(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) **World Intellectual Property Organization**
International Bureau

PCT

(43) **International Publication Date**
20 November 2008 (20.11.2008)

(10) **International Publication Number**
WO 2008/138356 A2

(51) **International Patent Classification:**
G06K 9/00 (2006.01)

(21) **International Application Number:**
PCT/EG2007/000018

(22) **International Filing Date:** 15 May 2007 (15.05.2007)

(25) **Filing Language:** English

(26) **Publication Language:** English

(71) **Applicant** *(for all designated States except US)*: **THE ENGINEERING COMPANY FOR THE DEVELOPMENT OF COMPUTER SYSTEMS ; (RDI)** [EG/EG]; 23 Umar Ibn AL-Khattab Street, Muhandiseen, Giza (EG).

(71) **Applicants and**
(72) **Inventors: RASHWAN, Mohsen Abdel-Razik Ali** [EG/EG]; 22 Saleh Salama Street, Faisal, Giza (EG). **AHMED, Mohamed Attia Mohamed El-Araby** [EG/EG]; 98th Al - yasmin Street, Al-Ashgar District, 6th of October City, Giza (EG).

(74) **Agent: EL-FAYED, Nasser Ahmed**; Nasser El-Fayed, Lawyer Office, 23 Umar Ibn AL-Khattab Street, Muhandiseen, Giza (EG).

(81) **Designated States** *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) **Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, MT, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Declaration under Rule 4.17:
— *of inventorship (Rule 4.17(iv))*

Published:
— *without international search report and to be republished upon receipt of that report*

(54) **Title:** A LARGE-SCALE ARABIC OMNI FONT-WRITTEN OCR BASED ON AN ANALOGY WITH HMM-BASED ASR

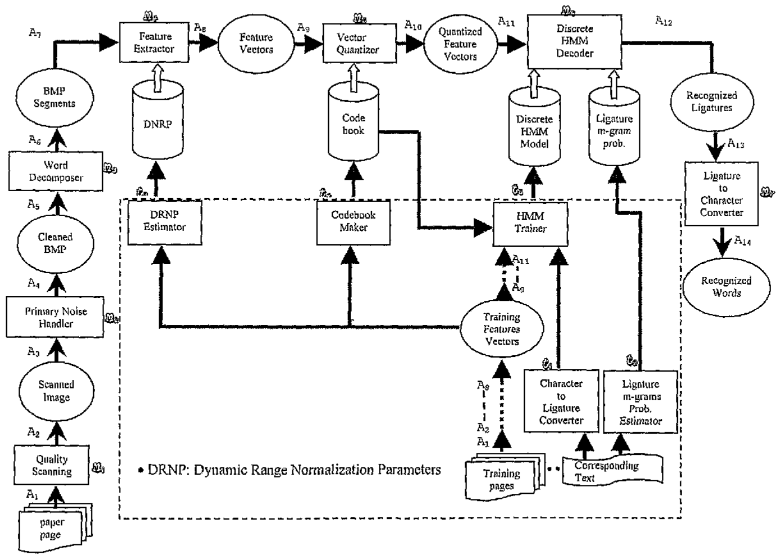


Fig.6

(57) **Abstract:** Research work on Arabic optical text recognition (AOTR), although lagging that of other languages, is becoming more intensive than before and commercial systems for AOTR are becoming available. For such a needy language like Arabic; there is a wide room for enhancement to lower the word error rate of typewritten systems as lower as 4%. So we present an omni font (sizes and styles), unlimited-vocabulary OCR system for Arabic scripts. The system is based on Hidden Markov Models (HMM), an approach that has proven to be very successful in the area of automatic speech recognition. The system uses our powerful, robust Histogram-based algorithm for lines & words decomposition and uses our autonomously time-domain differential features. Also the system gains benefits from a well built bi- gram language models.

WO 2008/138356 A2

A Large-Scale Arabic Omni Font-Written OCR Based on an Analogy with HMM-Based ASR

Technical field:

Converting the Text picture into an editable text file using the optical character recognition.

Background Art:

Optical character recognition (OCR) is the branch of pattern recognition that studies automatic reading. The ultimate goal of OCR is to imitate the human ability to read at a much faster rate by associating symbolic identities with images of characters.

The potential application areas of automatic reading machines are numerous. One of the earliest and most successful applications is sorting checks in banks, as the volume of checks that circulates daily has proven to be too enormous for manual entry. Other applications include reading postal addresses off envelops and automatically sorting mail, helping the blind to read, reading customer-filled forms (tax forms, insurance claims, application forms), automating offices, archiving and retrieving text, and improving human-computer interfaces (e.g., pen-based computers). Among numerous applications of typewritten OCR systems comes Document Management Systems (DMS) as the largest industrial consumer. In such systems scanned images of electronically unavailable documents are archived by the DMS, meanwhile OCR runs on each scanned image. While the images are used for viewing the document, the text resulting from OCRing the images is used for all kinds of Information Retrieval (IR) and Knowledge Management (KM) purposes which are insensitive to the inevitable error rate of the OCR process as long as this rate is kept small enough (< 4% of the word rate is a rational criterion).

Since the mid-1940s researchers have carried out extensive work and published many papers on character recognition. Most of the published work on OCR has been on Latin characters, with work on Japanese and Chinese characters emerging in the mid-1960s.

Although almost a third of a billion people worldwide, in several different languages, use Arabic characters for writing (alongside Arabic, Persian and Urdu are the most noted examples), Arabic character recognition has not been researched as thoroughly as Latin, Japanese, or Chinese. The first published work on Arabic optical text recognition (AOTR) may be traced back to 1975, and was by Nazif. In his master's thesis he developed a system for recognizing printed Arabic characters based on extracting strokes, that he called radicals (20 radicals are used) and their positions. He used correlation between the templates of the radicals and the character image. A segmentation stage was included to segment the cursive text. Years later, Badi and Shimura and Nouh worked on printed Arabic characters and Amin on handwritten Arabic characters. In the 1980s research work on AOTR increased considerably, a trend that is continuing in the 1990s.

During the past decade, Hidden Markov Models (HMMs) have become the predominant approach to automatic speech recognition. The success of HMMs in speech recognition has led many researchers to apply them to character recognition by representing each word image as a sequence of observations. HMMs qualify as a suitable tool for cursive script recognition for a number of reasons. First, they are stochastic models that can cope with noise and pattern variations occurring as in different fonts. Next, the number of tokens representing an unknown input word may be of variable length. This is a fundamental requirement in cursive fonts because the lengths of the individual input words may greatly vary. Moreover, using an HMM-based approach, the segmentation problem, which is extremely difficult and error prone, can be avoided. This means that the features extracted from an input word need not necessarily correspond with letters. Instead, they may be derived from part of one letter, or from several letters. Thus, the operations carried out by an HMM are in some sense holistic, combining local feature interpretation with contextual knowledge. Finally, there are standard algorithms known from the literature for both training and recognition using HMMs. These algorithms are fast and can be implemented with reasonable effort.

There has been little work in the use of HMMs for the recognition of Arabic script. Three references report on the use of HMMs to perform recognition of Arabic script. Allam uses contour tracing to locate groups of connected characters; the recognition is then performed on each such group as a whole, using features extracted from vertical slices. The feature extraction of the papers by Ben Amara and Belaid and by Yarman-Vural and Atici appear to be specific to Arabic and may not be easily generalizable.

In 1999 Issam Bazzi, Richard Schwartz, and John Makhoul present an omnifont, unlimited-vocabulary OCR system for English and Arabic. The system is based on Hidden Markov Models. Their system includes the use of a trigram language model on character sequences. the achieved character error rates of 1.1 percent (about 4.4% WER)on data from the University of Washington English Document Image Database and 3.3 (about 13.2% WER) percent on data from the DARPA Arabic OCR Corpus.

As the Arabic market – esp. in the Gulf countries – is currently a hot one, there is a quite high need for reliable typewritten Arabic OCR engines to be integrated in such DMS systems. Perhaps the - by far - most ready and best equipped system is Automatic Reader© 7.0 provided by Sakhr®. Affording document retrieval, omni Arabic OCR, learning mode, 95% word-level accuracy rate, SDK for integration, and being able to deal with bilingual Arabic-Latin documents, this system is the best-to-choose-now for heavy duty serious applications.

