To appear in 10th Int'l Conf. on Intelligent Systems Design and Applications (ISDA'10), 2010.

Recognizing Handwritten Arabic Script through Efficient Skeleton-Based Grapheme Segmentation Algorithm

Gheith A. Abandah, Fuad T. Jamour
Computer Engineering Department
The University of Jordan
Amman 11942, Jordan
abandah@ju.edu.jo, fjamour@gmail.com

***Abstract*—To recognize unlimited set of handwritten Arabic words, an efficient segmentation algorithm is needed to segment these cursive words into a limited set of primal graphemes. We propose a rule-based segmentation algorithm that segments cursive words into graphemes through collecting special feature points from the word skeleton. The development of this algorithm is motivated by the need to solve problems and limitations available in the state-of-the-art algorithms in this area. The preliminary evaluation of the proposed algorithm is promising with over 96% accuracy on a sample subset of the IFN/ENIT database.**

Keywords-optical character recognition; handwritten script; letter segmentation algorithms; Arabic language

I. INTRODUCTION

The recognition of handwritten scripts is a challenging task particularly due to high variability in letter shapes. The challenge with the Arabic scripts is even larger because Arabic is always cursive. If Arabic words are to be recognized with no explicit segmentation, the number of recognition classes becomes equal to the number of possible words, which is overly huge. A more practical approach is to segment Arabic words into letters then to recognize the segmented letters. This approach works well with printed words because segmenting printed Arabic words is relatively easy. Earlier researchers have develop accurate systems for recognizing printed Arabic text and some commercial products are available [1].

However, segmenting handwritten Arabic words is not as easy. Handwritten words lack the uniformity needed to detect invariant features for pinpointing the segmentation points. Most of the successful handwritten Arabic recognition engines use holistic approaches that recognize whole words without segmenting them to letters [2]. Such approach is successful with limited lexicon, thus, it is not a general solution.

We believe that building a practical, unlimited lexicon recognition system for handwritten Arabic script requires robust letter segmentation algorithm. We have studied several existing segmentation algorithms and noticed their strengths and limitations. In the following subsection, we briefly describe eight of the leading algorithms, state their major limitations, and provide some examples where they fail.

A. Letter Segmentation Algorithms Review

Motawa *et al.* use mathematical morphology to detect what they call regularities [3]. The segmentation points are then located within these regularities using certain rules. These regularities are stroke segments that each connects two vertical strokes. Their algorithm depends on the length and orientation of the regularity, stroke width, baseline, and some logical rules. The regularities should be horizontal and close to the baseline in order to be considered. Short regularities might have one segmentation point while long regularities might have two with no clear guidelines how exactly to locate these points. We believe that the features used in this algorithm are unreliable. The lengths of various handwritten strokes are misleading for segmentation and the baseline does not necessarily pass through all the segmentation points. The word in Figure 1(a) illustrates three limitations of this algorithm, the rightmost regularity is not segmented because it is not horizontal, the leftmost long regularity is over segmented in two segmentation points instead of one, and the medial **Seen** (سـ) is also over segmented with no special treatment.

Mansour *et al.* use the stroke width to locate segmentation points [4]. A location where the stroke width is narrower than the average stroke width is considered a segmentation point. This approach does not give accurate results. The stroke width is not a good indicator of the existence of a segmentation point. The leftmost box in Fig. 1(b) contains a stroke with a pit that gives a false segmentation of the isolated letter **Theh** (ث). On the other hand, the rightmost box contains a constant-width stroke where the segmentation point between initial **Meem** (مـ) and final **Alef** (ا) is missed.

Sari *et al.* use local minima detected on the lower contour of the word [5]. A local minimum that fulfills some rules is considered a segmentation point. Though the stated rules are powerful and cover many cases, the proposed approach does not perform well on handwritings because many sub-words containing segmentation points do not have local minima. For example, the sub-word in the leftmost box of Fig. 1(c) does not have a local minimum. And on the other hand, the rightmost box that has the isolated **Yeh** (ي) is an example of a letter that is falsely segmented because it has a local minimum.

This work is supported by the Deanship of Academic Research, The University of Jordan.

