Applying Optical Character Recognition (OCR) for Recognizing Texts Written in a Specific Persian Font: Final Document

A thesis submitted to BIHE for the degree of B.S. in the faculty of Computer Engineering

Submitted by: Vargha Dadvar

Submission Date: July 2013

Project Supervisor: Fares Hedayati

Table of Contents:

Abstract:

Recognition of the text in scanned documents of older Persian writings is a challenging task, because of the properties of such scripts and the cursive property of Persian writings in general. As separate segmentation of Persian characters is much more difficult than in Latin and some other alphabets, using segmentation-free approaches are more useful in Persian OCR. In this project, one of such methods called the Generalized Hidden Markov Models (or gHMM) is used for the recognition of texts written in a specific Persian font, which was mostly used in older Persian documents and books. The gHMM not only helps to combine text line segmentation and character recognition processes, but also provides a way to take advantage of both the character image properties and the language model in the service of character recognition. The language model probabilities and the parameters computed from the training character features are used in the recognition process, and should be prepared during the training phase. The main recognition program implemented in this project firstly detects and corrects the skew of the input pages and detects the text lines. Then in the body of the recognition process, a generalized version of the Viterbi algorithm is used to recognize simultaneously the best segmentation and the best character sequence corresponding to each text line. The Modified Quadratic Distance Function (or MQDF) is used in the heart of the Viterbi algorithm in order to measure the closeness of each character image with each of the character classes. The recognition accuracy of the implemented OCR program is 85.5%.

1. Introduction

This paper is the final document of my final thesis for the B.S. degree of Computer Engineering in BIHE. This project includes a study about different aspects of the Optical Character Recognition problem, and also the implementation of an OCR system for a specific Persian font used in the printed version of some older Persian books. In this document, the theoretical foundations and the practical considerations of designing and implementing the OCR program are explained.

Optical Character Recognition (OCR) is the process of recognizing the text in the scanned images of the documents or books. Despite advances in the recognition of Latin writings and texts written in some other alphabets, research in Persian and Arabic OCR and handwriting recognition still continues. Some background information about the OCR problem is given in Section 2. As one of the possible applications of Persian OCR, there are many historical and older Persian books that should be converted to computer-readable text. In this project, one of the fonts and writing styles used in such older Persian books is chosen, and an OCR program is implemented for recognizing the texts written in this font. Persian writings are cursive and most of them include some other complexities, as will be explained in Section 3, and this makes the recognition of such texts difficult and more complex.

Hidden Markov Models (HMM) is the main mathematical model used in this project for solving the character recognition problem, as it is detailed in Section 4. HMM helps to use both the target language properties and image properties of the characters for the recognition process. A generalized version of HMM also helps to combine the text line segmentation and the character recognition processes, as this approach is preferable in Persian OCR.

Before the text line images are used as inputs to the recognition process, some preprocessing should be performed on the scanned page images, such as skew detection and baseline detection, which will be explained in Section 5. Section 6 introduces the features which are used in this project to capture the properties of character images, so that the appearance of the characters can be distinguishable by the OCR program. As the OCR system should be trained to have the capability of recognizing the characters of a text line image, a separate training process should be done. In this phase, the language model probabilities and the parameters needed for predicting the character corresponding to each character image are trained (i.e. prepared) by analyzing the type-written text version and the scanned images of a training text document. The details are explained in Section 7.

In Section 8, the concept of distance functions is introduced as a special function which is used to compute the level of closeness between each character image and a character class. This function is used in the body the Viterbi algorithm, which is responsible to find the most probable character sequence for an input text line image. Also in this section, we will see how the generalized Viterbi algorithm is used in this project to find the best segmentation of the line and the best character sequence simultaneously. Section 9 includes a brief explanation about the software technologies used to implement this OCR system. The results of testing the implemented OCR program are discussed in Section 10, in addition to the explanation of some tasks done in order to improve the accuracy of the system.

2. Some Background on Optical Character Recognition

Optical Character Recognition (OCR) is an Artificial Intelligence field which aims to correctly detect and recognize the text sequences in scanned documents and writings. Recognizing the text in such documents makes these sources of information searchable and manipulable by computers, and prepares them to be used as input for text processing and many other computer applications. OCR can be used in different types of applications, including the recognition of handwritten documents, recognition of machine-printed documents, online recognition of text input in PDAs and smart-phones, recognition of numbers in forms and car number plates, etc.

OCR technology and software have been developed quickly in recent years, and many developed OCR solutions offer very accurate recognition of text with very low error rates. This level of accuracy in OCR technology has become possible in part because of the advancements in Machine Learning, an area in Artificial Intelligence which helps to promote methods for learning knowledge and logic from large amounts of data. Despite these developments in the recognition of texts written in Latin and some other alphabets, OCR research still continues in certain more specific areas, including recognition of texts in other languages such as Arabic and Persian scripts, as well as handwriting recognition in different alphabets.

In almost all of the OCR methods and implementations, certain types of tasks should be done sequentially in the recognition process, including noise removal, document skew detection, baseline detection, character segmentation, and finally character or text recognition. In some languages, segmenting the text line into characters is easier because of the detectable space between the characters. However, Arabic and Persian texts are cursive and therefore automatic character segmentation is a very difficult task in such languages. As a result, most of the methods proposed for Arabic and Persian OCR use a holistic approach, meaning that the characters are not separated from each other before the recognition process, and the recognition is done in word or text block level. A survey and review of the most important researches in Arabic and Persian OCR and handwritten recognition can be found in [1] and [2].

Among the methods with holistic approach to character segmentation, those using Hidden Markov Models are of great importance. Hidden Markov Models (HMM) make it possible to take advantage of both the language model and the image properties of characters in order to recognize the text sequences correctly. One approach in using HMM for OCR is to use a universal model that is composed of smaller interconnected character models, each of which represents one letter from the alphabet [3]. In another method which is useful for small vocabularies of words, a separate HMM is used for each word class [4]. Another way to use HMM without a previous character segmentation phase is to combine the recognition process and the segmentation process. That is, the final recognition process iterates over all possible segmentation states, and for each of them computes the best character sequence. In [5] this method is called Generalized Hidden Markov Models (gHMM), and is successfully used for recognition of Tibetan wood block prints.

In our project, the aforementioned Generalized Hidden Markov Models is used as the main recognition methodology in order to recognize the text in some older Persian writings. Because of the learning purposes of this project, the scope of the target texts and fonts chosen are limited, and we concentrated on a specific older Persian font for this project. This helps to concentrate on implementing different stages of the OCR process in this period, rather than spending time on gathering training data for different types of Persian fonts and writing styles, and addressing the wide range of requirements of these different fonts and texts.

3. About the Persian Writings and the Target Font Used

3.1. Properties of Persian Scripts

Persian alphabet includes 32 main letters, four more than the main characters of Arabic. Most properties of Persian and Arabic alphabet are the same. As in Latin alphabet where each character has two different forms (capital and small), in Persian and Arabic each letter has four different forms based on the place of a character in the word and the characters before and after it. These forms of a character include initial form, medial form, final form, and the isolated form. In Table 1, different forms of some Persian letters are shown. It can be seen that the difference between some characters are only in the number of dots or other extra strokes they have, and their main body is almost the same. For example the letters KAAF (ک) and GAAF (گ) only differ in the extra stroke above GAAF, and the difference between characters TEH (ت) and SEH (ث) is in the number of dots above their main body. It would be very challenging for the OCR system to distinguish between such similar characters.

	Initial Form	Medial Form	Final Form	Isolated Form
ALEPH	ا	ـا	ـا	ا
TEH	تـ	ـتـ	ـت	ت
SEH	ثـ	ـثـ	ـث	ث
KAAF	کـ	ـکـ	ـک	ک
GAAF	گـ	ـگـ	ـگ	گ
NOON	نـ	ـنـ	ـن	ن
VAAV	و	ـو	ـو	و

Table 1 - Different forms of some of the characters of the Persian alphabet

As different forms of characters suggest, Persian writings are cursive, and most of the characters are connected to each other within a word. This characteristic imposes a great challenge to text segmentation in Persian writings. The lack of space between characters is sometimes even more intense so that the two adjacent characters have vertical overlap with each other. Moreover, some characters are deformed when connected to each other which form a ligature, as another complexity in Persian and Arabic scripts. Also in Arabic and Persian, sometimes a character is accompanied with diacritics or symbols above or below its body, mainly the vowel symbols. These diacritics are different from dots or other symbols which are inherent to some characters. Some of these complexities in Persian and Arabic scripts are illustrated in Figure 1 below.

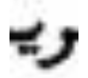
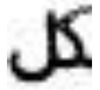
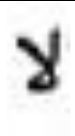
				
Examples of vertical overlaps between adjacent characters		An example of ligature of two connected characters	Examples of vowel symbols above the characters	

Figure 1 - Some of the complexities in the form of characters in Persian and Arabic writings

3.2. Properties of the Target Font and Text

The font analyzed in this project is a rather old typewritten font used in some older Persian writings. As an example, this writing style was used in the printed version of a series of holy Baha'i books called *'Maede-ye-Asemani'* (مائده ی آسمانی). Some volumes of this book were used as the main training and test writings of this project. In addition to the complexities of Persian and Arabic writings mentioned above, the target Persian font used in this project also has its own properties. Firstly, there is little regularity in the way spaces are used within and between words in the target text. For example, in some situations where words are finished with characters like REH (ر) or VAAV (و), there is no space between the current word and the next word, and sometimes there is overlap between the final character of a word and the first character of the next word. On the other hand, in some situations an extra space is seen between the characters of a single word. Figure 2 shows examples of such irregularities in the used document.