Automatic Reader© 7.0 from Sakhr® is essentially based on a huge set of ad hoc orthographic rules and tips put in a work frame of AI searching techniques to decide on each of the Arabic OCR phases; preprocessing of tiny blocks like dots and diacritics, segmentation, and grapheme recognition. The last phase of synthesizing the recognized text is cleverly guided by Sakhr's Arabic NLP tools for filtering out nonsensical results.

The lag in research on Arabic text recognition, compared to other languages, may be attributed to

(i) The lack of adequate support in terms of journals, books, conferences, and funding, and the lack of interaction between researchers in this field.

(ii) The lack of general supporting utilities like Arabic text databases, dictionaries, programming tools, and supporting staff.

(iii) The late start of Arabic text recognition.

(iv) The special characteristics of Arabic script.

<u>Challenges in Arabic Recognition:</u>

1- **The connectivity challenge**: Whether handwritten or typewritten, Arabic text can only be scripted in connected (or *cursive*) *mode*; i.e. graphemes are connected to one another within the same word with this connection interrupted at few certain characters or at the end of the word. This necessitates any Arabic OCR to do not only the traditional grapheme recognition task, but also another tough grapheme segmentation one (see figure 1). To make things even harder, both of these tasks are mutually dependent and must hence be done simultaneously.

2- **The dotting challenge**: As stated before; dotting is extensively used to differentiate characters sharing similar graphemes. According to figure 2 where some example sets of dotting-differentiated graphemes, it is apparent that the digital differences between the members of the same set are small. Whether the dots are eliminated before the recognition process, or recognition features are extracted from the dotted script, dotting is a significant source of confusion – hence recognition errors – in Arabic typewritten OCR systems especially when run on noisy documents; e.g. those produced by photocopiers. On the contrary, dotting may be helpful for Arabic handwritten OCR systems as dots are usually sensed as separate short strokes.

3- **The multiple grapheme cases challenge**: Due to the mandatory connectivity in Arabic orthography; the same grapheme representing the same character can has multiple variants according to its relative position within the Arabic word segment {Starting, Middle, Ending, Separate} as exemplified by the 4 variants of the Arabic character "Ein" highlighted in red in figure 3.

4- **The ligatures challenge**: To make things even more complex, certain compounds of characters at certain positions of the Arabic word segments are represented by single atomic graphemes called *ligatures*. Ligatures are found in almost all the Arabic fonts, but their number depends on the involvement of the specific font in use. Figure 4 illustrates some ligatures in the famous font "Traditional Arabic" highlighted in red.

Broad graphemes set: Multiple grapheme cases as well as the occurrence of ligatures directly lead to broad grapheme sets; e.g. a common highly involved font like Traditional Arabic contains around 190 graphemes, and another common less involved one (with less ligatures) like Simplified Arabic contains around 95 graphemes. Compare this to English where 40 or 50 graphemes are enough! Again, a broader grapheme set means higher ambiguity, and hence more confusion.

5- **The diacritics challenge**: Unless the reader is knowledgeable enough, each character in Arabic strictly needs one or more diacritical marks to be drawn over or under the corresponding grapheme in order to ensure the intended phonetic transcription and hence the correct pronunciation. Apart from the teaching purposes, Arabic diacritics are used in practice only when they help in resolving linguistic ambiguity of the text. The problem of diacritics with typewritten Arabic OCR is that their direction of flow is vertical while the main writing direction of the body Arabic text is horizontal from right to left. (See figure 5) Like dots; diacritics – when existent - are a source of confusion of typewritten OCR systems especially when run on noisy documents, but due to their relatively larger size they are usually preprocessed.

6- **The variation challenge:** Arabic characters do not have a fixed size (height and width). Size varies across different characters and across the different shapes of the same character

7- **The overlapping challenges:** - Characters in a word may overlap vertically (even without touching) (see Fig. 4).

In our research, we implement an Omni-font-open vocabulary HMM-based typewritten Arabic Text recognition system. We address the issue of how to perform a character recognition system for an independent style (unconstrained style) Arabic typewritten data; this will be done by using a large amount of training data to train the HMMs.

Building an open vocabulary system will derive us to the usage of a character based HMM system not a word based HMM system. Our research on building an independent style-open vocabulary HMM typewritten Arabic recognition system is handled by trying different kinds of HMMs (either discrete or continuous) after choosing the best features that best discriminate between different classes of Arabic characters. Our first system architecture is based on the Discrete HMM.

The basic philosophy of our used autonomously differential recognition feature is as follows; given a sequence of such a feature, one can fully retrieve the written script with only a scaling factor as well as the digital distortion error.

Whatever the features and the approach used for recognition, the language models are essential in recovering strings of words or characters after they have been passed through a noisy channel such as scanning or print degradation. Hence, we in cooperate a bigram character based language models in our system to improve its performance.

Any font-written OCR system needs to perform some processing on the input scanned text block prior to the recognition phase. Decomposing the text block into lines hence into words is perhaps the most fundamental such pre-processing where errors add in the total error rate of the whole OCR system.

Despite being a widely used, simple-to-implement, and computationally efficient, the histogram-based approach may be vulnerable to some idiosyncrasies of real-life documents esp. as per noise and structural textual complexities. The latter of those idiosyncrasies is apparent while dealing with Arabic script. In our work, we elaborated on this approach to overcome its shortcomings esp. - with Arabic script – and came out with a robust and powerful decomposition algorithm we used in our system. This quite robust algorithm were proved with real-life Arabic (or Arabic dominated) documents via extensive experimentation that it is highly efficient algorithm.

Disclosure of invention:

Our system is divided into two subsystems: the trainer subsystem which is used to initialize and train Hidden Markov Models using many scanned bitmaps with their corresponding text files, and the Recognizer subsystem which uses the Viterbi algorithm to recognize input bitmaps to output characters. The block diagram of the trainer subsystem and Recognizer subsystem is shown in Figure 6. The system is built up using the following modules:

I-Quality Scanning Module:

In this stage, optical scanners are used to convert printed documents to computer images. It is common practice to convert the multilevel image into a bi-level image of black and white. Often this process, known as thresholding, is performed on the scanner to save memory space and computational effort.

II-Primary Noise Handler Module:

This aims to remove noise and other uninteresting variations in the image which are often byproducts of image acquisition. Filters can be designed for smoothing, sharpening, thresholding, removing slightly textured or colored background, and contrast adjustment purposes. A method that is often used to reduce noise in two dimensional (off-line) images is to traverse the image, pixel-by-pixel, using a 3 x 3 window and to change the value of a pixel based on the value of its 8-neighbors. In our study, a 5×5 median filter has proven effective for the cleaning of input scanned text images from the *salt & pepper* noise

Median filters are, however, hard deciding and cause considerable erosive distortion to the graphemes of written text which is generally harmful to the recognition performance. So, they should be used only while the lines & words decomposition, whereas other softer deciding filters – like *Gaussian* ones – might be used - if necessary - while the recognition process.

III-Word Decomposer Module:

The word decomposer module is used to segment paper into lines, and then to segment each line into words or sub-words starting from upper right corner of the page. A robust Histogram-based algorithm is used for lines & words decomposition

It done with the following steps:

Step 1: Filtering

The target text rectangle bitmap is passed through a 5 by 5 median filter in order to clean the *salt & pepper* noise.

Step 2: Getting the Darkness & Brightness Centroids

For deciding about the *blankness* of horizontal/vertical pixels-scan-lines in later steps of this algorithm, it is necessary to accurately determine the centroid of darkness c_D , and the centroid of brightness c_B. We apply the famous *K-means* clustering algorithm on the pixels population histogram of the target text image, namely the histogram of their gray values on the standard scale from *0* (darkest)-to-*255* (brightest), to accurately acquire these two important values (see Figure 7)

Step 3: Writing-Lines Decomposition

Getting the vertical range $[y_{i,bottom}, y_{i,top}]$ of each writing line L_i, whose height $h_i = y_{i,top} - y_{i,bottom}$ is accomplished through the two following sub-steps:

3.1- Physical Writing Lines Decomposition. This sub-step is trivially done once we get rationally able to decide about whether a given horizontal scan-line l_j is *blank* or *non-blank*. Intuitively, l_j is blank if the following condition is true;

$$\sum_{ight_i - w_i + 1}^{= x_{Right_i}} (255 - p(k,j)) \le w_i \cdot (255 - c_B) \quad \text{(A)}$$

Where w_i is the effective width of L_i to which l_j belongs. However
We calculate w_i we use the following two sub-steps:

