

Digital Communication and Signal Processing System Design

Spring 2002

Automatic Machine Reading of Text

Final Report

Submission date: May 6 2002

Group 8

Adeep Mathur (amathur@andrew.cmu.edu)

Benjamin Schraner (bschrane@andrew.cmu.edu)

Iuliana Tanase (itanase@andrew.cmu.edu)

Department of Electrical and Computer Engineering

Carnegie Mellon University

Pittsburgh, PA 15213

Table Of Contents

1.0 Introduction.....	Page 3
1.1 The Story – How it all started.....	Page 3
1.2 How we approached the subject.....	Page 3
2.0 Bounds.....	Page 4
3.0 Overview of a Typical Algorithm.....	Page 4
3.1 Preprocessing.....	Page 5
3.2 Segmentation.....	Page 5
3.3 Representation.....	Page 5
3.4 Recognition.....	Page 5
4.0 Solution Approaches Explored.....	Page 5
5.0 Approaches Implemented.....	Page 6
5.1 Fourier Descriptors.....	Page 7
5.2 Geometric Moments.....	Page 8
5.3 Nearest Neighbor.....	Page 9
5.4 K-Nearest Neighbor.....	Page 9
5.5 Neural Network.....	Page 9
6.0 Implementation.....	Page 10
6.1 Data Flow.....	Page 10
6.2 Character Segmentation.....	Page 11
6.3 PC-EVM Transfers.....	Page 13
6.4 Feature Extraction.....	Page 14
7.0 Results.....	Page 17
7.1 Impact of the number of features.....	Page 17
7.2 Impact of the number of training characters.....	Page 18
7.3 Evaluation of the algorithm.....	Page 19
7.4 Results of other approaches.....	Page 20
8.0 Memory Allocation And Speed.....	Page 22
8.1 Memory Allocation.....	Page 22
8.2 Speed.....	Page 22
8.3 Parallelization.....	Page 23
9.0 Character Library.....	Page 23
10.0 Code For The Project.....	Page 23
11.0 References.....	Page 24
12.0 Appendix.....	Page 25

1.0 Background:

1.1 The Story – How it all started

The birth of our project came during a brainstorming session. During an extensive process of trying to find an interesting and 551 worthy project, Ben came up with an idea. What if a program could take our handwritten notes and convert them into computer word documents? We all felt this was a very cool proposition and decided to further research in ways we could solve this problem. This is how we ended up in the field of handwriting recognition, and how our project was started.

1.1 Past Projects:

After reviewing the projects from the past years, we found none that could really relate to our particular problem. The closest our project came to past project is in the fact that it's in the field of image processing, however further than that there wasn't much relation between our project and past 551 projects.

2.0 Bounds

Due to the complexity of the problem, we didn't deal with the most general case of handwriting and chose to implement a solution based on the following limiting factors.

Here are the following assumptions we made:

- hand printed & separated characters, (cursive required too great of complexity),
- small case (again to limit the complexity and length of implementing our project),

- straight lined (we had some ideas for implementing diagonal and various slanted writing, but didn't get the time to implement the code).
- the whole page is perfectly aligned with the scanners and doesn't need to be rotated.
- utilized the handwriting of one person in order to maximize the intelligence of the program, (Iulia's handwriting was the one used for that purpose)

3.0 Overview of a Typical Algorithm

There are a variety of different approaches to recognize handwritten text. However, the main process can be generalized and divided up into four major stages as described in [1]:

- 1) Preprocessing
- 2) Segmentation
- 3) Representation
- 4) Recognition

In the next paragraph, we will give a short description on what these different stages mean.

3.1 Preprocessing

Once the scanned page is converted to a raw format, the image has to be preprocessed in order to produce data that ensures that the recognition system can work accurately. Preprocessing includes:

Noise Reduction: Noise is introduced by the optical scanning process and causes disconnected line segments, bumps and gaps in lines.

Normalization: The goal is to remove the variations of the writing in order to obtain standardized data. The writing may be curved and therefore the baseline would have to be extracted in order to put the words on straight lines.

3.2 Segmentation

Now that the data is clean, the scanned page can be divided into its subcomponents. First of all, the lines have to be extracted; after that, the words need to be segmented into characters. It is important to mention that segmentation of cursive script letters is still considered an unsolved problem [1].

3.3 Representation

In order to reduce the complexity of the recognition algorithm and to increase its accuracy, a more characteristic and compact representation is needed. For instance, the representation has to compensate for variations in slant, size and rotation. The normalized image of a character needs to be parameterized in an appropriate manner to facilitate the recognition process.

3.4 Recognition

The recognition algorithm has to assign to each unknown character a character of the predefined library. A variety of classifiers might be used to address this task.