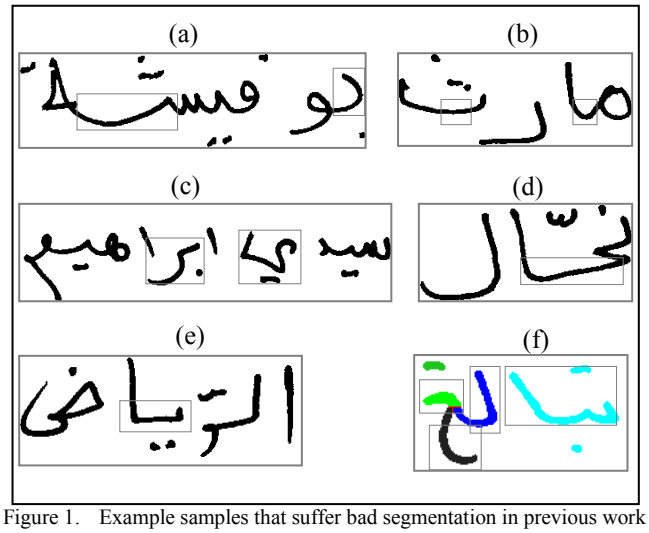


Figure 1. Example samples that suffer bad segmentation in previous work

Lorigo and Govindaraju get the baseline from the truth information of the word database they use, IFN/ENIT [6]. After they refine the baseline for each sub-word using the horizontal histogram method, they find segmentation points in horizontal strokes that lie in the baseline strip. The baseline strip is a horizontal region that extends above and below the horizontal baseline of the word. The limitations of this algorithm are already described in [6]. The most significant limitation is the algorithm's inability to handle sub-words with slanted strokes, e.g., the word in Fig. 1(b). Moreover, the high dependence on the baseline value is not safe in handwritings; there exist many cases where the sub-word does not have a consistent baseline.

Xiu *et al.* use the upper contour and the stroke width [7]. Local minima and regions of narrow strokes are initially considered segmentation points. The algorithm first performs over-segmentation then recognizes the produced graphemes. The segmentation points are then modified based on the grapheme pre-recognition results before producing the final recognition output. The algorithm fails for cases like the cases shown in Fig. 1(b) and 1(c).

Bentrcia and Elnagar use what they call *agents* to detect segmentation points [8]. The baseline is one of these agents. Other agents include cavities, loops, and the letter **Seen**. Some of the limitations of this algorithm are already mentioned in [8]. The algorithm conceptually works fine, but some segmentation points are missed because of lack of agents near them, and the detection of the agents is error prone because it is highly dependent on the accuracy of the baseline detection. Fig. 1(e) illustrates a case where the segmentation point is missed because of lack of agents.

Wshah *et al.* use the skeleton and the boundary of the word to do over/under segmentation [9]. The output of the algorithm is graphemes that represent anything between one fifth of a letter and three connected letters. The proposed segmentation concept does not eliminate the need of using a lexicon because the output of the algorithm contains combinations of up to three letters. Moreover, the graphemes from over-segmentation have unpredictable shapes, complicating the recognition engine design. Fig. 1(f) is an example of the output of this algorithm. Each box outlines one grapheme: the rightmost sub-word which contains three letters is not segmented, and the final **Ghain** (غ) is segmented into two graphemes.

Samoud *et al.* use the baseline detected using Hough transform and the corners detected using Harris operator [10]. A corner point that is very close to the baseline (within 5 pixels) is considered a segmentation point. The first major limitation of this algorithm is the high dependence on the baseline. The second limitation is the low reliability of corner detection. Hence, the algorithm performs over/under segmentation in an unpredictable way.

Most of these algorithms ignore the treatment of secondary bodies like dots and their association to their respective letters. Thus, they produce a set of candidate segmentation points rather than segmented letters.

In this paper, we present a segmentation algorithm that avoids most of the above limitations. It segments a handwritten Arabic word into a set of isolated graphemes and maps the recognized graphemes into letters. Our segmentation algorithm uses features solely extracted from the skeleton. We believe that the skeleton contains enough and reliable information for finding the correct segmentation points.

This paper is organized as follows: Section II describes our segmentation and recognition approach, Section III presents some experimental results obtained by running our system on selected samples from the IFN/ENIT database [11], and Section IV provides some conclusions and outlines our plans for future work.

II. APPROACH

To save development time, we use the open-source trainable OCR engine Tesseract for character recognition [12]. As this engine can recognize isolated characters only, we have inserted our segmentation algorithm in this engine. Fig. 2 summarizes our approach for recognizing handwritten Arabic text. This recognition process is carried out in three main stages: (1) sub-word separation, (2) grapheme segmentation, and (3) recognition and post-processing. Each stage consists of two or three steps described in detail in the following subsections.