1- **Initial Lines Decomposition:**

Here, initial writing lines L^*_i with vertical ranges $[y^*_{i,bottom}, y^*_{i,top}]$ and heights $h^*_i = y^*_{i,top} - y^*_{i,bottom}$, are computed with the blankness condition modified to:

$$\sum_{k=0}^{k=W-1}(255 - p(k,j)) \le W \cdot (255 - c_B) \qquad \text{(B)}$$

Noting that $w_i \le W$, all the scan-lines decided to be non-blank are truly so, according to (A), See figure 8. While few short scan-lines may be falsely decided blank and have to be reconsidered in the next sub-step

2- **Refined Lines Decomposition:**

For each scan-line l_j judged as blank in the previous sub-step; it gets re-tested in this sub-step with the blankness condition modified to:

$$\sum_{k=x_{Right_{n(j)}}-w_{n(j)}+1}^{k=x_{Right_{n(j)}}}(255 - p(k,j)) \le w_{n(j)} \cdot (255 - c_B) \qquad \textbf{(C)}$$

n(j) is defined as the index of the nearest writing-line, as obtained from the sub-step 3.1.1, to the scan-line l_j. After this second pass of blankness decisions; the writing lines L_i are accordingly re-labeled as shown in figure 9

3.2- Logical Writing-Lines Concatenation. Orthography in general, and Arabic script in specific, is full of dots and diacritics over and under the basic body of grapheme sequences This may, cause the sub-step 3.1 generate false *thin* physical writing-lines which logically belong to thick neighbors To realize that concatenation, a concatenation test is iterating until it states false for all the writing-lines in the target text-image. For a writing-line L_i, the concatenation conditions with its neighbors $L_{i-1;i>0}$ and $L_{i+1;i<i_{max}-1}$ are

$$Concat_{Before} = (\frac{h_i}{h_{i-1}} < \alpha) AND (\frac{y_{i-1,bottom} - y_{i,top}}{h_{i-1}} < \beta) \qquad \textbf{(D)}$$

$$Concat_{After} = (\frac{h_i}{h_{i+1}} < \alpha) AND (\frac{y_{i,bottom} - y_{i+1,top}}{h_{i+1}} < \beta) \qquad \textbf{(E)}$$

If both conditions are FALSE, no concatenation occurs and the test advances to the next writing-line. If either $Concat_{Before}$ or $Concat_{After}$ is TRUE, concatenation happens with the corresponding line, and the test resumes iterating at the first writing-line on top of text-image. If both $Concat_{Before}$ and $Concat_{After}$ are

TRUE, concatenation happens with the closer writing-line, and the test resumes iterating at the first writing-line on top of text-image. As per the nature of Arabic script, our extensive experiments show that, the best values for α and β are 0.35 and 0.30 respectively. Figure 10 shows the resulting logical writing lines.

Step 4: Writing-Lines Grouping

Heights of writing lines are usually considered as a rough - never an accurate - indication of the font size they are written in. So, gathering the writing lines in a given text-image in groups, where the heights of these lines in the same group are of a sufficiently small variance, may be useful for the recognition process that might tune some parameters for each group according to say its average height.

Far more importantly, this grouping is a necessary auxiliary step towards deciding whether a given space between successive separate word segments is an *inter* or *intra* word space. See step 6 below.

We here group the writing-lines so that the standard deviation of the writing-lines heights σ_g in each group and the average of these heights h_g satisfy the condition

$$\sigma_g \leq \gamma \cdot h_g \qquad \text{(F)}$$

In our experiments, γ is tuned to 0.2. The 1st and 2nd writing lines from the top of figure 10 are gathered in one group, while the rest are gathered in another one.

Step 5: Part-of-Words Decomposition

In this step, part-of-word segments on each writing line L_i are separated by vertical spaces among them. This step is trivially done once we get able to decide about whether a given vertical scan-line segment at $x = k$ within the range of L_i is *blank* or *non-blank*. Like formula (A) in step 3.1; such a blankness condition is formulated as:

$$\sum_{j=y_{i,bottom}}^{j=y_{i,top}} (255 - p(k,j)) \leq h_i \cdot (255 - c_B) \qquad \text{(G)}$$

Figure 11 shows the result of decomposing the part-of-word segments in our running example, with each marked by a surrounding rectangle.

Step6: Full Words Concatenation

6.1- Eliminating Rouge Spaces Among Part-of-Words. This problem is in fact a hard one of *outliers detection* as named in the literature of data mining. However, a much simpler and effective approach to solving this problem is presented here based on the heuristics emanating from our rich experience about the nature of Arabic, or Arabic dominated script.

Figure 12 illustrating a histogram of a spaces group where the widths of intra-word spaces, inter-word spaces, and rogue spaces are plotted.

The optimal threshold between the intra-word centroid S_{Intra} and inter-word centroid S_{Inter} is $S_{Intra} + ½ \cdot (S_{Inter} - S_{Intra})$. If symmetry around S_{Inter} is also assumed, the cut-off space width for eliminating the rogue spaces can then be expressed as

$$s_{Cut\text{-}Off} = s_{Inter} + ½ \cdot (s_{Inter} - s_{Intra}) \qquad \textbf{(H)}$$

While the average of all spaces S_{avg} is easily computable, the centroids S_{Intra} and S_{Inter} are unknown. So, we seek an expression of $S_{Cut\text{-}Off}$ in terms of only S_{avg}. Let

$$s_{Intra} = \varepsilon \cdot s_{Inter}; 0 \le \varepsilon < 1 \qquad \textbf{(I)}$$

Substituting from (I) in (H), we get

$$s_{Cut\text{-}Off} = ½ \cdot (3 - \varepsilon) \cdot s_{Inter} \qquad \textbf{(J)}$$

On the other hand, S_{avg} may be expressed as

$$s_{avg} = s_{Intra} \cdot (½ + \delta) + s_{Inter} \cdot (½ - \delta); 0 \le \delta < ½ \qquad \textbf{(K)}$$

Where δ is a weighting preference parameter. Substituting in (K); for S_{Intra} from (I), then for S_{Inter} from (J), we get

$$s_{Cut\text{-}Off} = \frac{3 - \varepsilon}{(1 - 2 \cdot \delta) + \varepsilon \cdot (1 + 2 \cdot \delta)} \cdot s_{avg} \qquad \textbf{(L)}$$

Setting $\varepsilon=0$, $\delta=0$ at the extreme, results in $S_{Cut\text{-}Off} = 3 \cdot S_{avg}$. However, our empirical knowledge of the Arabic script in general realistic conditions states $\varepsilon \in [0.1, 0.15]$, $\delta \in [0.05, 0.1]$ which produces the simple rogue-spaces cut-off formula that

$$s_{Cut\text{-}Off} \in [2.7 \cdot s_{avg}, 3.15 \cdot s_{avg}] \qquad \textbf{(M)}$$

6.2- Clustering Inter-word/Intra-Word Spaces. Once the rogue spaces are eliminated – if any - from a spaces group, the rest of these spaces in the group are sent to the K-means algorithm to cluster them into an intra-word spaces cluster whose centroid is S_{Intra} and another one of inter-word spaces cluster whose centroid is S_{Inter}; $S_{Intra} < S_{Inter}$. Figure 13 shows the result of diagnosing inter-word spaces, hence grouping the part-of-word segments within each word in our running example together in one framing rectangle.

IV-Feature Extractor Module:

The feature-extractor module is used to extract features of the sliding window, which slides through the body of the Arabic word from right to left, and provides a series of feature vectors to the vector quantizer module.