4.0 Solution Approaches Explored

There are a couple of different approaches that deal with online handwriting recognition (OCR). After research and discussing with different people from the field, we identified following main methods:

- Correlation Filters (CF)

- Principle Component Analysis (PCA)
- Geometric Moments (GM)
- Fourier Descriptors (FD)
- Special Features (for example skeleton coding)

All of these methods have their advantages and drawbacks. It became clear that today's algorithm usually involve dictionary based solutions, which means that in the recognition stage the output is always compared to a table of possible words; strings that don't match the table can be discarded. However, due to the given time constraints and the limitations of memory on the DSP board, it was not possible to use the content of a dictionary in our project and our choice depended therefore on the simplicity of the algorithm. According to [6, p.370], Fourier Descriptors proved to be useful in character recognition problems and were actually used in [2] and [3]. We therefore decided to base our system on this representation. We also tried to use the Geometric Moments and experimented with different classifiers such as Nearest Neighbor, k-Nearest Neighbor and a Neural Network. In the following paragraphs we will describe these methods and the results we could achieve with them.

5.0 Approaches Implemented

Here is a discussion of the theory behind each of the algorithms we implemented in order to solve our problem. We tried two types of feature descriptors, and two types of classifiers. Some worked out well, and some not so great. A more detailed discussion is provided in the result section (7.0).

5.1 Fourier Descriptors

Fourier Descriptors were the first method we implemented, and in the end also turned out to be the method with the best performance. They are obtained as follows:

1. Extract the boundary of a character image
2. Sample the boundary at L points
3. Calculate the L-point DFT of both x and y coordinates separately
4. Normalize the so obtained coefficients

Mathematically speaking, if $x[m]$ and $y[m]$ is a vector containing the x-coordinates and y-coordinates respectively of the boundary points, the searched coefficients are defined as:

$$a[k] = \sum_{m=0}^{L-1} x[m] \exp\left(-jk\left(\frac{2\pi}{L}\right)m\right) \text{ and}$$

$$b[k] = \sum_{m=0}^{L-1} y[m] \exp\left(-jk\left(\frac{2\pi}{L}\right)m\right).$$

Those coefficients are defined for $k \in [0, L-1]$, where L is the number of sample points. The coefficients $a[0]$ and $b[0]$ are discarded as they contain the DC components of the image and only represent the position of the image. The remaining coefficients are invariant to translation and form a feature vector. Only the absolute value of the complex coefficients a and b are used. The feature vector is defined as:

$$f = (|a[1]|, |b[1]|, |a[2]|, |b[2]|, \dots, |a[L-1]|, |b[L-1]|).$$

This vector is normalized so that its size becomes 1. The so obtained vector was used with a Nearest Neighbor classifier. In order to determine the distance of two-feature vectors f_1 and f_2 , the Euclidean distance was used:

$$d = \|f_1 - f_2\|$$

In the training stage, the feature vectors of the training characters are stored in a library along with their character labels, which is a number specifying the character type. In the recognition stage, the feature vector of the character to be recognized is compared to every character in the library and the result is defined to be the label of the feature vector that is closest to the input vector.

5.2 Geometric Moments:

This is the second method we experimented with, which turned out not to provide results as good as the Fourier Descriptors.

The $(p + q)$ th-order moment is defined as

$$m_{p,q} = \sum_x \sum_y f(x, y) x^p y^q .$$

It is not practical to use these moments directly because it is necessary that the moments are invariant to translation and scale. In order to compensate for translation $x' = x + \alpha$, $y' = y + \beta$ the central moments can be used:

$$\mu_{p,q} = \sum_x \sum_y f(x, y) (x - \bar{x})^p (y - \bar{y})^q$$

$$\text{where } \bar{x} = m_{1,0}/m_{0,0} \text{ and } \bar{y} = m_{0,1}/m_{0,0} .$$

Finally, the invariants to scale change $x' = \alpha x$, $y' = \beta y$ are defined as

$$\eta_{p,q} = \frac{\mu_{p,q}}{\mu_{0,0}^\gamma}$$

with $\gamma = (p + q + 2)/2$

In our experiments we used the moment invariants $\eta_{p,q}$ and stored the different moments in a feature vector. One of the difficulties we encountered was to determine which moments and up to which order we should use.

5.3 Nearest Neighbor

The Nearest Neighbor method is a very important algorithm in classification and is widely used in high-dimensional spaces. It probably represents the easiest way to match an input vector to a library as required in our project. In this approach, the whole library is searched for the element that is closest to the input vector in terms of Euclidean distance defined as $d = \|v_1 - v_2\|$, v_1 and v_2 being two vectors. In our case, the result of the nearest neighbor algorithm is the label of the nearest neighbor.