The input image contains one line of text passed to our routines from Tesseract after performing line segmentation and preprocessing. Moreover, Tesseract's character recognition engine is used for grapheme recognition in the third stage.

A. Sub-word Separation

Sub-word separation is important because it simplifies the segmentation process by removing any overlap between adjacent sub-words and thus avoiding some segmentation problems. More details about sub-word (or PAW – Part of Arabic Word) can be found in [13]. As shown in Fig. 2, this stage has two steps. The baseline is first estimated (shown as light horizontal line) and main bodies and secondary bodies are identified (secondary bodies are marked by boxes). Then the main bodies of the sub-words are extracted and secondary bodies are attached to their respective main bodies. The following paragraphs explain these steps.

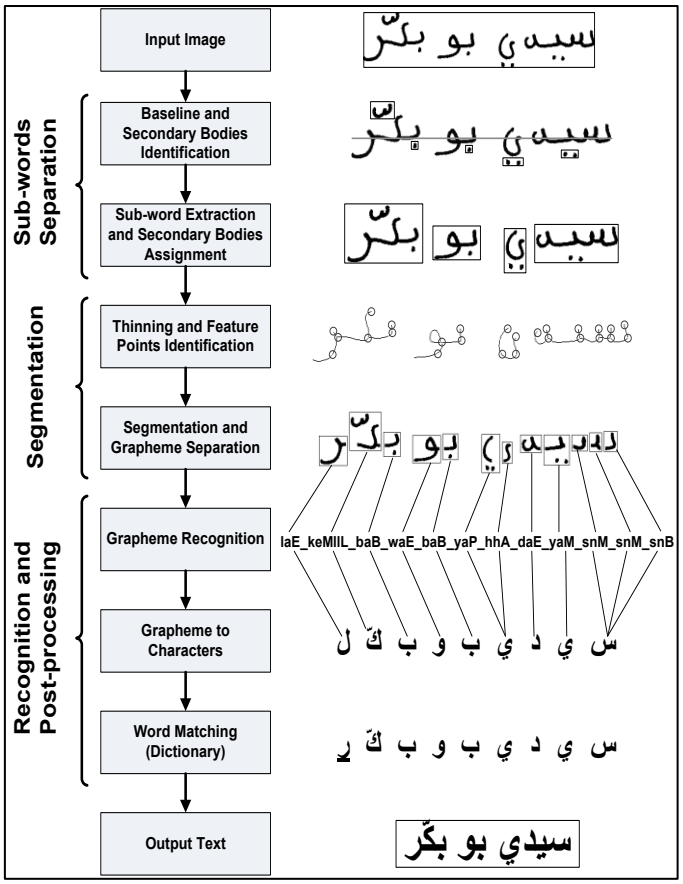


Figure 2. The proposed approach consists of three main stages

1) Baseline Estimation

We estimate the location of the script baseline to effectively identify the secondary bodies as described in the following paragraph. We use a simple yet adequate baseline estimation method, the horizontal projection histogram. The row that contains the maximum number of black pixels is considered the baseline. More accurate and sophisticated baseline estimation algorithms are discussed in [14].

2) Secondary Bodies Identification

The connected components analysis is used to identify the components of the image and to identify the secondary bodies. More than half the Arabic letters are composed of main body and secondary components [15]. The *secondary components* are letter components that are disconnected from the main body such as dots and diacritics. The main bodies of one or more adjacent letters can be connected in the cursive Arabic script to comprise *sub-words*.

A body is classified secondary when one of the following conditions applies: (1) it is very small compared to other bodies in the same image, (2) it is relatively small and far from the baseline, and (3) it is a vertical line and has a relatively large body below it. The light grey bodies in Fig. 3(a) are examples of identified secondary bodies. Furthermore, secondary bodies that are close to each other and similar in size are considered one *secondary body group*. This grouping is important in grapheme separation as described in Subsection II.B.5.