The basic philosophy of our autonomously differential recognition feature is as follows; given a sequence of such a feature, one can fully retrieve the written script with only a scaling factor as well as the digital distortion error. The feature proposed here is a time-domain differential one where the script contour is totally differentiated along the x-direction, which can be expressed by equation [1] as follows:

$$\underline{F}_i = D\left(\left. {}^{d\varsigma}\!/_{dx} \right|_{x=i} \right) = [\underline{S}_1 \cup \underline{S}_2 \cup \ldots \underline{S}_M] = \bigcup_{j=1}^{M} \underline{S}_j \qquad \text{Eq. [1]}$$

Where F_i is the frame feature vector and S_j is the segment feature vector

The words are consisting of number of frames and each frame is consisting of number of segments, and the max number of segments per frame is 4 in Arabic fonts. **(See fig. 14)**

For each segment we have four cases:

1- Segment located in the start frame of the word:

$$S = \left[\frac{\text{CG. of the segment - CG. of the first segment}}{\text{First segment length}}, \frac{\text{segment length}}{\text{First segment length}}, -1, -1 \right]$$

2- Segment located in a frame came after a non empty frame:

$$S = \left[\frac{\text{CG. of the segment - CG. of the previous frame}}{\text{previous frame length}}, \frac{\text{segment length}}{\text{previous frame lengthh}}, 1^{st} \text{ overlapped segment}, \text{last overlapped segment} \right]$$

$\longrightarrow$ **If there is overlapping.**

$$S=\left[\frac{\text{CG. of the segment - CG. of the previous frame}}{\text{previous frame length}},\frac{\text{segment length}}{\text{previous frame lengthh}},-2,-2\right]$$

$\longrightarrow$ **If there is no overlapping.**

3- Segment located in a frame came after an empty frames in the same word:

$$S=\left[\frac{\text{CG. of the segment - CG. of the first previous non empty frame}}{\text{first previous non empty frame length}},\frac{\text{CG. of the segment - CG. of the first previous non empty frame}}{\text{first previous non empty frame length}},-3,-3\right]$$

4- Empty frame:

$$S=[0, 0, 0, 0]$$

- Where CG is the Center of Gravity

Fig 15 is a flow chart for feature extraction.

V-Vector Quantizer Module:

The vector-quantizer module provides a series of quantized observations to the Discrete HMM Decoder module using the codebook made by the Codebook Maker Module during the training phase.

VI-Discrete HMM Decoder Module:

The HMM recognizer stage receives a series of observations representing the scanned word from the vector quantizer and uses the Viterbi algorithm to produce the most probable sequence of models that produces these observations. Using a statistical ligature bi-grams Probabilities generated by Ligature m-grams Prob. Estimator, the most probable model with the highest score is produced by the Viterbi algorithm.

In our system, we used left-to-right HMM as shown in figure 16. The state-transitions are allowed only from the current state to the next state or from the current state to itself (i.e. to stay or to advance). The number of states per model is one of the key factors that affect the system performance. It is specified empirically to increase the performance of the system.

Decoding is controlled by a recognition network compiled from a word-level network, a dictionary and a set of HMMs. The recognition network consists of a set of nodes connected by arcs.

Each node is either a HMM model instance or a word-end. Each model node is itself a network consisting of states connected by arcs. Thus, once fully compiled, a recognition network ultimately consists of HMM states connected by transitions. However, it can be viewed at three different levels: word, model and state. Figure 17 illustrates this hierarchy.

For an unknown input word with *T* frames, every path from the start node to the exit node of the network which passes through exactly *T* emitting HMM states is a potential recognition hypothesis. Each of these paths has a log probability which is computed by summing the log probability of each individual transition in the path and the log probability of each emitting state generating the corresponding observation. Within-HMM transitions are determined from the HMM parameters, between-model transitions are constant and word-end transitions are determined by the language model likelihoods attached to the word level networks.

The job of the decoder is to find those paths through the network which have the highest log probability. These paths are found using a *Token Passing* algorithm. A token represents a partial path through the network extending from time 0 through to time *t*. At time 0, a token is placed in every possible start node.

Each time step, tokens are propagated along connecting transitions stopping whenever they reach an emitting HMM state. When there are multiple exits from a node, the token is copied so that all possible paths are explored in parallel. As the token passes across transitions and through nodes, its log probability is incremented by the corresponding transition and emission probabilities.

A network node can hold at most *N* tokens. Hence, at the end of each time step, all but the *N* best tokens in any node are discarded. As each token passes through the network it must maintain a history recording its route. The amount of detail in this history depends on the required recognition output. Normally, only word sequences are wanted and hence, only transitions out of word-end nodes need be recorded.

<u>VII-Ligature to Character Converter Module:</u>

This module converts the sequence of recognized ligature patterns back into ASCII equivalent codes and the whole recognized text is stored in a text file to be stored.

<u>VIII-Character to Ligature Converter Module:</u>

This module uses the reference text files of the training set to produce a series of pattern numbers of all ligatures included in each word using the ligature-set of the used font and applies this series with quantized feature vectors of the word to the next stage for training the models of the ligatures included in the word.

IX-Ligature m-grams Prob. Estimator Module:

Statistical Language Model (SLM) uses the language text statistics for increasing the recognition accuracy. Language models provide an estimate of the probability of a character sequence for the given recognition task. In our system we used the bigram SLM. The bigram model is based on the approximation that a character (or ligature) is statistically dependant only on the temporally previous character (or ligature). In our system a huge Arabic text corpus from NEMLAR database is used for building an Arabic bigram language model that is used to improve the recognition accuracy.

X-DRNP Estimator Module:

This module is used to detect the dynamic range of the features and calculate the normalization parameter from the feature vectors of the training data after excluding the extreme values. These normalization parameters are then used during the recognition phase to normalize the feature vectors of the data to be recognized.

XI-Codebook Maker Module:

From many images of the training set of documents representing different Arabic fonts with different sizes, the codebook maker creates a codebook using the conventional K-means algorithm. The number of centroids of the codebook is another factor that affects the recognition accuracy. It is selected empirically to maximize the recognition rates of the system.

XII-HMM Trainer Module:

The HMM trainer receives the observation sequence which is the quantized feature vectors of each word, and also the pattern numbers of the ligatures included in this word generated by the character to ligature converter module and then uses the Baum-Welch re-estimation procedure to train the models of these ligatures, i.e. to maximize the probabilities of the observation sequences given the models.

The basic procedures involved in training a set of our system models are gone through three steps:

Step1: HMM Initialization (HInit)

The basic principle of HInit depends on the concept of a HMM as a generator of character vectors. Every training example can be viewed as the output of the HMM whose parameters are to be estimated. Thus, if the state that generated each vector in the training data was known, then the unknown means and variances could be estimated by averaging all the vectors associated with each state. Similarly, the transition matrix could be estimated by simply counting the number of time slots that each state was occupied. The above idea can be implemented by an iterative scheme as shown in Figure18.

Firstly, the Viterbi algorithm is used to find the most likely state sequence corresponding to each training example, then the HMM parameters are estimated. As a side-effect of finding the Viterbi state alignment, the log likelihood of the training data can be computed. Hence, the whole estimation process can be repeated until no further increase in likelihood is obtained. This process requires some initial HMM parameters to get started. To circumvent this problem, HInit starts by uniformly segmenting the data and associating each successive segment with successive states.

Step2: HMM Re-Estimation (HRest)

Its operation is very similar to HInit except that, as shown in Figure 19, it expects the input HMM definition to have been initialized and it uses Baum-Welch re-estimation in place of Viterbi training. This involves finding the probability of being in each state at each time frame using the Forward-Backward algorithm. This probability is then used to form weighted averages for the HMM parameters. Thus, whereas Viterbi training makes a hard decision as to which state each training vector was "generated" by, Baum-Welch takes a soft decision.

Step3: HMM Re-Estimation Embedded Training (HERest)

Whereas isolated unit training is sufficient for building whole word models and initialization of models using hand-labeled bootstrap data, the main HMM training procedures for building subword systems revolve around the concept of embedded training. Unlike the processes described before, embedded training simultaneously updates all of the HMMs in a system using all of the training data. It is performed by HRest step which, unlike HRest, performs just a single iteration.

In outline, HERest step (as shown in figure 20) works as follows. On startup, HERest loads in a complete set of HMM definitions. Every training file must have an associated label file which gives a transcription for that file. HERest processes each training file in turn. After loading it into memory, it uses the associated transcription to construct a composite HMM which spans the whole word. This composite HMM is made by concatenating instances of the phone HMMs corresponding to each label in the transcription. The Forward-Backward algorithm is then applied and the sums needed to form the weighted averages accumulated in the normal way. When all of the training files have been processed, the new parameter estimates are formed from the weighted sums and the updated HMM set is output.

Brief description of the drawings:

Fig 1: represents Grapheme segmentation process illustrated by manually inserting vertical gray lines at appropriate grapheme connection points.

Fig2: represents example sets of dotting-differentiated graphemes.

Fig3: represents the 4 cases; Starting, Middle, Ending, and Separate cases of the grapheme representing character "Ein" highlighted in red.

Fig4: represents some ligatures in the Traditional Arabic font and overlapped characters.

Fig5: represents the diacritics added to Arabic text.