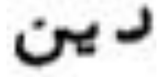
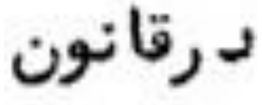
	
An example of space within a word	An example of lack of space between two words, while the space is present within the first word

Figure 2 - Examples of irregularities in the way spaces are used in the text used in this project

In many typewritten Persian texts (mostly the older ones), sometimes the characters are stretched for text justification purposes, and this leads to some deformation in the appearance of the characters. Also in the target texts of this project, sometimes at the end of lines one or two characters are translated a little above the baseline. These complexities are shown in figures below. All these irregularities impose challenges to the recognition process.

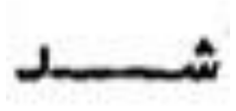
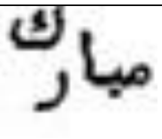
	
An example of stretch in the first character of a word	An example of a character translated above the baseline

Figure 3 - Examples of character stretch and character translation in the target text

3.3. Selection of Character Classes Covered by OCR System

Selection of the character classes to be covered by the OCR system was a difficult task, mainly because of the irregularities regarding the potential vertical overlaps between adjacent characters. Firstly, although some forms of the same characters have very similar appearance, we need to regard these similar forms as separate character classes in the OCR system, mainly because the transition behavior of various forms of a single character are different (Refer to Section 4 about the definition of transition model). For example, final form of the letter NOON (ن) always joins a middle or initial form of a character, while the isolated form of NOON (ن) always follows an isolated or final character in a word. In addition to different forms of the main characters, some other characters and symbols were also chosen to be included in the list of character classes, mainly based on their frequency in the target texts. Arabic digits from 0 to 9, some symbols like colon and parentheses, and some other Arabic characters like HAMZEH (ء) and TEH_MARBUTEH (ة) are among such characters.

As mentioned above, characters which have vertical overlaps when connected to each other can be sources of error in the recognition process. The reason is that during the recognition process, in such cases finding the best segmentation point is very difficult for the algorithm, and thus overlapping characters may not be segmented and thus recognized correctly. Therefore, it may be a good practice to also include some combinations of characters in the list of character classes, especially those combinations representing overlapping characters. This decision can help the software to recognize combinations of characters where the single characters are tightly connected and are vertically overlapping each other. In addition, some characters like ALEPH (ا) are so small in width that it may be better to consider their combinations with some other characters as new character classes. However, using combinatorial character classes has its own problems and limitations. Firstly, hundreds or thousands of such combinations are present in Persian scripts like the writings used in this project, and it is not feasible to include all these combinations as separate character classes. Moreover, the overlapping characters problem is not just limited to two-character combinations. For example, a sub-word like “ورا” in our target texts involves vertical overlaps between VAAV (و) and REH (ر) and between REH (ر) and ALEPH (ا). Because of these considerations, we decided to only include some more frequent combinatorial character classes in which the characters were tightly connected or were overlapping each other a lot.

On the whole, about 320 number of character classes were selected to be covered by the implemented OCR software in initial tests. In order to increase the accuracy and stability of the program, some of the character classes without a least number of training examples were eliminated in the final tests of the project.

4. The Main Recognition Approach

In this section, the main recognition methodology used in the project is explained, as well as the related theoretical foundations. As mentioned previously in this document, Hidden Markov Model (HMM) is the main mathematical tool used to solve the OCR problem in this project. In a probabilistic problem modeled by HMM, there are some random variables X_i which are sequentially dependent on each other, as Figure 4 shows. This means that each of these variables is only dependent on its previous adjacent variable. The values of these random variables are unknown and therefore these variables are called *hidden state variables*. Hidden variables can have one of the possible discrete values or states S_1, S_2, …, S_n. The probability $P(X_i = S_k | X_{i-1} = S_m)$ is the probability that the hidden variable X_i has value S_k and its previous adjacent variable X_{i-1} has value S_m. Such a probability is called the Transition Probability. Also there is another class of random variables O_i, each of which corresponding to one hidden variable, whose values are known and can be either discrete or continuous. They are called *observation variables* (shown in the lower part of Figure 4). The probability $P(O_i | X_i = S_k)$ is the probability that the observation O_i corresponds to the value S_k of the hidden state X_i. In the terminology of HMMs, this type of probability is called the Emission Probability.

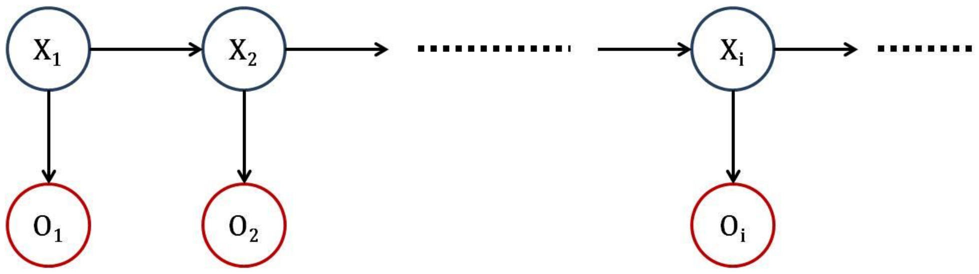


Figure 4 - A diagram showing the dependency relations between variables in a Hidden Markov Model

According to the probability theory, if the values of all observation variables and all possible transition and emission probabilities are known, then the joint probability of occurrence of each possible sequence of values for hidden state variables can be computed as follows:

$$P(X_1, X_2, \dots, X_T, O_1, O_2, \dots, O_T) = P(X_1).P(O_T|X_T).\prod_{i=1}^{T-1} P(X_{i+1}|X_i).P(O_i|X_i) =$$
$$P(X_1).P(O_1|X_1).P(X_2|X_1).P(O_2|X_2) \dots . P(X_T|X_{T-1}).P(O_T|X_T)$$

It is obvious that only the first hidden variable does not have any adjacent variable to its left. Therefore, instead of a transition probability from its previous variable, its own probability $P(X_1)$ is multiplied in the formula. This probability is called the Initial Probability and is only computed for the initial hidden variable.

Now the same method can be used to model the character recognition problem. If a text line image is segmented into T number of character images, then the observations are these character images. The actual character classes corresponding to each of these observations are unknown, and thus the guesses about the identities of these observations are the hidden variables of the model. Transition probability is the probability of the adjacency of two characters. Therefore, if we have N number of character classes, an NxN matrix can hold all the transition probabilities between characters. The details of computing language model probabilities are explained in Section 7. Emission probability is the probability that an observation (i.e. a segment of the line image) corresponds to a specific character class. It is a measure for the level of similarity between the image and the character class. As we will see in Section 8, a function similar to a Gaussian distribution is used to compute the emission probabilities for different segments of a line and for different character classes. The initial probabilities are also computed by counting the number of occurrences of each character in the language model. When all these probabilities are available, the probability that a set of character images corresponds to a set of character classes can be computed using the formula above for all possible sequences of characters. Therefore, it is possible to find the best sequence of characters which maximizes the probability $P(X_1, X_2, \dots, X_T, O_1, O_2, \dots, O_T)$. This process of finding the best character sequence for a line is performed in an algorithm called Viterbi. In this way, the Hidden Markov Models enable us to combine computer vision techniques (emission probabilities) and language model (transition and initial probabilities) in order to solve the character recognition problem.

As mentioned in previous section, in Persian OCR it may be better to combine the text line segmentation process and the character recognition process. Therefore, we use a modified version of HMM called Generalized Hidden Markov Models (gHMM). The main difference from the classic HMM is that the generalized Viterbi algorithm not only iterates over different combinations of characters to find the character sequence with maximum probability, but it also examines different possible segmentation states in order to find the best segmentation scheme. In better words, for each segmentation state, the best character sequence is computed using classic HMM, and the best character sequences of different segmentation states are compared. The details of implementing this method are explained in Section 8.

5. Preprocessing on the Scanned Documents

As mentioned in previous section, the images of the text lines of the target document are the inputs of the recognition process. However, the very first input of OCR software is the images of target document pages. These page images mostly have different types of noises, and sometimes they are skewed. These problems can affect the accuracy of the recognition process and should be addressed. Moreover, images of the text lines should be detected and extracted from the whole page. In this section, the details of the methods used in this project for these types of preprocessing are discussed.

5.1. Noise Removal

Most of the scanned documents, especially older documents and books, include different types of noises. The most prevalent type of noises in document images is the extra black or gray pixels in white background of the pages, or lighter pixels in the black body of the characters. This type of noises is sometimes called salt-and-pepper noises. In the target text used in this project, the presence of salt-and-pepper noises is very little. Most of the noises in this text are located around or on the body of the characters, as some of them are shown in Figure 5 below.