5.4 K-Nearest Neighbor

K-Nearest Neighbor is a straightforward extension of the Nearest Neighbor algorithm. Instead of considering only the nearest neighbor, the k nearest neighbors are compared to determine the result. There is a variety of ways to perform this comparison, for example the weights of each of the k nearest neighbors can be set to a distinct value in order to emphasize the best estimates differently.

5.5 Neural Network

The neural Networks (NN) technique is proposed in almost every paper that deals with online character recognition. We therefore adopted the idea and tried to integrate it with the Fourier Descriptors (FD). In our experiments we used as input to the NN the

features obtained from the FD. We used one hidden layer containing 5 nodes and tried different parameter for learning rate and saving the system at different epochs.

6.0 Implementation

In this section we explain in detail how we implemented our algorithm on the EVM, which was quite different from our test solution on PC.

6.1 Data Flow

The process starts with a paper being scanned into the computer and then converted from jpeg to raw (ppm) format. Then this picture is processed on the PC side and characters are parsed from the page. Each character is sent to the EVM where based on the sent information the board recognizes as either a training or recognition character.

The first characters sent to the EVM board are part of the training process and they serve to create a database of features on the EVM board which will characterize and help recognize features on the unknown data scanned pages. The next stage is to send over scanned pages with unknown data, which will be recognized utilizing the features collected from the training process. The chart in Figure 1 demonstrates this process.

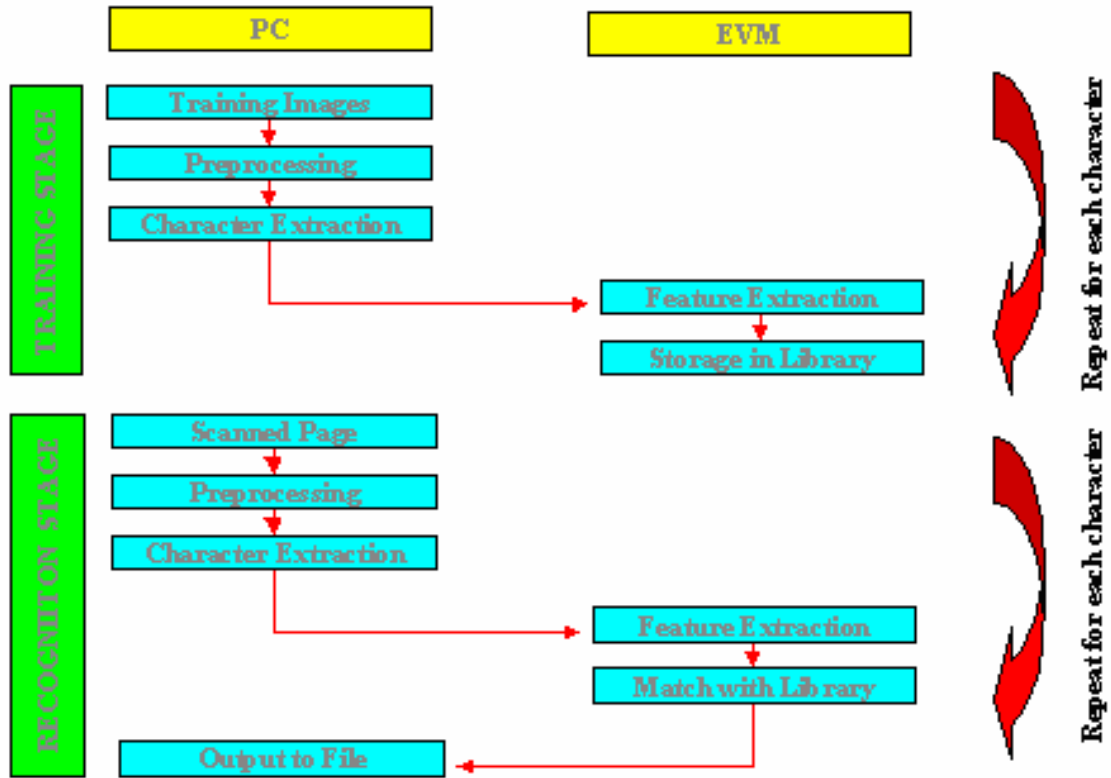


Figure 1: Graphical Representation of Data Flow.

6.2 Character Segmentation

First a document is scanned using an Epson Printer. After experimenting with various resolutions we found that 360 dots per inch resolution maximized both the number of pixels we could fit on the page (we wanted to limit to less than 4020 x 4020 pixels for a 11 x 8.5 inch paper), and at the resolution of the letter. Next the picture is manually converted to raw format using Xnview, (downloaded from www.xnview.com).