Fig6: represents Discrete HMM Based Recognition System Block Diagram. And it is composed of the following modules:

Quality Scanning (M1) Module:

In this stage, optical scanners are used to convert printed documents to computer images. It is common practice to convert the multilevel image into a bi-level image of black and white. Often this process, known as thresholding, is performed on the scanner to save memory space and computational effort.

Primary Noise Handler (M2) Module

This aims to remove noise and other uninteresting variations in the image which are often byproducts of image acquisition. Filters can be designed for smoothing, sharpening, thresholding, removing slightly textured or colored background, and contrast adjustment purposes. A method that is often used to reduce noise in two dimensional (off-line) images is to traverse the image, pixel-by-pixel, using a 3 x 3 window and to change the value of a pixel based on the value of its 8-neighbors.

In our study, a 5×5 median filter has proven effective for the cleaning of input scanned text images from the *salt & pepper* noise

Median filters are, however, hard deciding and cause considerable erosive distortion to the graphemes of written text which is generally harmful to the recognition performance. So, they should be used only while the lines & words decomposition, whereas other softer deciding filters – like *Gaussian* ones – might be used - if necessary - while the recognition process..

Word Decomposer (M3) Module

The word decomposer module is used to segment paper into lines, and then to segment each line into words or sub-words starting from upper right corner of the page. A robust Histogram-based algorithm is used for lines & words decomposition

Feature Extractor (M4) Module

The feature-extractor module is used to extract features of the sliding window, which slides through the body of the Arabic word from right to left, and provides a series of feature vectors to the vector quantizer module.

The basic philosophy of our autonomously differential recognition feature is as follows;. *given a sequence of such a feature, one can fully retrieve the written script with only a scaling factor as well as the digital distortion error.* The feature proposed here is a time-domain differential one where the script contour is totally differentiated along the x-direction, which can be expressed by equation [1] as follows:

$$\underline{F}_i = D\left({}^{d\underline{\varsigma}}\!\big/\!{}_{dx}\Big|_{x=i} \right) = [\underline{S}_1 \cup \underline{S}_2 \cup \ldots \underline{S}_M] = \bigcup_{j=1}^{M} \underline{S}_j \qquad \text{Eq. [1]}$$

Vector Quantizer (M5) Module

The vector-quantizer module provides a series of quantized observations to the Discrete HMM Decoder module using the codebook made by the Codebook Maker Module during the training phase.

Discrete HMM Decoder (M6) Module:

The HMM recognizer stage receives a series of observations representing the scanned word from the vector quantizer and uses the Viterbi algorithm to produce the most probable sequence of models that produces these observations. Using a statistical ligature bi-grams Probabilities generated by Ligature m-grams Prob. Estimator, the most probable model with the highest score is produced by the Viterbi algorithm.

In our system, we used left-to-right HMM. The state-transitions are allowed only from the current state to the next state or from the current state to itself (i.e. to stay or to advance). The number of states per model is one of the key factors that affect the system performance. It is specified empirically to increase the performance of the system.

Ligature to Character Converter (M7) Module:

This module converts the sequence of recognized ligature patterns back into ASCII equivalent codes and the whole recognized text is stored in a text file to be stored.

Character to Ligature Converter (T1) Module:

This module uses the reference text files of the training set to produce a series of pattern numbers of all ligatures included in each word using the ligature-set of the used font and applies this series with quantized feature vectors of the word to the next stage for training the models of the ligatures included in the word.

Ligature m-grams Prob. Estimator (T2) Module:

Statistical Language Model (SLM) uses the language text statistics for increasing the recognition accuracy. Language models provide an estimate of the probability of a character sequence for the given recognition task. In our system we used the bigram SLM. The bigram model is based on the approximation that a character (or ligature) is statistically dependant only on the temporally previous character (or ligature). In our system a huge Arabic text corpus from NEMLAR database is used for building an Arabic bigram language model that is used to improve the recognition accuracy.

DRNP Estimator (T3) Module

This module is used to detect the dynamic range of the features and calculate the normalization parameter from the feature vectors of the training data after excluding the extreme values. These normalization parameters are then used during the recognition phase to normalize the feature vectors of the data to be recognized.

Codebook Maker (T4) Module

From many images of the training set of documents representing different Arabic fonts with different sizes, the codebook maker creates a codebook using the conventional K-means algorithm. The number of centroids of the codebook is another factor that affects the recognition accuracy. It is selected empirically to maximize the recognition rates of the system.

HMM Trainer (T5) Module

The HMM trainer receives the observation sequence which is the quantized feature vectors of each word, and also the pattern numbers of the ligatures included in this word generated by the character to ligature converter module and then uses the Baum-Welch re-estimation procedure to train the models of these ligatures, i.e. to maximize the probabilities of the observation sequences given the models.

Fig7: represents the Gray values histogram of the pixels population of the sample text rectangle.

Fig8: represents the Sample text rectangle after initial lines decomposition.

Fig9: represents the Sample text rectangle after refined lines decomposition.

Fig10: represents the Sample text rectangle after logical writing lines concatenation.

Fig11: represents the Sample text rectangle after part-of-words decomposition.

Fig12: represents a hypothetical histogram of intra-word, inter-word, and rogue part-of-word spaces.

Fig13: represents the Sample text rectangle after full words concatenation.

Fig14: represents the illustration of the frames and segments.

Fig15: represents the flow chart for feature extraction.

Fig16: represents the Simple Left-Right HMM.

Fig17: represents the Recognition Network Levels.

Fig18: represents the Initialization Operation.

Fig19: represents the HRest Operation.

Fig20: represents the HERest File Processing

Claims

1. System for enhancing the Arabic OCR includes;
Means for feature extractor, training process, and recognition process
System is divided into two subsystems: the trainer subsystem (feature extractor, training process) which is used to initialize and train Hidden Markov Models using many scanned bitmaps with their corresponding text files, and the Recognizer subsystem (feature extractor, and recognition process) which uses the Viterbi algorithm to recognize input bitmaps to output characters.
The system enhances the problem of the Arabic OCR system by using the new features extractor steps.

The new feature-extractor module is used to extract features of the sliding window, which slides through the body of the Arabic word from right to left, and provides a series of feature vectors to the vector quantizer module. The basic philosophy of our autonomously differential recognition feature is as follows; given a sequence of such a feature, one can fully retrieve the written script with only a scaling factor as well as the digital distortion error. The feature proposed here is a time-domain differential one where the script contour is totally differentiated along the x-direction, which can be expressed by equation [1] as follows:

$$\underline{F}_i = D\left({}^{d\underline{\varsigma}}\!\big/_{dx}\Big|_{x=i} \right) = [\underline{S}_1 \cup \underline{S}_2 \cup \ldots \underline{S}_M] = \bigcup_{j=1}^{M} \underline{S}_j \qquad \text{Eq. [1]}$$

Where F_i is the frame feature vector and S_j is the segment feature vector

2. According to claim 1 this feature can be used with any language (i.e. Arabic, English,...) whether it was with connected characters or with isolated characters ,Also whether it was handwritten or type written.

3. According to claim 1 this feature can be implemented on any platform whether it was implemented in a software on a computer with any operating system (Windows, Linux, OS2, Solaris,.....) or was implemented on any hardware device.

4. The method for calculating features according to claim 1, comprise
The words are consisting of number of frames and each frame is consisting of number of segments, and the max number of segments per frame is 4 in Arabic fonts.

For each segment we calculate it according to the location of its frame.

The first case if the Segment is located in the start frame of the word:

$$S=\left[\frac{\text{CG. of the segment - CG. of the first segment}}{\text{First segment length}},\ \frac{\text{segment length}}{\text{First segment length}},\ -1,\ -1\right]$$

5. The method for calculating features according to claim 1 and 2, comprise
The words are consisting of number of frames and each frame is consisting of number of segments, and the max number of segments per frame is 4 in Arabic fonts.

The second case if the Segment is located in a frame came after a non empty frame:

$$S=\left[\frac{\text{CG. of the segment - CG. of the previous frame}}{\text{previous frame length}},\ \frac{\text{segment length}}{\text{previous frame lengthh}},\ \text{1st overlapped segment, last overlapped segment}\right]$$

$\longrightarrow$ **If there is overlapping.**

$$S=\left[\frac{\text{CG. of the segment - CG. of the previous frame}}{\text{previous frame length}},\ \frac{\text{segment length}}{\text{previous frame lengthh}},\ -2,\ -2\right]$$

$\longrightarrow$ **If there is no overlapping.**

6. The method for calculating features according to claim 1 and 2, comprise
The words are consisting of number of frames and each frame is consisting of number of Segments and the max number of segments per frame is 4 in Arabic fonts.