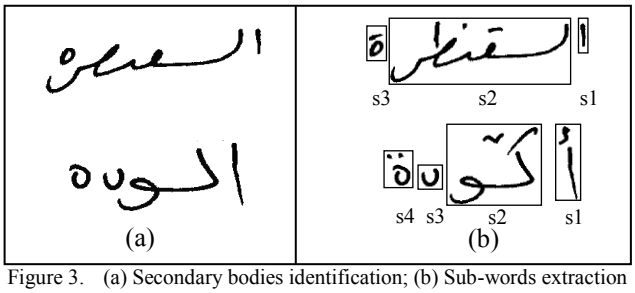


Figure 3. (a) Secondary bodies identification; (b) Sub-words extraction

3) Sub-word Extraction and Secondary Bodies Assignment

Bodies that are not secondary bodies are main bodies of the sub-words. Every sub-word is extracted with its secondary bodies and passed to the next stages. Secondary bodies are examined from right to left. For every secondary body, the algorithm assigns it to the main body that is above its midpoint, below its midpoint, above its left endpoint, below its left endpoint, to its right, or the rightmost main body in the word (first that applies). In Fig. 3(b), the upper word contains three sub-words, and the lower word contains four sub-words. Note that the overlap between some of the adjacent sub-words in Fig. 3(a) is eliminated.

B. Segmentation

In this stage, sub-words are segmented to graphemes and passed to the recognition stage. Passing graphemes to the recognition engine instead of passing letters is done to tolerate over-segmentation of complex letters and under-segmentation of vertical ligatures. In Fig. 2, the rightmost letter (initial **Seen** (س)) is over-segmented to three graphemes and then successfully reconstructed in the post processing step. The segmentation algorithm would be much more complex if it must produce individual letters [16]. The graphemes to characters algorithm described in Subsection II.C.2 takes care of mapping graphemes into individual characters. Our segmentation algorithm uses morphological features extracted from the skeleton of the sub-words as described below.

1) Thinning and Feature Points Identification

We temporarily remove the secondary bodies before using Deutsch's thinning algorithm to get the skeleton of the sub-word's main body [17]. Feature points identified from the skeleton include *end points*, *branch points*, and *cross points*. Feature points are identified by examining the eight neighbors of every skeleton pixel. An end point has one black neighbor, a branch point has three, and a cross point has four (see Fig. 4).

2) Continuities Identification

Continuity is a continuous string of black pixels in the skeleton that connects two feature points. If the starting point is the ending point then the continuity is a simple *loop*. Fig. 5 shows a sub-word composed of three continuities. The attributes kept for each continuity include its pixels and the features at its ends.

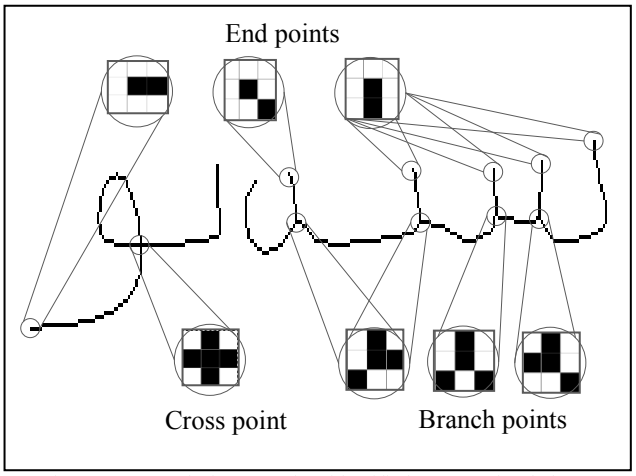


Figure 4. Sample feature points identified on a word's skeleton

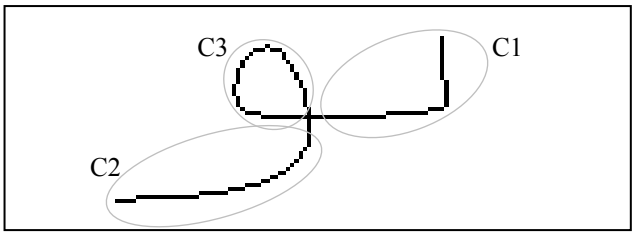


Figure 5. Continuity Examples

3) Subtle Branch Points and Edge Point Detection

Subtle branch points are branch points that are lost through thinning. An example is shown in Fig. 6. They are extracted by searching the skeleton for local maxima within a window of size proportional to the average stroke width.

