type fingerprint in the database (***). The next best match is consistently found to be class18.
The next best match for class30 is not L because ridges in the prints are smeared due to bad pressing of finger. Center point location for this class of finger is badly done too.

Comments on no Matches

NOTE: The following table lists all of the Rejects/No Matches produced. In addition, it also shows which best match and second best match produced before being rejected.

Class#-Print#	Best Match	2nd Best Match	B.M Type	2.B.M Type
Class3-3 (W)	0	3	W	W
Class6-2 (W)	7	6	W	W
Class6-4 (W)	7	6	W	W
Class6-5 (W)	7	6	W	W
Class7-3 (W)	7	14	W	W
Class13-3 (W)	13	2	W	W
Class14-1 (W)	14	7	W	W
Class21-2 (W)	21	22	W	W

```
Class21-4 (W)   |     21     |       22       |    W     |     W      |
Class21-5 (W)   |     21     |       22       |    W     |     W      |
Class30-1 (L)   |     25     |       19       |    R     |     W      |
Class30-4 (L)   |     23     |       11       |    L     |     W      |
-------------------------------------------------------------------------
```

VI. Conclusion

This project was a good learning experience for all of us in the group. Besides learning new Signal Processing topics such as Pattern Recognition, Gabor Filtering, and multiple classification schemes, we also learned that in real life, projects such as this one need other factors such as team work, time management, and modular programming.

Given the chance to build a similar system again, we would use a different algorithm for the center point determination that is more consistent and robust. To speed up the system, we can save pre-FFTed Gabor filters somewhere to reduce the need to recompute them for every fingerprint input. Doing so will save 60% of the time required.

We also found the Analog Device's SHARC board very easy to use and a time-saver relieveing much of the strain in memory transferring issues and debugging. We severely recommend switching the course to using SHARC DSP boards.

VII. Code - Sharc implementation

MATLAB portion Center point & cropping	cropscript.m Script to find center and crop center551.m Center point determiner imload.m Loads an 8-bit grayscale image file imdraw.m Draws an image to the screen imsave.m Saves an image as an 8-bit grayscale file
SHARC portion Sectorization, Normalization, Gabor Filtering, Variance	extfeat.c Main loop to extract features (runs on PC) sectnorm.c Sectorization and Normalization for SHARC proj551.h Header file specific for sectnorm.c gaborgen.c Gabor Filter generation for SHARC gaborfft.c FFT code for SHARC trigtbl.h Header file holding trig tables for gaborfft.c convolve.c Convolution and FFTshift for SHARC ezlab.ach Architecture file for SHARC compilation ezlab2.ach Architecture file for SHARC compilation mkbcc.bat Script to compile extfeat.c using Borland C++ mksharc.bat Script to compile all the SHARC code using g21k
PC portion Classification	pc_match.c Classification code that will match or reject proj551.h Header file for pc_match.c mkmatch.bat Script to compile pc_match.c using Borland C++

VIII. References

1. Anil K. Jain, S. Prabhakar, Lin Hong, "A Multichannel Approach to Fingerprint Classification," MSU CPS Technical Report Library.
2. Rao, A. Ravishankar, "A Taxonomy for Texture Description and Identification." Springer-Verlag, New York, 1990.
3. Anil K. Jain, S. Prabhakar, Lin Hong, "Fingercode: A Filterbank for Fingerprint Representation and Matching," MSU CPS Technical Report Library.
4. Anil K. Jain, S. Prabhakar, Lin Hong, "A Filterbank-based Representation for Classification and Matching of Fingerprints," MSU CPS Technical Report.
5. Anil K. Jain, F. Farrokhnia, "Unsupervised texture segmentation using Gabor filters," Pattern Recognition, Vol. 24, No. 12, pp. 1167-1186, 1991.
6. K. Karu and A. K. Jain, "Fingerprint classification," Pattern Recognition, Vol. 29, No. 3, pp. 389-404, 1196.
7. Analog Devices, http://www.analog.com.
8. Bittware Research, Inc., http://www.bittware.com.
9. Borland, http://www.borland.com.