Using the ppm interface code, which we obtained from 18-798, we wrote C code that parsed this image file. The first step in the parsing was to quantize picture in order to reduce complexity and remove noise. We did this by choosing pixel threshold of 120. If pixel lower than that, we changed the value to 0 (black); if it was higher we changed it to

255 (white). This approach worked pretty well in eliminating noise on the average lined sheet of paper.

Then, a vertical histogram is created for the image, and based on where low thresholds occur, the lines are parsed out of the page. For each of these lines, a horizontal histogram is created using the same process as the vertical histogram; and then the coordinates of about where each character begins and ends are estimated. An example of the parsed character output is shown in Figure 2.

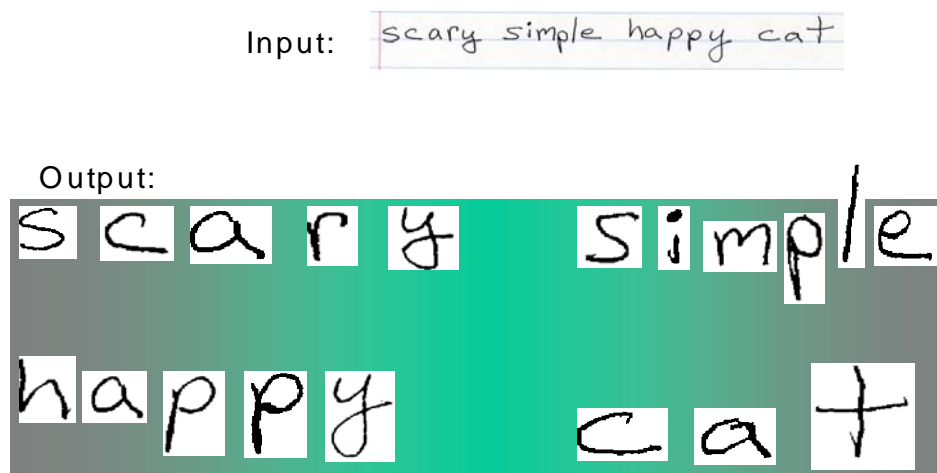


Figure 2: An example of the parsed output of a scanned file

Next, information on each of these parsed images are stored within a data structure, which helps us keep a cleaner track of what needed to be sent to the EVM.

6.3 PC – EVM Transfers

After the pre processing, the image is transferred to the EVM for character recognition. Transfers take place for three cases; ‘Training’, ‘Evaluation’ and ‘Run’. ‘Evaluation’ and ‘Run’ come jointly under the ‘Recognition’ case.

Using the previously defined data package, transferring of information occurs from the PC to the EVM. Besides the binary pixel data of the image, information in the data package includes height and width of the image. The struct also includes a flag called ‘mode’, which as its name suggests is used to separate between Training and Recognition mode. If we are in the Training mode, we also pass the character label (a, b, c...) and trainee number (0,1, 2...) in the data package so that the information is stored at the right place in the EVM feature library. There is a restriction on the dimensions of the image that is being transferred. It cannot be greater than 150*250 in size.

On the EVM side, the connection to the PC is in a continuous loop. There are only two cases, sending data from PC to EVM and from EVM to PC. We slightly modified the code from lab 3 in order to create the transfers between the PC and EVM board. We did not get the time to optimize the transfers between the PC and EVM using the DMA transfer function.

For transfers from EVM to PC side we defined a struct, EVM2PC_INFO. If in the Training mode, this struct is then just a signal for the PC to send the next training character. If in the Recognition mode, it carries the recognized ASCII character.

Processing in our program includes finding the boundary points and storing its Fourier Descriptors in the Feature Vector when in the Training mode.

6.4 Feature Extraction:

We coded up the methods described in the Algorithms section (5.0), however in order to extract the necessary information from the picture we needed to the boundary of the letters. The methodology in the code of the boundary extractor is explained bellow.

6.4.1 Boundary Extraction:

The purpose of the algorithm is to extract the information of the boundary (contour) of a segmented character and present it in a more compact form. There are various methods for boundary tracing. *Anil K. Jain* talked about boundary extraction using the “crack following algorithm” [6]. In this algorithm each pixel is viewed as having a square-shaped boundary, and the object boundary is traced by following the edge-pixel boundaries. He also mentioned “Edge Linking and Heuristic Graph Searching” algorithms ^[1], but in our project we were interested in the actual path taken and not determining the global optimum path. *Yuk Yik Chung* and *Man To Wong* suggested the “four neighbor” adjacent method for boundary extraction [3]. In our project, we have modified the “four neighbor” method to a “eight neighbor” method so that it can be used along with the “crack following algorithm”.