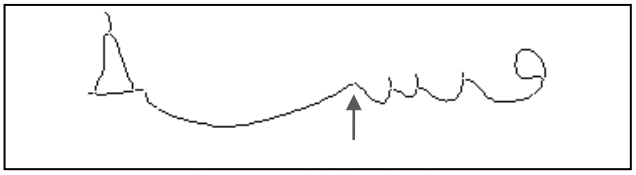


Figure 6. Subtle branch point

Edge points are points where the direction of the stroke changes and are detected using the polygonal approximation of the skeleton. Each vertex in the polygon that is not an end point, cross point, nor branch point is considered an edge point. Note the example in Fig. 7 that has two edge points. Each edge point has two parameters: the *edge angle* (EA) which is a measure of the edge type and the *bisector angle* (BA) which is a measure of the edge direction. The edge angle is the angle between the two lines that make the edge point. The bisector angle is the angle between the bisector line of the edge and the horizontal line. Edge angles have values between 0º and 180º and bisector angles have values between 0º and ±180º.

Edge points that have large edge angles and bisector angles close to 90º or -90º are ignored because they are usually local minima or maxima. From now on, we call subtle branch points and edge points feature points in addition to the previous three simple feature points. Once these two feature points are detected, a continuity that contains these points is split into as many new continuities as needed. At the end of this step, we get a list of continuities that each has two ends. The orientation of the continuity and the attributes of its ends are used to extract the segmentation points as explained in the following subsection.

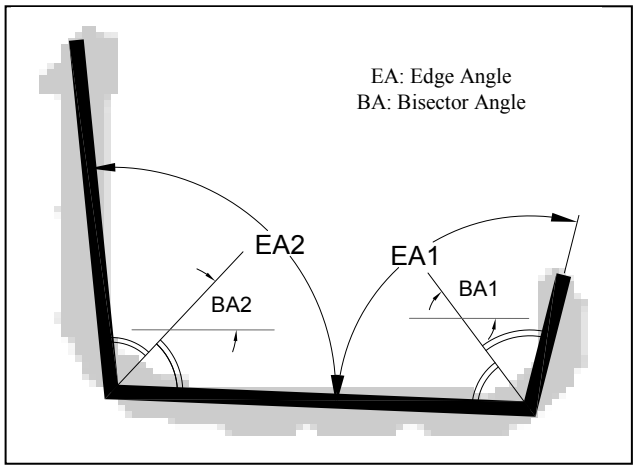


Figure 7. Polygonal approximation with two edge points

4) Segmentation Rules

Continuities that have the following properties are segmented. In Fig. 8, continuities that have these properties are marked with the letter C and continuities that don't have any of these properties are marked with the letter N and not segmented:

- Not vertical: the orientation of the continuity should be between -45º and +45º. In Fig. 8, the continuities C1 through C11 have this property, while N1 doesn't.
- If the right end is an edge, its bisector angle should be between 45º and -135º, as in C1, C3, C4, and C11.
- The left end is not an end point. For violating this property, continuities N3, N4, and N5 are not segmented.
- If the left end is an edge, its bisector angle should be between -155º and 65º, as in C3 and C10.
- It is not totally covered from above or from below except in cases like C2 and C4 where there is clearance in the upper-left direction or lower-right direction, respectively. This property avoids segmenting loops and other cases as in N2 and N6.

The angles in the above rules are selected according to the shapes of Arabic letters that have edges in different configurations. For example, the sub-word (نا) contains two letters: the right letter is initial **Noon** (ن) that has an edge with bisector angle of 135º, and the left letter is final **Alef** (ا) that has an edge with bisector angle of 45º. The continuity connecting the two edges is horizontal. The angle ranges we have selected cover the various cases of letter configurations. The continuities that have the above properties are then examined to select the exact cut position.