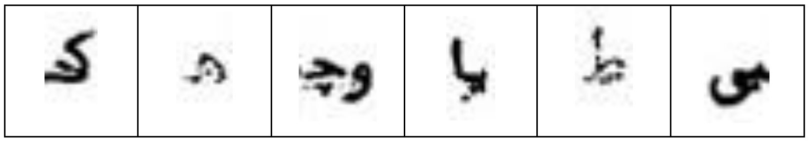


Figure 5 - Some examples of noises in the scanned document used in this project

Removing these types of noises is not easy, mostly because removing them needs some intelligence about the form and structure of characters. In most of OCR software, some types of statistical filters are applied to document images for noise removal. These filters replace the gray-scale value of each pixel with a weighted average of the values of the neighbor pixels. Some examples are Gaussian Filter and Median Filter. Unfortunately, our tests showed that removing noise based on the neighbor pixels is not useful enough for the target text used in this project. Most of these filters smooth the bodies of characters in practice, and sometimes this leads to the removal of the little space between adjacent characters, or the space between the main character body and its dots and diacritics. As a result of these tests, it was decided not to use any explicit method for noise removal.

5.2. Skew Detection

In order to detect and correct the skew of the pages, some measure is needed so that it can be decided whether the page has skew, and if yes, how many degrees. One clue which is widely used for skew detection is the fact that in most writings, the deskewed state of the pages has the most number of horizontal white lines. Therefore, rotating the input image of the page in different angles and counting the number of horizontal pixel lines with a high percentage of white pixels for each angle can be the skew detection method.

However, in some writings the mentioned trick may not be enough for skew detection. In this project, the product of number of white lines and one other measured parameter is used for this purpose. This second parameter is the average of the distance between the horizontal projections of the bounding boxes of the detected lines and the horizontal projections of the training line bounding box. This measure is computed during the baseline detection stage, and therefore in our implementation the skew detection and the baseline detection processes are executed together at once. This second parameter can be understood better by reading the next subsection about baseline detection.

In summary, for skew detection firstly the code iterates over different possible skew angles, for example angles between -5.0 and 5.0 degrees, with an increment of 0.5 or 0.25. For each angle, the page image is rotated by the angle, and then the two mentioned parameters are calculated for the rotated page. After analyzing all the possible skew angles, one with the highest product of two parameters is selected as the skew angle. The page image rotated by the result angle is used in the next stages of the OCR process.

5.3. Baseline Detection

Baseline Detection (or Text Line Detection) is the process of detecting the baselines or the bounding boxes of the text lines in the image of a document page. The extracted text line images are used as input to the next stages of the OCR process. The most common method used for baseline detection is using the concept of *horizontal projections* (or *horizontal histograms*), which means the number or percentage of black pixels in each horizontal pixel line. It is obvious that horizontal projections of the text lines in a document have higher values than those of the white space between the text lines, and this observation can be useful for baseline detection.

In our approach to baseline detection, firstly we need a typical pattern of the horizontal projections of the text lines of the document used in the project. This can be reached by computing the average (or more complex statistical measures) of the set of horizontal histograms of some manually-extracted text line images. As a result, we have a trained ordered vector of horizontal projections that can be compared with different parts of a page image in order to find the text lines. In each page, a bounding box (whose width is the same as page width, and its height is the same as the height of typical lines of this text) is slid down gradually from top of the page through the bottom. For each part of the page, the horizontal projections of the rows are computed and then a distance function is used to compare this set with the trained set of horizontal projections. Those parts of the page with the minimum distance from the trained set are those which are the best candidates to be the text lines of the page. These candidate parts almost always have better distances than their neighbor segments of the page, and thus finding the local minima among the distance values leads to detecting the text lines. Figure 6 illustrates the effect of executing skew detection and baseline detection programs on part of a page image.

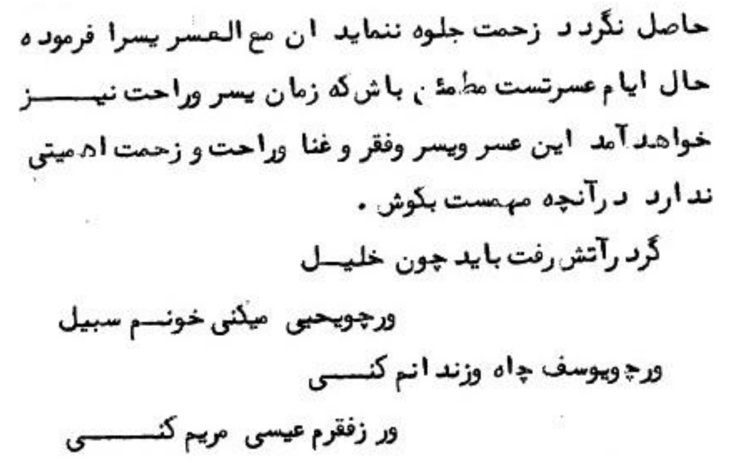
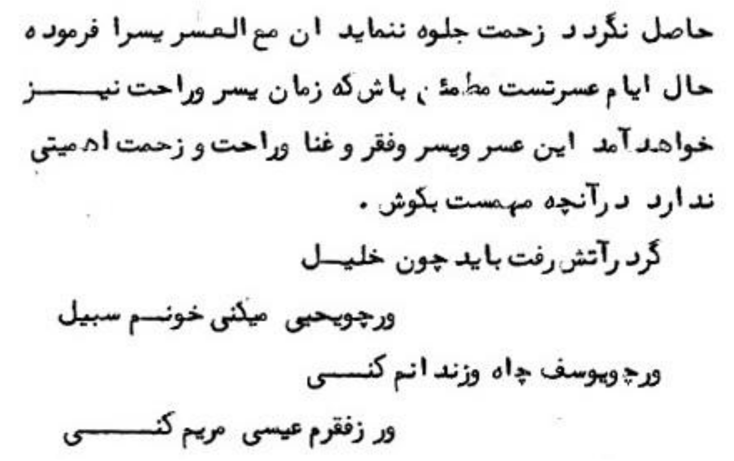
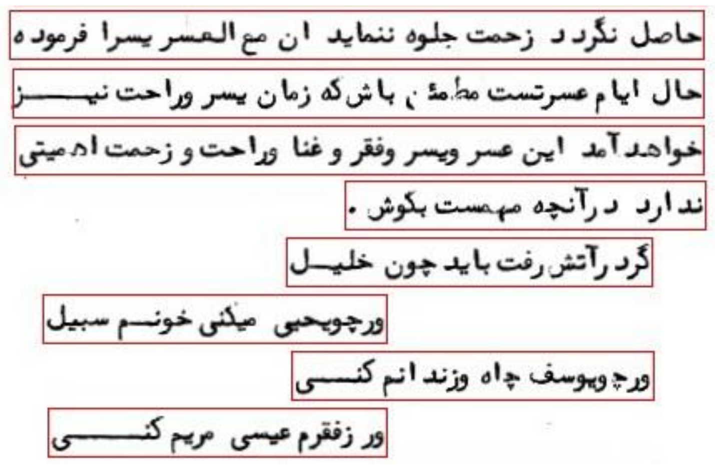
The initial page image, which is skewed:	 حاصل نگرد د زحمت جلوه ننماید ان مع العسر یسرا فرموده حال ایام عسرتست مطمئ ن باش که زمان یسر وراحت نیــــز خواهد آمد این عسر ویسر وفقر و غنا وراحت و زحمت اد میتی ندارد درآنچه مهست بکوش . گرد رآتش رفت باید چون خلیــل ورچویحیی میکنی خونــم سبیل ورچویوسف چاه وزندانم کنــی ور زفقرم عیسی مریم کنــی
The deskewed image of the page:	 حاصل نگرد د زحمت جلوه ننماید ان مع العسر یسرا فرموده حال ایام عسرتست مطمئ ن باش که زمان یسر وراحت نیــــز خواهد آمد این عسر ویسر وفقر و غنا وراحت و زحمت اد میتی ندارد درآنچه مهست بکوش . گرد رآتش رفت باید چون خلیــل ورچویحیی میکنی خونــم سبیل ورچویوسف چاه وزندانم کنــی ور زفقرم عیسی مریم کنــی
The bounding boxes of the text lines shown, as a result of the baseline detection process:	 حاصل نگرد د زحمت جلوه ننماید ان مع العسر یسرا فرموده حال ایام عسرتست مطمئ ن باش که زمان یسر وراحت نیــــز خواهد آمد این عسر ویسر وفقر و غنا وراحت و زحمت اد میتی ندارد درآنچه مهست بکوش . گرد رآتش رفت باید چون خلیــل ورچویحیی میکنی خونــم سبیل ورچویوسف چاه وزندانم کنــی ور زفقرم عیسی مریم کنــی

Figure 6 - The main preprocessings performed on one part of the image of a page

Testing this method on more than 50 pages of the target document, more than 97% of the text lines are detected correctly. Some very short text lines (those including a few characters or at most one or two words) are not detected correctly. This is because sometimes histogram pattern of very short lines does not conform well to the histogram pattern of the training text lines. For example if the phrase “عع” (which is very common in our target text) is the only word in a text line, it is highly probable that it is not detected by our baseline detector, because more black pixels are present in the upper part of this phrase than in the center rows. Another inefficiency of this method is that in the current implementation, the height of the text lines is regarded a fixed value, which is the height of the sliding bounding box. If a text includes lines with different or unknown font sizes, the current method may fail in detecting the text lines correctly. However, because all the pages of the document used in this project have been written with a single font size, we decided not to support multiple font sizes in the current implementation. Using a variation of the current method or applying the thresholding method used by [6] for baseline detection can solve this problem.