Basically the algorithm scans the binary image until it finds the boundary. Once a black pixel is detected, it will check another black pixel and so on. The tracing will follow the boundary automatically. When the first black pixel is found the program will be assigned the coordinates of that position to indicate that this is the origin of the boundary. The tracer will search for the next nearby black pixel. A boundary pixel is defined as having a black value and out of the four neighbors in the top , bottom , left

and right , one of them is a white pixel. The searching will follow according to the last direction taken and move in a clockwise direction. This is shown in Figure 3.

1	2	3
0	X	4
7	6	5

Figure 3: The center pixel X is the boundary and the eight directions are given by the above numbering.

For instance, if the previous boundary point was found in direction 4, then the search for the next point will start from direction 4 and move in a clockwise direction. The search will skip the direction symmetrically opposite the beginning direction. That is, if the search starts from 4 it will skip direction 0, if it starts from 6 it will skip direction 2. Otherwise chances are that the tracer will be moving back and forth between two points. As the tracer moves along the boundary of the image, the corresponding coordinates are stored in an array for the computation for Fourier Descriptors Calculations. During the boundary tracing the program will check if the boundary coordinates are equal to the first coordinates (the origin). If they are equal, that means the whole boundary has been traced. The next two figures show an example of this process.

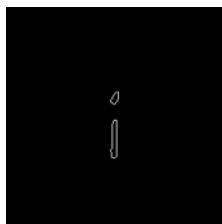


Figure 5: The traced boundary



Figure 6: The original image

For i 's and j 's the boundary extraction algorithm will have two starting and ending points because of the dots.

A problem arises with this algorithm when the characters happen to have very sharp edges. In that case the boundary tracer can again be going in circle with a particular set of points. To avoid that the program has tests to ensure that the point traced is not the same as the one found three to four steps ago.

A case where boundary extraction would fail to give the points for Fourier Descriptors Calculation would be when the characters are not "closed". In this case the boundary extracted would show an inner and outer boundary. The next two figures show an example

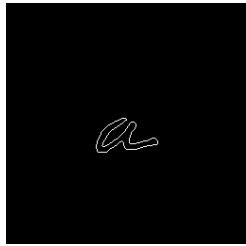


Figure 7: The traced boundary



Figure 8 : The original image

The figure on the left is the boundary of the character "a" on the right. Since "a" is disjoint the boundary traces an inner and outer boundary.

One way of dealing with this would be to have a large number of training characters and increasing the number of sampling points used for Fourier Descriptors.

7.0 Results:

7.1 Impact of the number of features:

One important parameter is the number of features used. If the number of features is too small, the different characters might not be distinguishable; on the other hand a too big number of features slows down the recognition process and might even degrade the performance. We tried to maximize the recognition rate in function of the number of features. The figure below shows a plot of recognition rate versus number of features.