The third case if Segment located in a frame came after an empty frame in the same word:

$$S=\left[\frac{\text{CG. of the segment - CG. of the first previous non empty frame}}{\text{first previous non empty frame length}},\ \frac{\text{CG. of the segment - CG. of the first previous non empty frame}}{\text{first previous non empty frame length}},\ -3,\ -3\right]$$

7. The method for calculating features according to claim 1 and 2, comprise
The words are consisting of number of frames and each frame is consisting of number of segments, and the max number of segments per frame is 4 in Arabic fonts.

The fourth case if the frame is Empty:

$$S=[0, 0, 0, 0]$$

8. According to any previous claims, the process of the new feature is done by the following steps:

1- Getting a new word

2- Get frame

3- Test if the frame is any case from the four previous cases at claims 2, 3, 4, and 5

4- Calculate the frame features according to the case that determined in step 3

5- If this frame is not the last one go to step 2

6- End

9. According to claim 1, the training process includes:

Quality Scanning (M1) Module, Primary Noise Handler (M2) Module, Word Decomposer (M3) Module, Feature Extractor (M4) Module, Vector Quantizer (M5) Module and Discrete HMM Decoder (M6) Module.

10. According to claim 1, the recognition process includes:

Quality Scanning (M1) Module, Primary Noise Handler (M2) Module, Word Decomposer (M3) Module, Feature Extractor (M4) Module, Vector Quantizer (M5) Module, Discrete HMM Decoder (M6) Module, Ligature to Character Converter (M7) Module, Character to Ligature Converter (T1) Module, Ligature m-grams Prob. Estimator (T2) Module, DRNP Estimator (T3) Module, Codebook Maker (T4) Module and HMM Trainer (T5) Module.

11. Method for enhancement Arabic OCR includes:

Method for feature extractor, training process, and recognition process
System is divided into two subsystems: the trainer subsystem (feature extractor, training process) which is used to initialize and train Hidden Markov Models using many scanned bitmaps with their corresponding text files, and the Recognizer subsystem (feature extractor, and recognition process) which uses the Viterbi algorithm to recognize input bitmaps to output characters.
The system enhances the problem of the Arabic OCR system by using the new features extractor steps.

The new feature extractor module is used to extract features of the sliding window, which slides through the body of the Arabic word from right to left, and provides a series of feature vectors to the vector quantizer module. The basic philosophy of our autonomously differential recognition feature is as follows; given a sequence of such a feature, one can fully retrieve the written script with only a scaling factor as well as the digital distortion error. The feature proposed here is a time-domain differential one where the script contour is totally differentiated along the x-direction, which can be expressed by equation [1] as follows:

$$\underline{F}_i = D\left(\left. {}^{d\varsigma}\!\!\diagup_{dx} \right|_{x=i} \right) = [\underline{S}_1 \cup \underline{S}_2 \cup \ldots \underline{S}_M] = \bigcup_{j=1}^{M} \underline{S}_j \qquad \text{Eq. [1]}$$

Where F_i is the frame feature vector and S_j is the segment feature vector

12. According to claim 11 this feature can be used with any language (i.e. Arabic, English,...) whether it was with connected characters or with isolated characters ,Also whether it was handwritten or type written.

13. According to claim 11 this feature can be implemented on any platform whether it was implemented in a software on a computer with any operating system (Windows, Linux, OS2, Solaris,.....) or was implemented on any hardware device.

14. According to claim 11, the method for calculating features according to claim 1, comprise
The words are consisting of number of frames and each frame is consisting of number of segments, and the max number of segments per frame is 4 in Arabic fonts.

For each segment we calculate it according to the location of its frame.

The first case if Segment located in the start frame of the word

$$S = \left[\frac{\text{CG. of the segment} - \text{CG. of the first segment}}{\text{First segment length}}, \frac{\text{segment length}}{\text{First segment length}}, -1, -1 \right]$$

- The second case if Segment located in a frame came after a non empty frame:

$$S = \left[\frac{\text{CG. of the segment} - \text{CG. of the previous frame}}{\text{previous frame length}}, \frac{\text{segment length}}{\text{previous frame lengthh}}, \text{1st overlapped segment}, \text{last overlapped segment} \right]$$

$\longrightarrow$ **If there is overlapping.**

$$S = \left[\frac{\text{CG. of the segment} - \text{CG. of the previous frame}}{\text{previous frame length}}, \frac{\text{segment length}}{\text{previous frame lengthh}}, -2, -2 \right]$$

$\longrightarrow$ **If there is no overlapping.**

- The third case if Segment located in a frame came after an empty frame in the same word:

$$S= \left[\frac{\text{CG. of the segment - CG. of the first previous non empty frame}}{\text{first previous non empty frame length}} , \frac{\text{CG. of the segment - CG. of the first previous non empty frame}}{\text{first previous non empty frame length}} , -3 , -3 \right]$$

- The fourth case if the frame is Empty:

$$S= [0, 0, 0, 0]$$

15. Feature extractor Process is calculated by the algorithm according to the following equation

$$\underline{F}_i = D\left(\left. {}^{d\underline{\varsigma}}\!/_{dx} \right|_{x=i} \right) = [\underline{S}_1 \cup \underline{S}_2 \cup ..\underline{S}_M] = \bigcup_{j=1}^{M} \underline{S}_j \qquad \text{Eq. [1]}$$

Where F_i is the frame feature vector and S_j is the segment feature vector

Includes the following steps:

1- Getting anew word
2- Get frame
3- Test if the frame is any case from the four previous cases at claim 10
4- Calculate the frame features according to the case that determined in step 3
5- If this frame is not the last one go to step 2
6- End

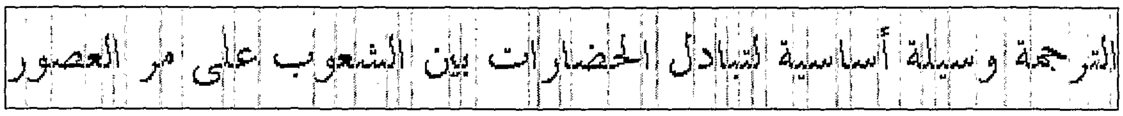

الترجمة وسيلة أساسية لتبادل الحضارات بين الشعوب على مر العصور

Fig. 1

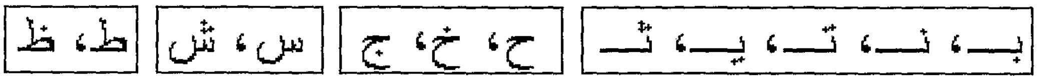

بـ، نـ، تـ، يـ، ثـ
ح، خ، ج
س، ش
ط، ظ

Fig. 2

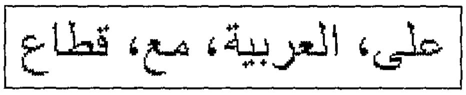

على، العربية، مع، قطاع

Fig. 3

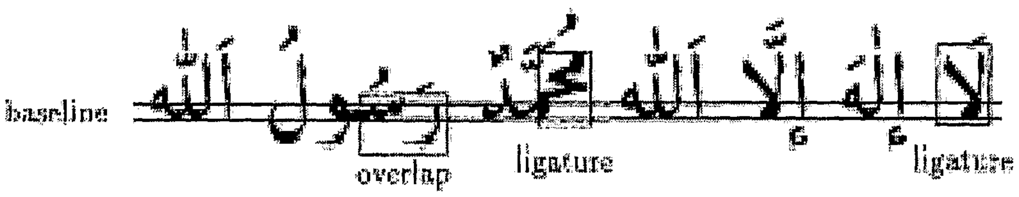


Fig. 4

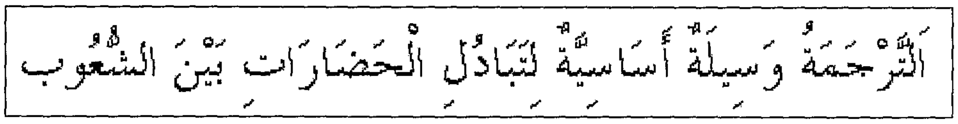

اَلتَّرْجَمَةُ وَسِيلَةٌ أَسَاسِيَّةٌ لِتَبَادُلِ الْحَضَارَاتِ بَيْنَ الشُّعُوبِ