Finally, adequately large white spaces within and around (left and right of) the line images are removed, because pure line images with the least quantity of white spaces are needed for the next phases. Figure 7 shows a text line before and after the elimination of white spaces:

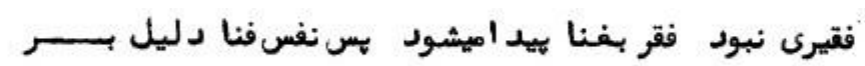
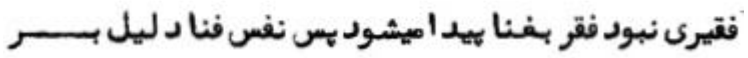
The initial state of the line image:	 فقیری نبود فقر بفنا پیدا میشود پس نفس فنا دلیل بسـر
The line image after eliminating the extra white spaces:	 فقیری نبود فقر بفنا پیدا میشود پس نفس فنا دلیل بسـر

Figure 7 - A line image before and after the elimination of extra white spaces

6. Features Extraction

As mentioned in previous sections, emission probability is computed by measuring the similarity between a new character image and a character class. In order to perform this computation, the most important features of the character image should be extracted and then these features are compared with the important features extracted from the training character images of the character class. Therefore, *features extraction* is an essential stage both in the training phase and in the final text recognition process. The set of values of the features extracted from each character image is called the feature vector. In this section, some details of the features extraction process are explained, as well as the types of features used to represent character images in this project.

A good type of feature is one that can help to distinguish better between different characters, and to determine better the similarities of images of the same character class. Usually two different types of features are used in OCR projects: *structural features* and *statistical features*. Structural features are those features which are related to the structure and form of the characters and their main body and strokes. For example, features about the connected components of a character, or number of branches or dots or crosses in the body of the character are among the structural features. On the other hand, statistical features are computed by gathering different statistical information from the image of the character, rather than the structure or form of the character body itself. Examples are the centroid (i.e. center of mass) of black pixels in a character image, or the density of black pixels in different regions of the image. Structural features are regarded better in differentiating similar characters, which are numerous in Persian alphabet [7] [8]. However, extracting such features correctly in the images is sometimes a complex problem. For instance, in the target document of this project, the bodies of characters are mostly noisy, and thus structural features such as branches or crosses or loops may not be extracted correctly. Moreover, sometimes the dots or inherent diacritics of characters are connected to the main body because of noise, and this can prevent successful extraction of features related to dots or other components of characters.

Because of these complexities, we decided to use mostly the statistical features in the current version of the OCR program. One good semi-structural type of feature is used as well, which is called centroid-to-boundary distance [6]. Computing statistical features for smaller regions in the character image (in addition to the whole image) is a good practice, because it can better reveal the differences in details of the character images. In our project, each character image is resized to a fixed size before extracting the features, so that the features values of the character images with different sizes are measured based on the same frame and thus can be compared with each other correctly. Also, each character image is binarized before features extraction, meaning that the gray-scale values of the pixels are converted to either white or black, based on a binarization threshold.

A summary of the main types of features used in this project is given here:

- **Centroid (or Center of Mass):** means the location (x and y) of the center of mass of the black pixels in a character image, or in one of its regions.
- **Number or percentage of black pixels:** that is, the percentage of black pixels in different regions of the image.

- **Number of rows and columns with black pixels:** means number or percentage of rows and columns in the image or its regions that contain at least one black pixel.
- **Location of extremum black pixels:** is the location (x or y or both of them) of the rightmost, leftmost, uppermost and lowermost black pixel in the character image or one of the smaller regions of the image. Usually the x-coordinate of the rightmost and leftmost black pixels, and the y-coordinate of the uppermost and lowermost black pixels are the most important values in this type of feature.
- **Centroid-to-boundary distance:** means the distance of the character centroid point from the outer border of the character body, in different angles (e.g. from 0 to 360 degrees, with an increment of 10 or 15 degrees). In order to compute this distance for an angle θ, we should start from the borders of the image in that direction, and then check the pixels toward the centroid point to see where is the first black pixel (which would be the outmost black pixel of the character itself in the direction θ). If d is the distance of the centroid from the border of the image in direction θ, the coordinates of the first pixel to be analyzed are computed as below:

$$\begin{cases} p_x = c_x + (d.\cos\theta) \\ p_y = c_y + (d.\sin\theta) \end{cases}$$

  Then by decrementing d one by one, we move toward the outer border of the character in the direction θ. Figure 8 shows the concept of centroid-to-boundary distances for a character.

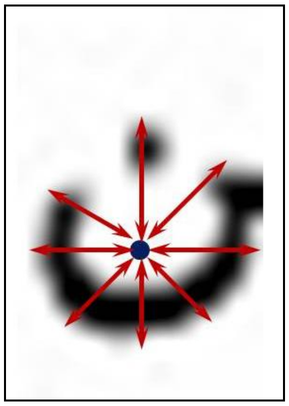


Figure 8 - Centroid-to-boundary distances shown in different directions for the character NOON.

At first, we intended to use Hough Line and Hough Circle transforms in order to include some features about the lines and curves in the body of characters. However, because the implementation used was not accurate enough in detecting lines and curves in a suitable way, it was decided to discard these types of features. The number of regions used and the parameters related to each type of features were determined in cross validation tests (refer to Section 10). The best accuracy was achieved by dividing the character image to 6 number of regions (3 horizontal and 2 vertical divisions). In this state of parameters, totally 106 number of features form each feature vector.

7. Training Process

Training is the process of preparing (or training) the parameters and values needed by the recognition process for analyzing and recognizing the text in new document pages. Computing the transition and initial probabilities from the language model, and aggregation of the features of training character images of various character classes are among the tasks performed in this stage, which are done only once during the development of the OCR software before it can be used for recognizing the text in new documents and pages.

7.1. Preparing the Training Character Images

In order to capture the characteristics of different character classes, the feature vectors of the training character images of each class should be aggregated to be used during the recognition process for computing the emission probabilities. Collecting the training character images for each character class can be a cumbersome task. One way is to manually crop out the character images from the line images of the training pages. Each character image should be labeled properly based on the name of its corresponding character class, so that later in the training process the program can load and extract features from character images of each class.

This manual process can take a long time, and therefore we used another semi-automatic method for extracting and labeling the training character images. Firstly, the images of the text lines of the pages used for training are extracted and written to disk by the preprocessing part of the program. Then in each of the line images, we manually mark the boundary between every two characters by vertical black lines with one pixel width. This is done by the means of an image editing software. An example of these marked lines can be seen in the Figure 9 below. On the other hand, the text file containing the text of the training pages should be prepared (Refer to Section 7.3 about this preparation of the text file). At the end of this manual process, a small software program is implemented to extract the character images from the marked lines (by detecting the places of vertical black lines), and to save each character image to disk with the appropriate label. The character class corresponding to each character image is known by reading the type-written text line corresponding to each marked line image. The inclusion of combinatorial characters in the list of character classes makes the implementation of this code a little complex. In Figure 9, notice some of these combinations of characters which are considered as one single character. At the end of this process, the training character images of all the classes are written to disk, and are ready to be used as input for the next stage of the training process (as explained in the next section).

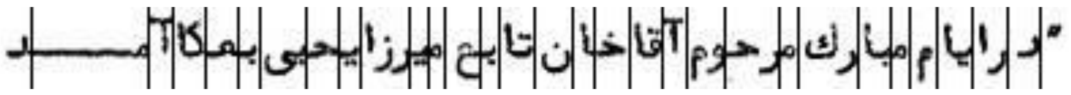


Figure 9 - The boundaries between characters are marked, for the later automatic extraction of training characters

7.2. Computing Fundamental Parameters of Emission Probabilities

As mentioned earlier, some parameters are needed to hold the most important characteristics of each character class, so that they can be used in the recognition process for computing the emission probabilities. These representative characteristics are computed by statistically aggregating the feature vectors of the training character images. The type of parameters required to compute emission probabilities is determined by the function used for computing emission probabilities during the recognition process. As it will be explained in Section 8, the function used in this project for this purpose is Modified Quadratic Distance Function. This function requires the mean feature vector, the eigenvectors matrix, and the eigenvalues matrix of each character class, so that it can compare the feature vector of the new character images with all the character classes. Therefore, these three vectors and matrices should be computed for each character class in the training process. The mean vector is simply the average of the feature vectors of the training characters, and thus its size is the same as the number of features used. The eigenvectors and eigenvalues matrices capture more complex statistical information about the features of the training character images. If D is the number of features, both of these matrices are of size DxD.

The process of computing these parameters for all character classes is rather straightforward. For each class, its training character images are loaded, the features of each image are extracted, and the emission parameters for all the training images are computed. These computations are done by using the statistical functions of the computer vision library used in this project (Refer to Section 9 for more information). Finally, the computed parameters for all character classes are saved to disk, so that they can be reloaded and used in the recognition process.

7.3. Computing Language Model Probabilities

Language model probabilities include the probabilities which reflect some of the properties of the language of the text under analysis. Although the general language properties of the texts written in the same language are almost the same, each book or document can have its own specific language properties, especially when analyzing older texts in which the types of grammar rules or vocabularies used may be different from newer writings in that language. In this project, transition probabilities and initial probabilities are among the language model parameters used to improve the accuracy of the final recognition process.