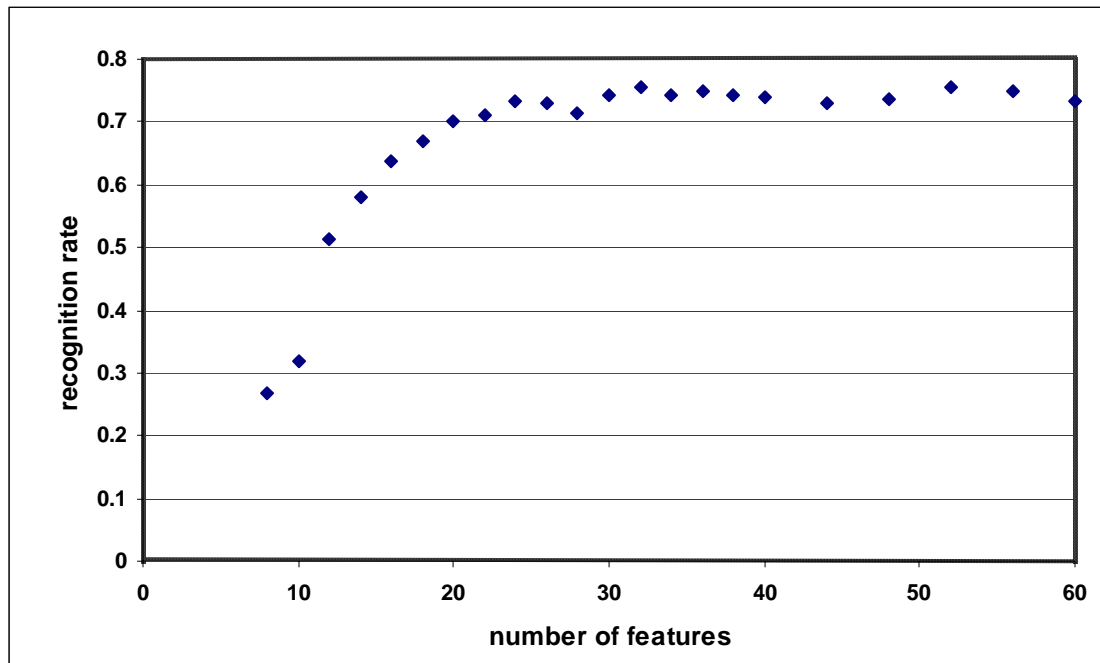


Figure 9: Demonstrates the recognition rate versus number of features.

We recognize that the recognition rate is very low for less than 20 features and gets into saturation when the number exceeds about 35. For this reason it is not necessary to have more than 35 features. Note that this simulation was done on the PC. On the EVM only the radix 2 and radix 4 DFT are implemented and optimized for fast computation. Therefore we chose a number of 32 features which are obtained by calculation a 16-point DFT of both x and y coordinates of the sampled boundary points.

7.2 Impact of the number of training characters

The number of training characters is also a factor that can affect the performance of the recognition algorithm. In order to examine the impact of this parameter we ran a set of evaluations with different library sizes and reported the recognition rates in a table as shown in Figure 10.

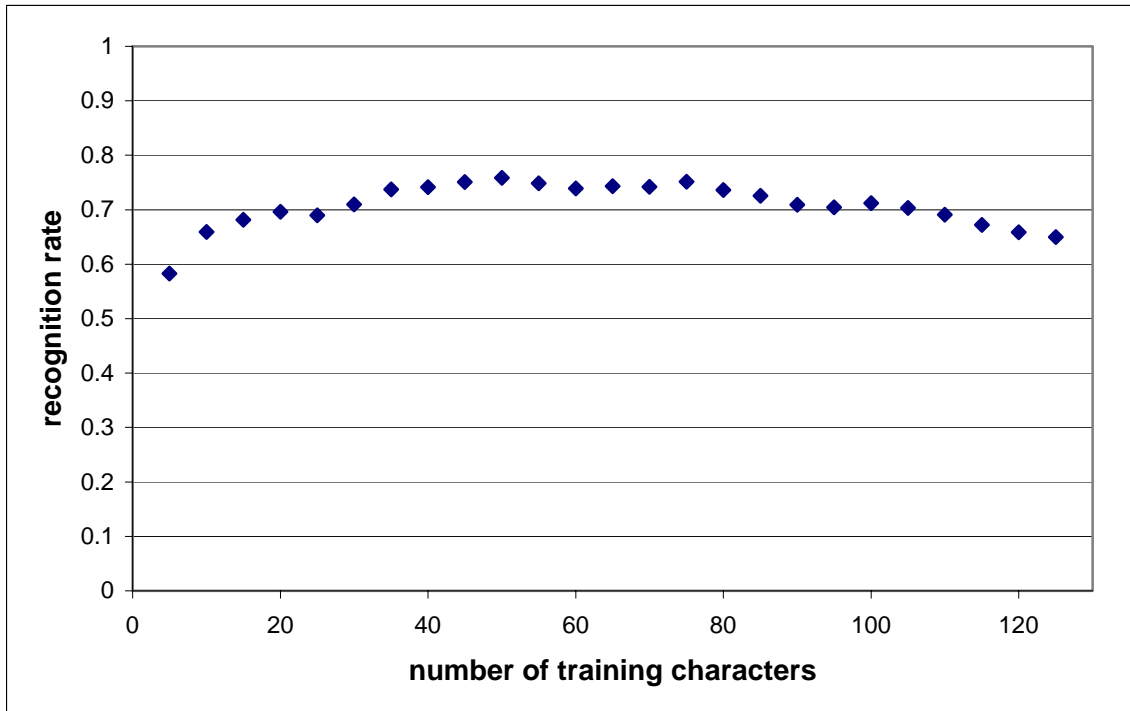


Figure 10: Demonstrates the impact of the library size on the recognition rate.

We recognize that even for a small set of library characters the result is quite good. For an increasing number of training characters the results get better but start to decrease after having reached a maximum at 50 training characters. This is not a surprise considering the fact that if the size of the library is big, it is more likely to have characters in the library that are messed up.