Fig. 5

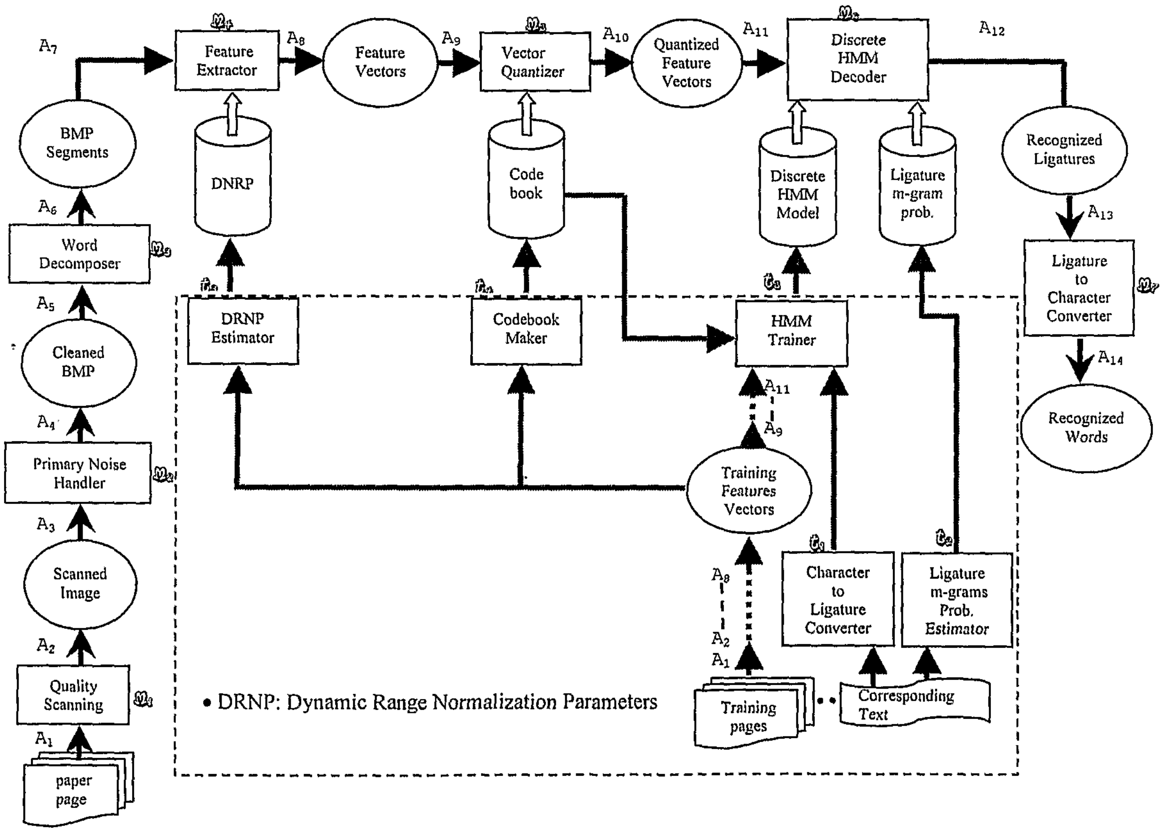


Fig.6

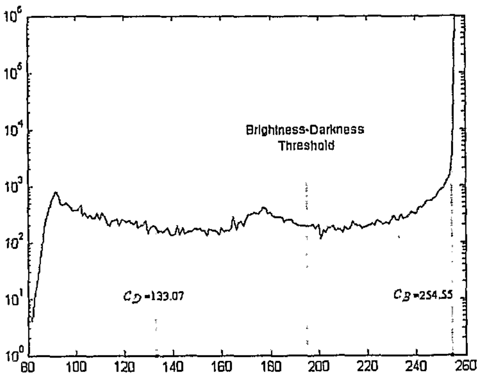


Fig. 7

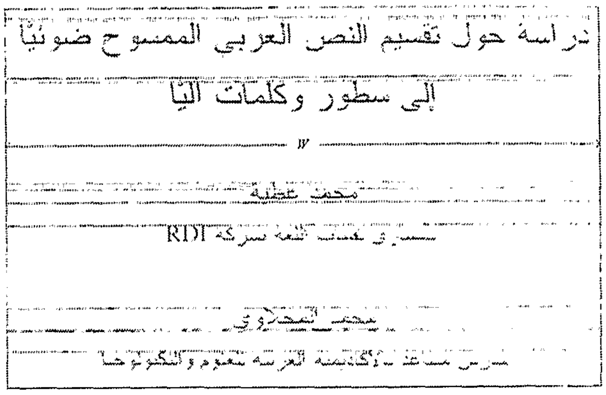

دراسة حول تقسيم النص العربي الممسوح ضوئيًا
إلى سطور وكلمات آليًا
W
محمد عطية
مستشار تقنيات اللغة شركة RDI
محمد المحلاوي
مدرس مساعد بالأكاديمية العربية للعلوم والتكنولوجيا

Fig. 8

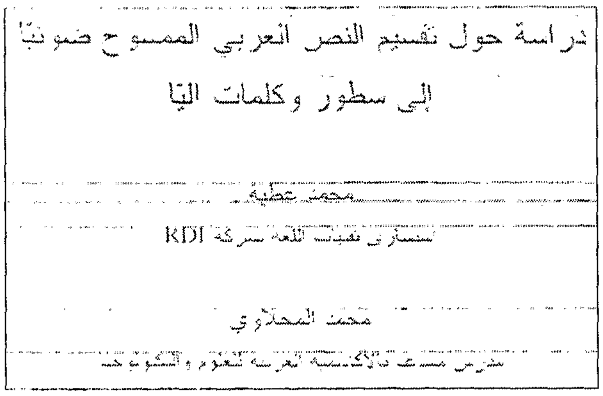

دراسة حول تقسيم النص العربي الممسوح ضوئيًا
إلى سطور وكلمات آليًا
محمد عطية
مستشار تقنيات اللغة شركة RDI
محمد المحلاوي
مدرس مساعد بالأكاديمية العربية للعلوم والتكنولوجيا

Fig. 9

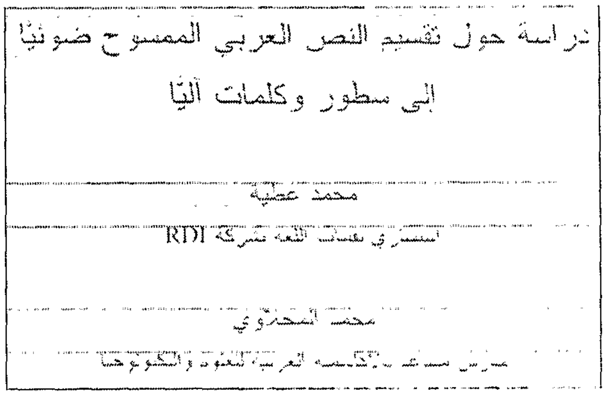

دراسة حول تقسيم النص العربي الممسوح ضوئيًا
إلى سطور وكلمات آليًا
محمد عطية
مستشار تقنيات اللغة شركة RDI
محمد المحلاوي
مدرس مساعد بالأكاديمية العربية للعلوم والتكنولوجيا

Fig. 10

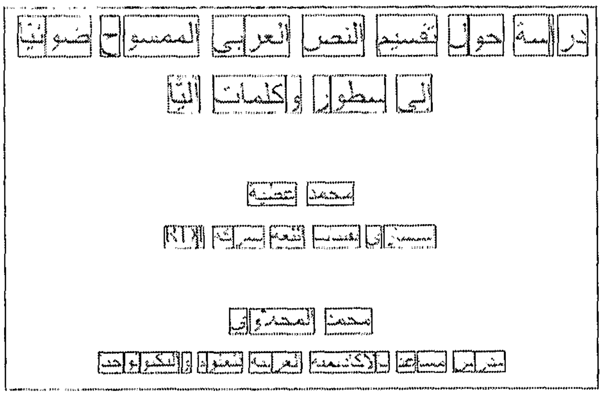

دراسة حول تقسيم النص العربي الممسوح ضوئيًا
إلى سطور وكلمات آليًا
محمد عطية
مستشار تقنيات اللغة شركة RDI
محمد المحلاوي
مدرس مساعد بالأكاديمية العربية للعلوم والتكنولوجيا