7.3.1. Transition Probabilities

Transition probability is a measure of the frequency of the adjacency of two characters in the language or the document under analysis. Therefore, the transition probability of two character classes C_1 and C_2 can be computed by the following simple formula:

$$P_t(C_1, C_2) = \frac{n(C_1, C_2)}{N(C_1)}$$

In this formula, $n(C_1, C_2)$ is the number of times C_1 is followed by C_2, and $N(C_1)$ is the total number of occurrences of C_1 in the training text. For storing the transition probabilities, an MxM matrix is used, where M is the number of character classes. The training text file of the target document or book is read by the software program, and for each pair of consecutive characters i and j in the text, the element $[i, j]$ of the transition matrix is incremented by one. Finally, the elements of each row of the matrix are divided by the total number of occurrences of the character class corresponding to that row.

One problem is that the adjacency of many pairs of characters may not have occurred in the training text file, and thus their transition probability would be zero, which is a very strict and inflexible decision. In order to smooth these transition probabilities, we use *Laplacian Smoothing* by using the smoothing parameter k in the following way:

$$P_t(C_1, C_2) = \frac{n(C_1, C_2) + k}{N(C_1) + (k . M)}$$

where M is the number of character classes.

About the typewritten text of the target document, it should be stated that, the training text were typed again properly in order to be used in the training process, although a typewritten text file for this document was available. When a character is typed in Persian in a word processor environment, unfortunately different forms of a character are all written by the same Unicode value to the file. As a result, when the code for computing language model probabilities or the program extracting training character images from marked lines wants to read the text file, they cannot distinguish between different forms of the same character. Therefore, there was a need to retype the text of the target document by using the Unicode values of the characters (instead of typing straightforwardly by the keyboard), so that the program will be able to regard each form of a character as a separate class.

7.3.2. Initial Probabilities

As explained earlier in this paper, initial probability measures the frequency of the occurrence of a character in the text. However, because this probability is only used for the first character of the line, only when a character is at the beginning of a word it should be counted for the computation of initial probabilities. If M is the number of character classes, a vector with size M can hold the initial probabilities for all characters. If $f(C_1)$ is the number of times the character C_1 is located at the beginning of a word, the initial probability for C_1 can be computed by this simple formula:

$$P_i(C_1) = \frac{f(C_1)}{N(C_1)}$$

As we will see in Section 8, the length (i.e. width) of a character image is an effective parameter in gHMM for determining the way text line is segmented. In this regard, the length chosen for the first character of the line in the segmentation is very important, because Viterbi algorithm starts computing probabilities from the beginning of the line, and a correct choice for the length of the first character can weaken the probabilities of incorrect segmentations. Almost all of the characters appear in their most likely length when they are located at the beginning of the line in the target document of this project. We can take advantage of these facts and compute the initial probabilities based on the length of the character images. If we enter the length parameter to the formula above, the initial probability of the character C_1 if its length is l can be computed as below:

$$P_i(C_1, l) = \frac{f(C_1, l) + k}{N(C_1) + (k \cdot L)}$$

where $f(C_1, l)$ is the number of times that C_1 is first of word and its image is of length l, and L is the number of different possible lengths. Because characters in the target text of this document can appear in many different lengths (especially when considering combinatorial character classes and the stretched forms of characters), it was decided to use a smaller number of length categories, each of which covers 4 or 5 possible lengths. Therefore, if number of length categories is L (which is 6 in our project), now the initial probabilities are stored in an MxL matrix.

With this new change in the definition of initial probability, now this probability is both related to the language model and the properties of character images. This makes the process of computing initial probabilities a little complex, because the character image corresponding to each first-of-word character in the training text should be found in order to measure its width. Because of this complexity, we decided to compute basic probabilities during the process of extracting character images from marked lines (as explained in Section 7.1), as this process analyzes both the character images and the training text file.

8. Recognition Process

The recognition of the character sequence corresponding to a text line image is the main functionality of an OCR software program. The main inputs to the recognition process are the text line images which are the result of the preprocessing stage performed on the scanned pages which are entered by the user to be recognized. Also the language model probabilities and the fundamental parameters of emission probabilities, which are the results of the training process, are available to be used in the recognition process. In this stage, for each line image, the generalized version of the Viterbi algorithm is executed, so that the best segmentation scheme and the best character sequence are recognized simultaneously by the algorithm. As mentioned earlier in this paper, in the heart of the Viterbi algorithm a function called a *distance function* is used to compute the emission probabilities of each character image in the line and each of the character classes. In this section, the details of the Viterbi algorithm and the distance function used are explained, as well as other related concepts.

8.1. Distance Function

A distance function helps to compute the level of closeness between an image and a character class. The less this distance is, the more similar is the image to the character class. Therefore, the result of the distance function is somehow proportional to the negative of the emission probability. While it is desired that emission probability is maximized, smaller distance values show higher similarity between the image and the model of the character.

The simplest of the distance functions is the *Euclidean distance*, which simply computes the sum of the squared differences of the features of the new character image and the mean vector of the character class. This type of distance functions ignores the variances of the features and the relationship of different features with each other. A more accurate distance function should address the difference in the level of significance of different features in each character class. In better words, in the distance function of each character class, the features which are more representative of the characteristics of the class should have higher weights than less important features. The *covariance* of the features is a measure which can capture these complexities. Covariance matrix not only includes the variances of the values of each feature, but also covers the way different features vary with relation to each other. The *Mahalanobis distance* function is one which takes advantage of the covariance matrix for achieving higher accuracy. The formula of this distance function is as follows:

$$Distance = \sqrt{(x-\mu)^T \Sigma^{-1} (x-\mu)}$$

where Σ^{-1} is the inverse of the covariance matrix of the character class, μ is the mean vector of the character class, and x is the feature vector of the new character image. If D denotes the number of features, the covariance matrix is of size DxD. The diagonal elements of the covariance matrix reflect the variances of the values of each feature, while other elements represent the covariance of the features with relation to each other. It is interesting that the Mahalanobis distance can be derived from the negative logarithm of the emission probability computed by the multivariate Gaussian distribution, as the formula of this distribution suggests:

$$P(x|C_i) = (2\pi)^{-\frac{D}{2}} . |\Sigma_i|^{-\frac{1}{2}} . \exp\left(-\frac{(x-\mu)^T \Sigma_i^{-1} (x-\mu)}{2}\right)$$

One common problem that may occur when computing the Mahalanobis distance (or the related probability computed by the Gaussian distribution) is that the covariance matrix is not always invertible; it occurs when the determinant of the covariance matrix is zero or is a very small value. In order to solve this problem, one way is to replace the inverse of the covariance matrix with the equivalent *eigendecomposition* matrices [5]. For a symmetric square matrix like the covariance matrix, the following equation holds:

$$\Sigma = B\Lambda B^{-1}$$

The whole term on the right side of this equation is called the Eigendecomposition of the matrix Σ. And then by taking the inverse from both sides, the following equation can be derived:

$$\Sigma^{-1} = B\Lambda^{-1}B^{-1}$$

In this equation, Σ is a DxD symmetric matrix (e.g. covariance matrix), B is the DxD eigenvectors matrix, and Λ is the DxD diagonal eigenvalues matrix. The ith element on the diagonal of the eigenvalues matrix corresponds to the ith row of the eigenvectors matrix (i.e. the ith eigenvector). The eigendecomposition matrices include the directions in which the data has the highest variance, and these new directions are sorted in decreasing order by their power. Therefore, the more powerful and important directions of data are collected in the lower indexes of the eigenvectors and eigenvalues matrices. Some of the advantages of using the eigendecomposition of the covariance matrix are as follows:

- Firstly, the eigendecomposition term, which replaces the covariance inverse, solves the possible non-invertiblity of the covariance matrix.
- Moreover, the determinant of the covariance matrix (which is also a separate term in the Gaussian distribution formula) can now be computed just by computing the multiplication of the eigenvalues (which is the determinant of the eigenvalues matrix too).
- Also the inverse of the eigenvectors and eigenvalues matrices can be computed indirectly to avoid non-invertiblity problems. Because eigenvectors matrix has the *orthonormal* property, its inverse is the same as its transpose. Also the inverse of the eigenvalues matrix can be computed by inversing the values on the diagonal, because the eigenvalues matrix is a diagonal matrix.

As a result of replacing the covariance matrix by the eigendecomposition term, the emission probability computed by the Gaussian distribution can be computed this way:

$$P(x|C_i) = (2\pi)^{-\frac{D}{2}} . |\Lambda|^{-\frac{1}{2}} . \exp\left(-\frac{(\mathrm{x}-\mu)^{\mathrm{T}}\, \mathrm{B}\Lambda^{-1}\mathrm{B}^{\mathrm{T}}\, (\mathrm{x}-\mu)}{2}\right)$$

However, for the emission probability it is preferred to use the distance form. Because in the Gaussian distribution, several values (most of them very small) are multiplied by each other, sometimes the values of emission probabilities may underflow. Therefore, we can add up the negative logarithm of the constituent terms of the Gaussian distribution in order to avoid underflow. The result can be called the *emission distance*, and this new form of the function is called Modified Quadratic Distance Function (or MQDF):

$$D_E = -\log P(x|C_i) = \frac{D}{2}(\log 2\pi) + \frac{1}{2}\left(\sum_{j=1}^{D} \log \lambda_j\right) + \frac{1}{2}\left((x-\mu)^T (B\Lambda^{-1}B^T)(x-\mu)\right)$$

where D_E is the emission distance and λ_j is the jth eigenvalue. Note that the determinant of the covariance matrix can also be computed as the addition of the logarithms of the eigenvalues. As we will see in Section 8.3, also when computing the total probability of a character sequence in the Viterbi algorithm, negative logarithm of the probabilities are added instead of the multiplication of probabilities, in order to avoid underflow.