7.3 Results of the Evaluation

With our best configuration the maximum recognition rate was about 75%. An evaluation table based on a set of 50 training and 50 test characters is shown in Figure 10. The first element in a row indicates the input character and the following columns list how many times the character was recognized to be the corresponding characters in the first column. Ideally, if the recognition rate was 100%, and all numbers would be on the diagonal.

	a	b	c	d	e	f	g	h	l	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	49																	1								
b		31		10												7		1								1
c			37		13																					
d		16		28												6										
e			7		41											1						1				
f				1	1	37			1	3				1		2	3	0								1
g							41	2		1						1	2			3						
h								44										3		2					1	
l						2		2	42	2															1	1
j						2			10	34		1					1			1						1
k				1				1			33									2				1	12	
l		3		3	1		1				1	28			1	6	2								1	3
m													26									1	1	22		
n	1												1	33								12	2	1		
o															50											
p		10		9				3			1						27									

q		1		1															4	40							1						1	1				
r	14			1																1	31												1					
s			1	2	1																											1	6					
t				1		5	1														5												7	1				
u					1																						1					44	3	1				
v	1																																7	39				
w	1																																1	44	1			
x		2		1			1	1																										1		40		
y																																				49		
z		1																																	8		1	40

Figure 11: Evaluation table for a set of 50 training and 50 test characters.

We clearly see the impact of neglecting the phase of the Fourier Transform in the normalization process. The features not only become invariant to rotation but also invariant to reflection at the center. This causes the confusion between letters b, d, p and q; m and w; s and z; n and u. Further confusion exist between k and x, which can be quite similar in terms of the boundary. Other problems are caused in the segmentation algorithm; for example if the segmentation cuts of a character, the boundary algorithm produces a completely different boundary, which leads to a random result.

7.4 Results of other approaches

Geometric Moments:

The success achieved with geometric moments was very limited. Our experiments showed that this method seems to be very sensitive to light variations in character shape and noise. It seems that this approach is not very good idea to address our problem.

Using the moments up to the order of 10, the best result that could be achieved was 35% recognition rate. Table I presents the result for different number of moments.

<i>Order</i>	3	4	5	6	7	8	9	10
<i>Rate</i>	0.19	0.34	0.35	0.35	0.35	0.33	0.334	0.33

Table I: Recognition rates for Geometric Moments up to the 10th order.

K-Nearest Neighbor

Using K-Nearest Neighbor did not improve the overall performance of our system. The problem was the correlation between the k best estimates. It turned out that when the first estimate was wrong, the other estimates were wrong, too. In general, the problem has to be searched in either segmentation or boundary extraction. In fact, if one of these two steps produces an error the output is pretty random. However, the results obtained by the K-Nearest Neighbor are very interesting as they allow analyzing in detail the performance of the algorithm, and help to figure out possible problems.

Neural Network

As Geometric Moments, Neural Networks were only tested on the PC and produced and were not that successfully. On the training data the recognition rate was 80% but on the test data we could only achieve 40%. The reason for the low success is related to the low number of training characters we used. We only used 50 characters, which is not enough, but due to given time constraint it was not possible to build a library big enough to train the system to a reasonable extent. The code we used for our experiments is used in 15-492.

8.0 Memory Allocation and Speed

8.1 Memory Allocation

We found the amount of memory space we utilized on the board greatly depended on the size of character library; for the a library with 50 versions of each letter we were able to fit our program on SBSRAM_PROG, however when we increased each letter version to 100, we had to move it to SDRAM0. Here are the memory allocation for each version.

For a 50 version per letter library:

<u>Description</u>	<u>Memory Used</u>
Variables are stored on ONCHIP_DATA:	704 bytes
Code is stored on SBSRAM_PROG:	67 Kbytes
Feature vector data stored on SBSRAM_PROG:	167 Kbytes

For a 200 version per letter library:

<u>Description</u>	<u>Memory Used</u>
Variables are stored on ONCHIP_DATA:	704 bytes
Code is stored on SBSRAM_PROG:	67 Kbytes
Feature vector data stored on SDRAM0:	1,067 Kbytes

8.2 Speed

Utilizing the profiler, we timed the amount of time the major functions of our program took to execute. The results are as follows.

<u>Description</u>	<u>Lines of Code</u>	<u>Avg number of Cycles</u>	<u>Avg Time</u>
Main part of loop	119 to 159	2370840	94.8 ms
Character Training	176 to 185	426381	17.1 ms
Character Recognition	186 to 197	17770840	710.8 ms

8.3 Parallelization

By examining the assembly file, we discovered that the number of iterations for all our main loops was 4.

9.0 Character Library

In order to properly train the algorithm to recognize handwritten letter, known versions of each letter had to be scanned in and processed using the preprocessing code. It was discovered that in order to attain intelligent results, the library had to be greatly expanded. Thus, in the end the resulting library consisted of 200 versions for each of the letters in the alphabet.