Fig. 11

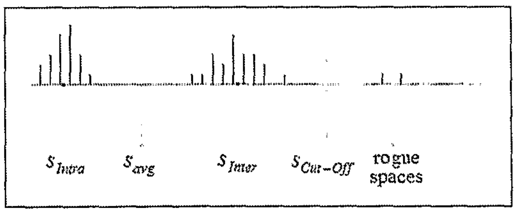


Fig. 12

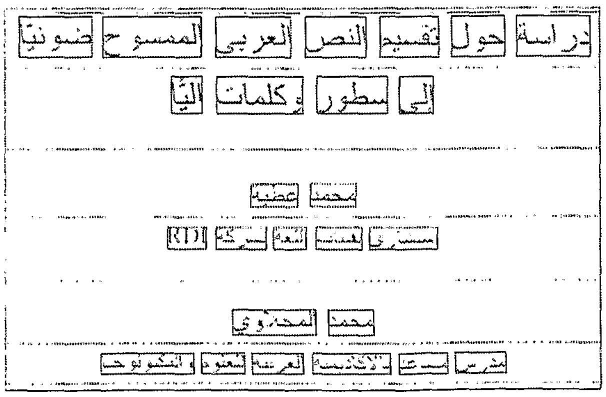
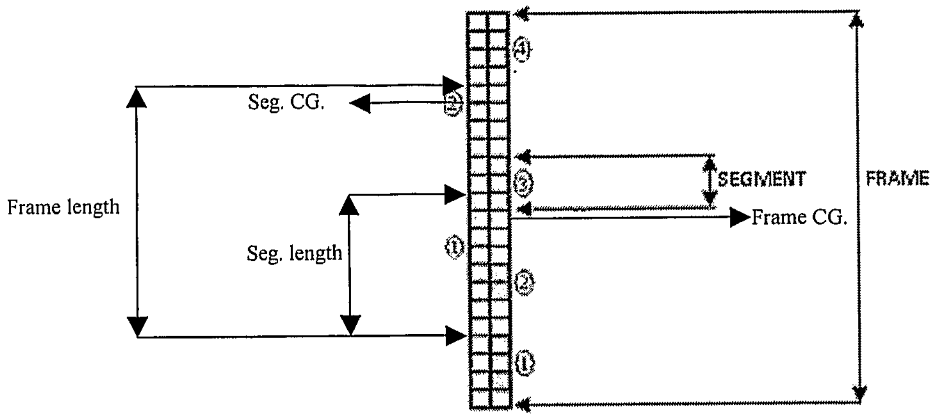
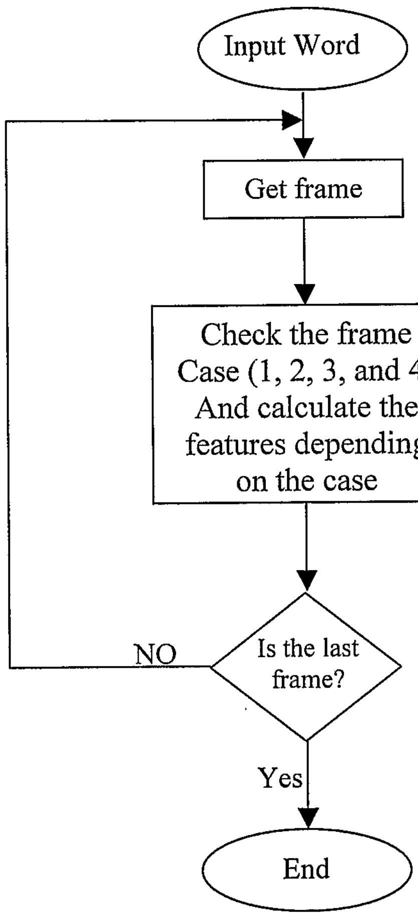

دراسة حول تقسيم النص العربي الممسوح ضوئيًا
إلى سطور وكلمات آليًا
محمد عطية
مستشار تقنيات اللغة شركة RDI
محمد المحلاوي
مدرس مساعد بالأكاديمية العربية للعلوم والتكنولوجيا

Fig. 13



Seg. CG.
Frame length
Seg. length
SEGMENT
FRAME
Frame CG.
④
③
③
②
①
①

Fig. 14



Input Word
Get frame
Check the frame Case (1, 2, 3, and 4) And calculate the features depending on the case
Is the last frame?
NO
Yes
End

Fig. 15

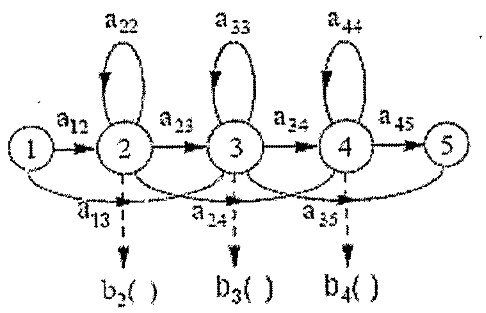


Fig. 16

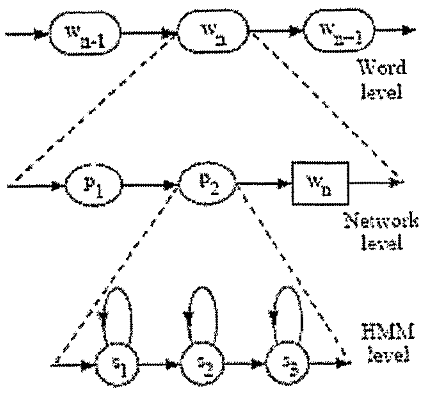


Fig. 17

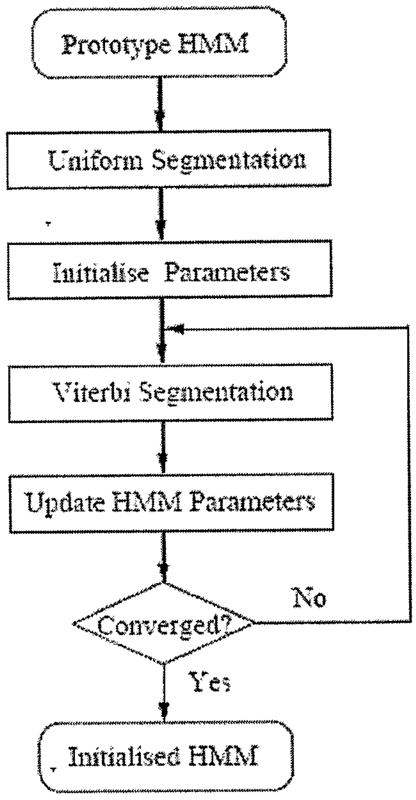


Fig. 18

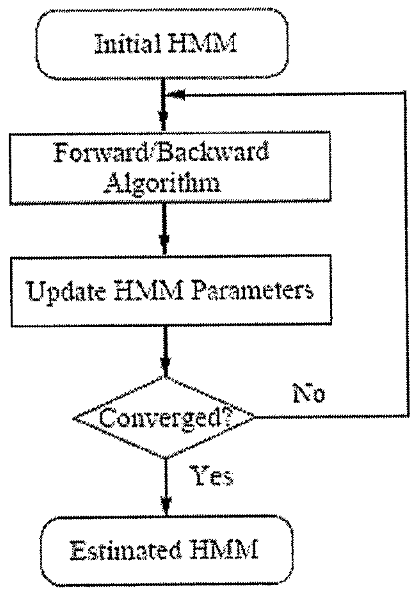


Fig. 19

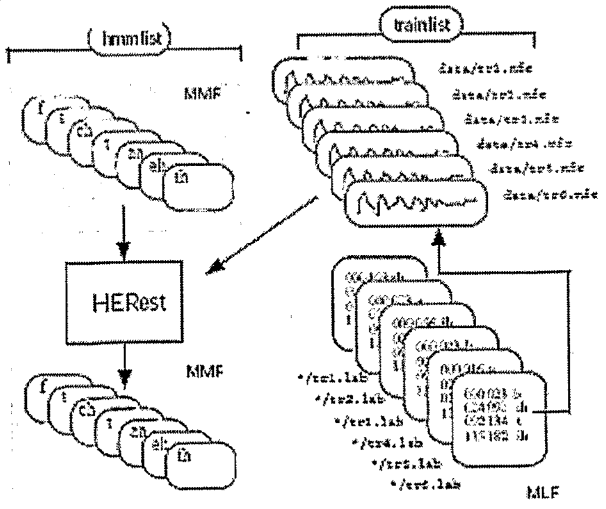


Fig. 20