8.2. The Challenge of Smoothing Eigenvalues

As mentioned in the previous section, the determinant of the covariance matrix can be computed by multiplying all the eigenvalues (or equivalently by adding the logarithms of the eigenvalues in the distance form). Unfortunately, if some of the eigenvalues are zero or negative, the determinant becomes zero (or infinity in the distance form) and this will affect not only the computed emission probability, but all the multiplications (or additions) to find the best character sequence in the Viterbi algorithm. Also very small eigenvalues can be the cause of noise in the results of the recognition process, either in the determinant or in the eigendecomposition term of the distance function. A segment of a typical eigenvalues matrix (which is for the isolated form of the character SEH (ث)) is shown in Figure 10.

$$\begin{bmatrix}
2.132 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 1.457 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0.553 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0.148 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0.032 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 9.21e^{-17} & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 6.69e^{-17} & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 3.28e^{-17} & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.36e^{-17} & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{bmatrix}$$

Figure 10 - The initial eigenvalues matrix computed from the training characters of the class isolated SEH

In order to solve this problem, the general solution is to replace minor eigenvalues of different classes with a constant small value δ [5]. Because eigenvalues are sorted decreasingly based on the importance and power, this replacement solution means replacing the eigenvalues from the kth element toward the end of the eigenvalues matrix. This general solution is called *eigenvalues smoothing*.

In order to implement the eigenvalues smoothing, one method we tested was to replace the eigenvalues which are smaller than a threshold α in all the classes. In this way, in different classes different number of eigenvalues are replaced by δ. The initial recognition results of using this method were not satisfying. The problem is different number of minor (i.e. very small) eigenvalues in different classes. The tests showed that stronger character classes (i.e. those with larger number of training examples) usually have less minor eigenvalues that should be replaced than weaker classes (i.e. those with fewer number of training character images). This is mostly because of the larger variances when a wider range of training examples are available for a class. In some of the classes with a very few number of training examples, nearly all of the eigenvalues should be replaced. Because of this fact, there will be a huge divide between the values of the determinant in weak classes and strong classes. In weaker classes, a higher power of the replacement value δ will constitute the determinant and this leads to a much lower determinant in comparison with the determinant of the stronger classes. The lower determinant abnormally increases the emission probabilities of weaker classes and unfairly increases the possibility of choosing such classes by the Viterbi algorithm in the recognition process.

An ideal solution for the challenge of smoothing eigenvalues is to use different thresholds α and replacement values δ for different classes. However, these suitable values of α and δ can just be computed by performing a large number of cross-validation tests (for different possible values of α and δ for different classes) which seems not feasible. Therefore, at least some of these smoothing parameters (i.e. the threshold α, the starting index k, or the replacement value δ) should be set the same for all classes.

The method we finally used successfully is to make the value of k and δ the same for all character classes. This method ensures more stability than the previous method used, because the eigenvalues of all the classes are smoothed from the same index toward the end of the matrix. Despite this advantage, for the classes which the number of their meaningful eigenvalues (i.e. large enough eigenvalues) is lower than the index k, the decreasing order of the eigenvalues may become disorganized. Therefore, for the eigenvalues of some classes, from where in the matrix that eigenvalue is lower than δ to the element k should also be replaced so that the decreasing order of eigenvalues is kept intact. In Figure 11 below, the first segment of the eigenvalues matrix of the class isolated SEH (ث) is shown after replacing minor eigenvalues when $k = 10$ and $\delta = 0.001$. Although k is 10, only the first five eigenvalues remain intact in order to keep the decreasing order.

$$\begin{bmatrix}
2.132 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 1.457 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0.553 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0.148 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0.032 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0.001 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0.001 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.001 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.001 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.001 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.001 & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{bmatrix}$$

Figure 11 - Performing eigenvalues smoothing for the class isolated SEH, when k=10 and delta=0.001

The suitable value for k and δ should be determined by cross validation tests. Our tests show that when there is some number of classes with very few training examples and thus a few numbers of meaningful eigenvalues, smaller values for k are more suitable. Small values for k mean that many of the meaningful eigenvalues in stronger classes should be replaced by the constant δ. As a result, the recognition power of the model of stronger classes may decrease to some level. However, when all of the classes have an adequate minimum of training examples, then larger values for k can be chosen, and most of the meaningful eigenvalues in different classes will remain intact. This usually results in forming better emission models for character classes.

8.3. The Generalized Viterbi Algorithm

As explained earlier in this document, the Viterbi algorithm is the heart of the character recognition process, because it computes the character sequence with maximum probability of being the true guess for the input text line image. Because in this project no separate character segmentation process is used, a generalized version of the Viterbi algorithm is applied so that both the best segmentation scheme and the best character sequence are determined simultaneously.

8.3.1. How Viterbi Algorithm Works

In the classic version of the Viterbi algorithm, an outer loop iterates over the segmented character images (i.e. the observations), and an inner loop spans all the character classes. In the generalized Viterbi algorithm, no set of segmented character images are available, therefore the outer loop should iterate over different possible segmentations. Neither the number of the characters of the line nor the position and length of each character are known. In order to parameterize a character image in a segmentation state, two variables l and t are used. l is the length (i.e. width) of each character, while t is the ending position of the character in the line image. By iterating over all possible values of t and l, all the possible character images in different segmentation states are covered. Therefore, in the generalized Viterbi, two other variables are also changed in addition to the character class, and this forms three nested loops in the body of the Viterbi algorithm. While the minimum and maximum values of the variable t depend on the width of the text line image, the minimum and maximum values of l are determined based on the minimum and maximum possible lengths of a character image (which can be decided based on the training character images). Figure 12 shows the way the implemented generalized Viterbi algorithm has segmented an input line, and the best character sequence recognized by the algorithm.

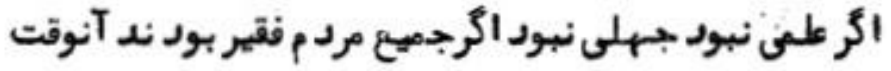
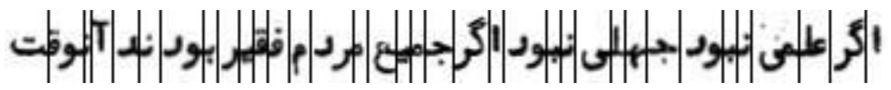
The input line image:	
The way the Viterbi algorithm has segmented the text line into character images:	
The character recognition result of the program:	اگرطص نبودجهئی نبوداگرجمیع مردمفقیربودندآنوقت

Figure 12 - The segmentation and recognition result of the OCR program for an input line image

As mentioned earlier, the Viterbi algorithm attempts to find the character sequence with maximum joint probability (or with minimum total distance). For this purpose, in the iterative implementation of the generalized Viterbi algorithm, two main three-dimensional tables are used to store the important information needed: the probabilities (or distances) table and the paths table. Each element $D[t][l][c]$ in the probabilities table should store the joint probability of the best character sequence which is finished in position t of the line image, and the final character of this sequence is the character c with length l. Also each element $P[t][l][c]$ in the paths table stores the information of the previous character in the best character sequence which is finished in position t with character c as the last character of the sequence and l as its length. The paths table is mainly used at the end of the Viterbi algorithm in order to rebuild the best character sequence corresponding to the entire line. By starting from the first elements of the probabilities table and iteratively filling the next elements by using the information of the previously-filled elements, the iterative Viterbi algorithm finds the best character sequence.

In the first step, the probabilities of all the possible first characters of the line are computed and stored in the probabilities table. It means for each possible length l which is equal to t (meaning it is the first character of the line) and for each character class c, the following probability is computed and stored in the probabilities table:

$$D[t][l][c] = \pi(c,l)\ .e(c,l,t)$$

where $\pi(c,l)$ is the initial probability of character c with length l, and $e(c,l,t)$ is the emission probability for the similarity between character c and the character image with length l ending in position t of the line image. Note that actually the length category is important in initial probability rather than the length itself.

After this first initialization step, t is incremented one by one in the outer loop and the following computations are done for different possible lengths (those which are not equal to t) and different character classes:

$$D[t][l][c] = e(c,l,t)\ .\max_{b,k}\big(D[t-l][k][b]\ .t(b,c)\big)$$

$$P[t][l][c] = \operatorname*{argmax}_{b,k}(D[t-l][k][b]\ .t(b,c))$$

$t(b,c)$ is the transition probability between the characters b and c. Note that $D[t-l][k][b]$ contains the best probability for the character b with length k before the current character. When at the end of the Viterbi algorithm, the probabilities for the highest possible position t (which is the ending position of the line image) are computed and stored, the best character sequence (and the corresponding segmentation scheme) can be retrieved by going back in the paths table and determining the best character and length at each step until the best first character of the line is reached.

As mentioned in Section 8.1, in order to avoid underflows arisen from large number of multiplications of small values, it is better to use distances instead of probabilities in the Viterbi algorithm. As a result, the character sequence with minimum total distance should be chosen as the best one. In order to use distances instead of probabilities, the negative logarithms of the probabilities are used and the formulas above are modified in the following way:

$$\begin{cases} (t = l): & D[t][l][c] = -\log\big(\pi(c,l)\big) + e_L(c,l,t) \\ (t \neq l): & D[t][l][c] = e_L(c,l,t) + \min\limits_{b,k}(D[t-l][k][b] - \log(t(b,c))) \end{cases}$$

In these new formulas, $e_L(c, l, t)$ is the emission distance (i.e. the distance form of the emission probability) and is computed by the MQDF introduced in Section 8.1.