10.0 Code For The Project

We wrote most of the code used in programs, because the literature online provided us mostly with algorithms and pseudo code, but no actual hard code we could use. Thus the only code we did use was the FFT code from lab 3 on the EVM side, and the pc side we utilized some of the ppm code and classes from our 798 class, which helps in reading and manipulating a raw data format image.

11.0 References

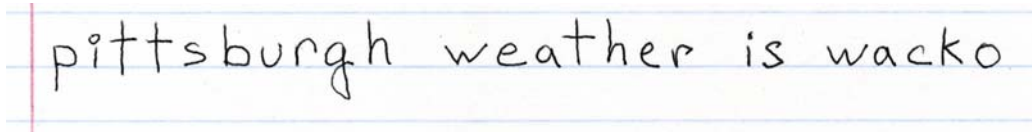
- [1] *Nafiz Arica and Fatos T. Yarman-Vural. An Overview of Character Recognition.* Middle East Technical University, Ankara, Turkey, 1999.
- [2] *Ian Phillip Morns and Satnam S. Dlay. The Dynamic Supervised Forward-Propagation Neural Network for Handwritten Character Recognition using Fourier Descriptors and Incremental Training.* Electronics, Circuits, and Systems, 1996. ICECS '96, Proceedings of the Third IEEE International Conference on , Volume: 2 , 1996.
- [3] *Yuk Ying CHUNG. Handwritten Character Recognition by Fourier Descriptors and Neural Network.* TENCON '97. IEEE Region 10 Annual Conference. Speech and Image Technologies for Computing and Telecommunications, Proceedings of IEEE , Volume: 1 , 1997.
- [4] *Ian Phillip Morns and Satnam S. Dlay. The DSFPN, a New Neural Network for Optical Character Recognition.* Neural Networks, IEEE Transactions on, Volume: 10 Issue: 6 , Nov. 1999.
- [5] *R. Cederberg. Chain-Link Coding and Segmentation for Raster Scan Devices.* *Computer Graphics and Image Proc.* 10, (1979): 224-234.
- [6] *Anil K. Jain. Fundamentals of Digital Image Processing.* ISBN-81-203-0924-4.

12.0 Appendix

Here are the results after running our code on five sample files

12.1 Test 1

Input:



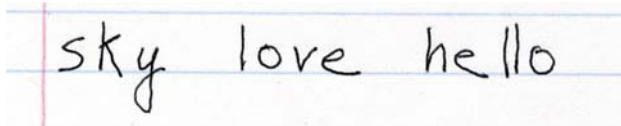
pittsburgh weather is wacko

Output:

aittkburuh weagher iz sack y oy a

12.2 Test 2

Input:



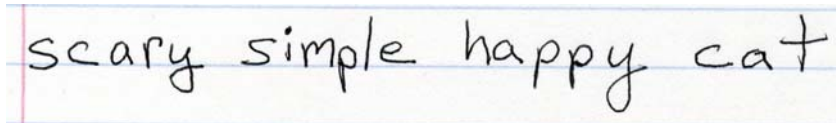
sky love hello

Output:

sxg love hcll a

12.3 Test File 3

Input:



scary simple happy cat

Output:

scayk zimple happu c

12.4 Test 3

Input:

in the jungle the mighty jungle the lion
sleeps tonight

Output:

aiv fhe jvnlle the wishgy junglc the s le ep s toniuhg
m

12.4 Test 4

Input:

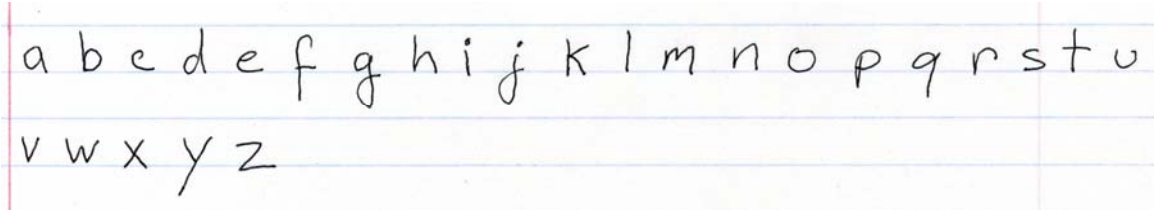
love is in the air

Output:

lore fs in kheai

12.5 Test 5

Input:



abcdefghijklmnopqrstuvwxyz

Output:

apchef ghij k lm no pqrpuwx vz

12.6 The Files

1. filterpc.c :

This file contains the code for the PC side

2. filterevm.c :

This file contains the code for the EVM side

3. ocr_constants.h :

This header file contains the constants and data structures used in our program