8.3.2. Improving the Performance of the Viterbi Algorithm

The generalized Viterbi algorithm is much slower than the classic version of Viterbi, especially because of iterating over all possible segmentations. One way to normalize its performance is to assume that the unit of length is more than one pixel (e.g. three or four pixels). Each of these units is called a *block*. In this way, far less number of iterations would be used in the algorithm for changing length values, though the accuracy of the algorithm would also be affected (as we will see in Section 10). In our project, a character image can have a minimum length of 4 and a maximum length of 40 pixels. Therefore if a block length of 3 is used, the length of each character image can be at least equal to one block and at most equal to 13 blocks. Now the iteration over all possible segmentations can be interpreted as dividing the line image into maximum possible number of blocks, and then analyzing all the possible ways that series of adjacent blocks can join each other to form character images.

Another bottleneck in the performance of the generalized Viterbi algorithm is the fact that $e_L(c, l, t)$ (i.e. the emission distance of a specific character image and a character class) may be repeated many times during the algorithm, because the character image ending in position t with length l is present in many different segmentation schemes. The total time used for computing emission distances constitutes a large part of the total execution time of the Viterbi algorithm, because each of the emission distance computations includes both extracting features and computing several number of matrix multiplications. Therefore, another way to improve performance is to store the emission distances the first time they are computed, and in the next times read the previously stored values instead of recomputing the same emission distances. This can be achieved by using another three-dimensional table for emission distances. The element $E[t][l][c]$ stores the emission distance of the character class c and the character image starting from position $t - l$ toward the position t in the line image.

We still can have more performance optimizations through the emission distances, by paying attention to those computations in the MQDF that are not dependent on the character image under analysis. In this regard, the eigendecomposition term (i.e. the term $B\Lambda^{-1}B^{-1}$) is only dependent on the character class and can be computed from the eigenvalues and eigenvectors matrices of the class (which were prepared during the training process). Also this computation is a time-consuming task, because it involves two multiplications between three square matrices. Therefore, computation of the eigendecomposition term in the distance function can be performed once before the recognition process (e.g. in the training process itself). Doing this modification was also very helpful in reducing the execution time of the recognition process.

9. About the Implementation Process

In this section, some brief information about the implementation process and the programming languages and libraries used are given. Various parts of the OCR program of this project were all implemented in C++ programming language. Also, the OpenCV library was massively used for the image processing and the matrix computations of the program.

Both the descriptive information (i.e. the ID and Unicode values) and the emission parameters of character classes were stored in YAML files, through the persistence API of the OpenCV library. The tables of language model probabilities were also saved in a separate YAML file during the training process. All the information in these files is loaded at the beginning of the recognition process.

10. Tests and Results

10.1. Initial Tests and Results

For testing the OCR program implemented, firstly the training program should be executed in order to prepare the language model probabilities and the emission parameters needed in the character recognition process. The training text used is the holy Baha'i book *'Maede-ye-Asemani'*, the 5th volume. The scanned version of the book was available, but the appropriate text version was not available, and therefore the needed pages of the book were typed (because of the considerations explained in Section 7). About 25 pages of the text were used for the training process. This may not be enough for having strong language model and emission parameters; however, because preparing the training data is a time-consuming task, at this stage we preferred to continue with these 25 training pages. A larger set of training data can be prepared in future in order to achieve better results.

The first task in the training process is to extract the character images from the training pages. For this purpose, firstly the manual process of marking the boundaries between characters in the text lines of the training pages was done, which was a time-consuming task. Then, the small program implemented for extracting the characters from these marked lines was executed. As a result, about 15000 training character images were extracted. As we included about 322 character classes for the initial tests, an average of 46 training characters were prepared for each character class. However, number of training characters of classes is not uniform enough. For example, more than 80 classes have less than 10 training characters, while the number of classes with more than 50 training characters is only 65. In the next step, the language model probabilities and the emission parameters were computed in separate processes.

In order to determine the best values for some important parameters and variables of the system, a set of cross-validation tests were done on the character recognition program. Different parameters were changed during these tests, from the index k and the replacement value δ in eigenvalues smoothing, to the parameters related to feature types and regions. Three other pages of the training text were used (both scanned images of the text and the typed text version) for the cross-validation tests.

In the tests, in order to estimate the accuracy of the results, the character recognitions of the program should be finally compared with the ground truth (that is, the typed text version of the test pages). This evaluation should be done for each text line, and finally the results of all the text lines can be aggregated. One measure suitable for comparing character sequences is the Levenshtein Distance. This measure computes the minimum number of insertions, deletions, or substitutions that can be done to convert the first text sequence into the second one. Therefore, it actually computes the level of error in the results, and should be minimized as possible. The reason we use such a measure instead of simply comparing the corresponding characters, is that the number of characters recognized by the OCR program may not match with the true number of characters of the text line. The percentage of the accuracy can be estimated by dividing the Levenshtein distance value by the true number of characters and subtract the result from 1.

The fact that combinatorial characters were included among the character classes of the system imposes some complexities in computing the Levenshtein distance. For example if the system correctly recognizes two adjacent characters separately, while the combination of these two is also a separate character class, obviously the system should not compute this recognition of the system as an error. In order to circumvent such problems when computing the Levenshtein distance, for each line both the recognized character sequence and the true character sequence are firstly reduced to the Unicode values of the characters, and then the computation of Levenshtein distance is done on the lists of Unicode values (instead of the sequences of character classes). Note that combinatorial characters include more than one Unicode values. In addition to the mentioned trick, one other complexity is related to the different forms of each character. For each character, usually at least two of the forms have a very similar appearance. For example, the initial and the medial forms of the character CHE (چ) are almost the same, while the final and isolated forms are also very similar. Therefore, if the OCR program recognizes one form instead of the other similar form, it may be more logical to consider the recognition as correct. The logic behind including different forms of the characters and the combinatorial characters as independent character classes is that this inclusion can help to better recognize the character sequences, and therefore these are somehow helper classes. Because of this fact, we included some rules of similarity in the test process, so that similar forms of the same character are supposed to be the same when computing the Levenshtein distance.

The best accuracy achieved in the first set of cross-validation tests was 65.6%. This level of accuracy is not acceptable enough; therefore, after analyzing the problems in the recognition results, some new tricks were implemented in order to improve the total accuracy.

10.2. Attempts to Improve the Results

10.2.1. Using the Length Probabilities of the Characters

One fact observable in the initial recognition results was that sometimes a combination of three or four characters were recognized as one single character by the OCR program. Almost none of the training images of the recognized character were such lengthy. Therefore, it was decided to use the effect of character length probabilities in order to reduce the possibility of choosing a character for a line segment whose length is not common for the mentioned character. About the initial probabilities, the lengths of the characters were taken into account. Now, it was decided to include the length probabilities for all the characters in the line (not only the first character of the line).

The length probabilities are computed in the training process in almost the same way as the initial probabilities; the only difference is that for length probabilities no condition on the location of characters in the words or lines is considered. As a result, the length probabilities can be computed by analyzing the length of the training character images. Also the lengths are categorized into six different categories, the same as the categories used for initial probabilities.

Length probabilities are entered to the total distance formula of the Viterbi algorithm, with a similar role to the emission distances and the language model probabilities. If the length probability of a character image is denoted by $w(c, l, t)$, the formula is changed in the following way:

$$D[t][l][c] = e_L(c, l, t) - \log(w(c, l, t)) + \min_{b,k}(D[t-l][k][b] - \log(t(b, c)))$$

10.2.2. Removing Some Constant Values from the Emission Distances

One other observation in the initial results was that some emission distance values are negative. These negative values are not signs of error in the OCR system, and it is normal for the MQDF to output negative values. However, we guessed that these negative values may be sources of some instability in the results. Also we found that some part of the emission distance formula includes a fixed large negative value which is the same for all the character classes in the current implementation, and thus can be removed from the MQDF. One part of this fixed value is $\frac{D}{2}(\log 2\pi)$, which is only dependent on the number of features. Another part is the part of the determinant term which is calculated from the minor eigenvalues (that is, eigenvalues from k to the end of the matrix). This fixed part of the determinant can be mathematically showed as $\frac{1}{2}\sum_{j=k}^{D} \log \lambda_j$ and in practice is equal to $\frac{D-k}{2}(\log \delta)$, which is a large negative value. Therefore, removing these two terms (shown below) can make all the emission distance values positive:

$$\frac{D}{2}(\log 2\pi) + \frac{D-k}{2}(\log \delta)$$

10.2.3. Scaling Down the Emission Distance Values

Another potential source of error was the fact that emission distance values have a very large range of variance (about hundreds or thousands), while the difference between the highest and the lowest language model probabilities is at most 20. Therefore, the effect of emission distances on the final recognition results is far more than the effect of language model probabilities and the newly added length probabilities. This large variance among the emission distances may be mostly the result of the parameters chosen for eigenvalues smoothing in cross-validation tests.

One way to reduce this large dominance of emission distances in the final results is to scale down the emission distance values. Different new intervals should be tested to know how much scaling should be done to achieve the best results. Although this is not a mathematically-proven trick, a considerably large improvement in the results is achieved with this type of change (as we will see in Section 10.3).

10.2.4. Using the Average Distances in Comparisons

In the body of the Viterbi algorithm, the character sequences whose total distances are compared with each other in order to find the minimum distance necessarily does not include the same number of characters. For example, at one point the total distance of a sequence with 20 characters may be compared with the total distance of a sequence with 12 characters. This may not always be a fair comparison, because in this way there will be a general tendency to choose the sequences with less number of characters. The reason is that adding up less number of positive values generally may lead to a lower total distance. It may be more reasonable to compare the average distance of the character sequences. The average distance is computed by dividing the total distance by the number of characters in the sequence.

In order to use average distances in comparisons, the number of characters of each character sequence should be known at each moment. A new three-dimensional table (e.g. called sequence lengths table) can be used to store the number of characters of the best sequences. The element $S[t][l][c]$ of this new table stores the number of characters in the best sequence that ends at position t with a final character image of length l and recognized as character c. Now at each stage of the Viterbi algorithm, the best previous character can be determined by:

$$\underset{b,k}{\operatorname{argmin}}\left(\frac{D[t-l][k][b]-\log(t(b,c))}{S[t-l][k][b]+1}\right)$$

Where c is the current character and l is its length, b and k are respectively the class and the length of the previous character (which should be determined). Also when the best b and k are determined, the number of characters of the best sequence ending with c is stored in the appropriate element of the sequence lengths table:

$$S[t][l][c] = S[t-l][k][b]+1$$

10.3. Final Tests and Results

After implementing each of the explained modifications to the OCR program, some new cross-validation tests were performed. A summary of the best test results of the system after each of these changes is shown in Table 2. As it can be seen, the most significant improvement in the results was achieved by scaling down the emission distance values, while applying length probabilities and using average distances in the comparisons of the Viterbi algorithm both improved the results to some extent. Eliminating the constant values from the emission distances did not show any significant improvement in the accuracy of the program.

Description	Accuracy
Initial tests of the OCR program	65.6%
After including the length probabilities	67.6%
After using the average distance in comparisons of Viterbi	70.9%
After scaling down the emission distance values	76.8%
Final tests with all the improvements including horizontal overlap in regions, and with block length = 3	81.0%
Final tests with all the improvements, and with block length = 2	85.5%

Table 2 - The best results of the OCR program under different conditions

One other modification which helped to improve the results was considering a horizontal overlap between the smaller regions of the image during the feature extraction phase. This technique can help to reduce the effect of minor horizontal translations of the character body in the segmented character images. One other factor which should be mentioned here is the length of the blocks used in the Viterbi algorithm (as explained in Section 8.3.2). If we decrease the block length, naturally the results may improve, because the text line image can be segmented more neatly and precisely. However, this improvement comes with a side effect on the performance and execution speed of the OCR program.

Regarding the performance of the program, the typical average execution time of the program in recognition of a typical text page is about 150 minutes. That is about 7 minutes for a typical text line which has a width of about 380 pixels. Before implementing the performance optimizations explained in Section 8.3.2, the performance was much worse and unpractical, and it took about 100 minutes to recognize a typical text line. Also one other source of slowness in that time was the recursive implementation of the generalized Viterbi algorithm, so we turned to the iterative approach which is much faster. It should be noted that some other factors also affect the execution time of the program, such as the block length, the minimum and maximum possible lengths of a character image, the number of character classes, the number of features, etc.

11. Conclusion

In this project, we implemented an OCR software program which is capable of recognizing the text in the scanned images of the documents written in a specific older Persian font. Hidden

Markov Models (HMM) was used as the mathematical tool for modeling the character recognition problem, mainly because HMM helps to take advantage of both the language model and the emission distances in service of the recognition of characters. Also because Persian writings are cursive and thus the segmentation of text lines into characters is a complicated problem, we preferred to perform the character segmentation and the character recognition processes simultaneously. This combination of segmentation and recognition becomes possible by using the generalized Hidden Markov Models (gHMM).

Language model probabilities and emission probabilities are the most important parameters used by gHMM for recognizing the best segmentation of characters and the best character sequence corresponding to a text line image. Therefore, these parameters should be prepared in the training process, before the OCR program can run. Language model probabilities, including transition and initial probabilities are trained using the appropriately type-written text version of the training document. For preparing the parameters used for computing emission probabilities, firstly the training character images of all the character classes are extracted from the scanned pages of the training document. This extraction of training character images is done by marking the boundaries of characters in line images and then running a simple program to extract and label each character image. After that, the emission parameters of all the character classes are computed and saved by extracting and aggregating the features of the training character images. The features used in this project for representing character images are mostly statistical features, which give information about the way black pixels are distributed in different regions of each character image.

The input to the recognition process is the scanned images of the pages. Therefore, before starting the main recognition process, some preprocessing is done by the program on the page images. The skew of the pages are detected and corrected by testing some criteria on different rotations of each page and finding the best angle. Then, the bounding boxes of the text lines in each page are detected by comparing all the possible horizontal strips in the page with a trained pattern of text lines. These comparisons are done by using the horizontal projections of the text line boxes. The final step in the preprocessing stage is the elimination of white spaces from within and right and left sides of each line image.

In the main character recognition process, the generalized version of the Viterbi algorithm is used to iterate over all possible segmentations and all possible character sequences for each segmentation in order to find the best characters for the line image. For the comparisons of different character sequences in the Viterbi algorithm, a joint probability (or a total distance) formula based on the HMM is used, which combines the emission and the language model probabilities. For computing emission probabilities (or emission distances), a distance function called MQDF is used, which uses the mean vector and the eigenvalues and eigenvectors matrices of a character class in order to compute the closeness of a character image to the class.

After doing a training phase by using 25 training pages, a series of cross-validation tests were done in order to find the best values for the important variables and parameters of the program. The best result achieved in the final tests of the program was 85.5% accuracy in recognizing the true characters. Modifications in the program like using the length probabilities in the Viterbi total distance formula, scaling down the values of the emission distances, and using average distances in comparisons, all helped to improve the accuracy of the implemented OCR program. Also factors such as using a larger body of training data and reducing the size of the block lengths in the Viterbi algorithm are effective on improving the results of the program.

12. Future Work

Some of the steps that can be done to improve the recognition accuracy of the OCR program implemented in this project are explained below. In addition to these potential improvements, for implementing a complete and reliable Persian OCR software application which can be used widely (especially for recognition of historical documents and books), many other complications should be addressed and the program should have many other features. For instance, it will be much better that an OCR application is capable of covering more different font sizes and font families, and it should be able to handle different page styles. Another worthy direction of research in relation to Persian OCR is the recognition of handwritten Persian writings, which introduces new complexities and variations.

- The first way of improving the results of this OCR program is to collect and prepare more training data. As mentioned earlier in this document, the number of training pages used in this project was not enough. Although it is a time-consuming manual task, preparing a much larger body of training text not only makes language model probabilities and emission parameters more precise and stronger, but also helps to increase the stability in the recognition results. All the character classes should have a good minimum number of training character images.
- The features extracted from character images are the most important factors in recognizing the characters. In Persian OCR, using more structural features may improve the discriminative power of the recognition process. Some important structural features are the lines and curves of each character, and also information about the connected

components, dots, links, branches, etc. Thinning the body of the characters and creating a thin skeleton of each character before the feature extraction process can help to increase the power of the structural features used.

- Using the Gaussian Mixtures technique is another way to improve the recognition results. In this method, firstly a clustering technique is applied to the training images of each character, in order to determine how the training feature vectors are distributed. Then for each cluster different emission parameters are computed and stored. In the recognition process, the distance function for measuring the closeness to a character image computes a weighted sum (or a mixture) of the emission parameters of the clusters of the class. This method helps to address the large variances in the training images of the characters, and improves the emission model of each character class.

13. References

[1] A. Amin, “Offline Arabic Character Recognition: the State of the Art”, Pattern Recognition, vol. 31, issue 5, pp. 517-530, March 1998.

[2] L. Lorigo and V. Govindaraju, “Offline Arabic Handwriting Recognition: A Survey”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 28, issue 5, 2006.

[3] M.S. Khorsheed, “Recognizing Handwritten Arabic Manuscripts using a Single Hidden Markov Model”, Pattern Recognition, vol. 24, pp. 2235-2242, 2003.

[4] M. Dehghan, K. Faez, M. Ahmadi, and M. Shridhar, “Handwritten Farsi (Arabic) Word Recognition: A Holistic Approach using Discrete HMM”, Pattern Recognition, vol. 34, pp. 1057-1065, 2001.

[5] F. Hedayati, J. Chong, K. Keutzer, “Recognition of Tibetan Wood Block Prints with Generalized Hidden Markov and Kernelized Modified Quadratic Distance Function”, EECS Department, University of California, Berkeley, November 2010.

[6] M. Laine and O.S. Nevalainen, “A Standalone OCR System for Mobile Camera Phones”, The 17th Annual IEEE International Symposium on Personal, Indoor, and Mobile Radio Communications.

[7] M.S. Khorsheed, W.F. Clocksin, “Structural Features of Cursive Arabic Script”, The 10th British Machine Vision Conference, University of Nothingham, vol. 2, pp. 422-431.

[8] J. Shanbehzadeh, H. Pezashki, A. Sarrafzadeh, “Features Extraction from Farsi Handwritten Letters”, Proceedings of Image and Vision Computing, pp. 35-40, Hamilton, New Zealand, December 2007.