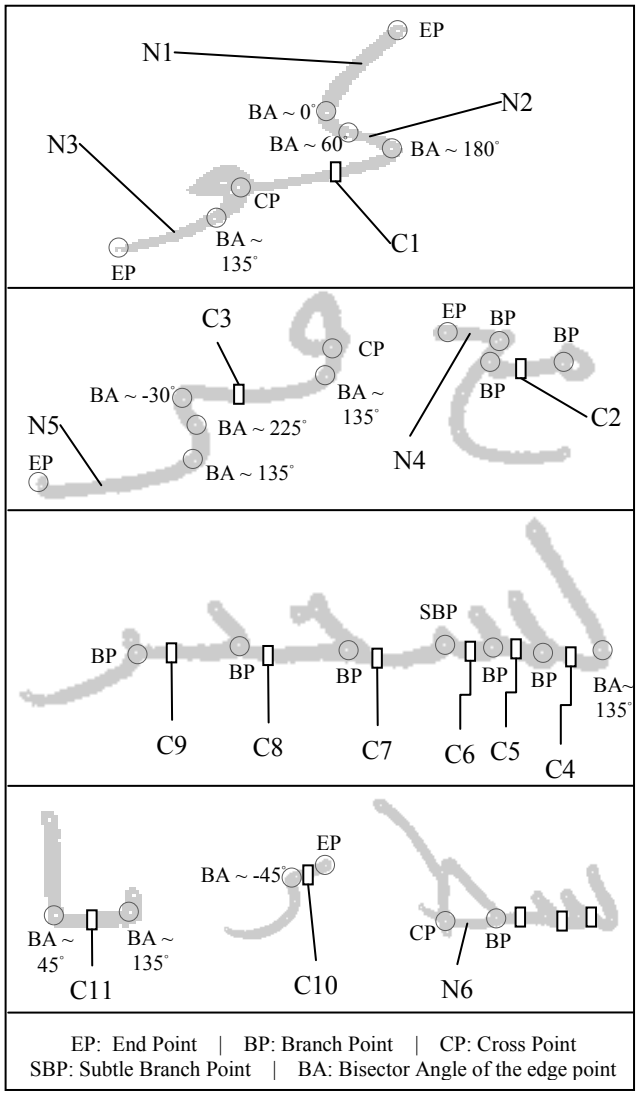


Figure 8. Segmentation examples

We search for the cut location starting after the first left forth of the continuity. The first point in a horizontal small segment of the continuity is the cut point. If the cut point happens to cut a letter (by inspecting the stroke width at this point), the cut point is shifted to the point where the stroke width is minimum. If the continuity has no horizontal segments, its midpoint is selected as the cut point. This cut point selection method enhances the accuracy of secondary bodies assignment.

In the segmentation rules described above, we included the rules for the angle points to avoid unpredictable over-segmentation. Using these rules, a single letter's segments are narrowed and thus the resulting graphemes are identified.

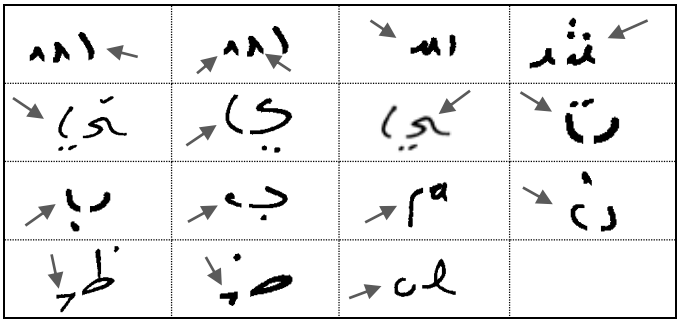
5) Grapheme Separation

In this step, the secondary bodies are reassigned to the new main bodies of the graphemes. The same assignment algorithm used in Subsection II.A.3 is used here also. Note that we assign an entire secondary body group to one grapheme because these secondary bodies usually belong to the same letter and we want to avoid distributing them over multiple letters. In Fig. 2, note that the two dots of the second letter from right, the medial **Yeh** (ـيـ), is separated as one grapheme along with its two dots.

C. Recognition and Post-processing

In this stage, the extracted graphemes are recognized and mapped to letters and words. Our set of graphemes includes the graphemes of the four forms of the Arabic letters (isolated, initial, medial, and final), graphemes of over-segmented letters (shown in Table I), and graphemes of the vertical ligatures.

TABLE I. GRAPHEMES OF OVER-SEGMENTED LETTERS



1) Grapheme Recognition

We use Tesseract to recognize the extracted graphemes [12]. This is possible because Tesseract has training features. To train Tesseract on these graphemes, we segment the training words using our segmentation algorithm described above, specify the code of each segmented grapheme, and run Tesseract training programs. We use grapheme codes similar to the grapheme codes used in the IFN/ENIT database [11]. Fig. 2 shows that the output of the grapheme recognition step is 12 grapheme codes.

2) Graphemes to Characters

Graphemes are mapped to characters using a table of the possible grapheme combinations that produce every character. Each combination has a weight value. The grapheme combination that has one grapheme has low weight. This compels a grapheme to be combined with adjacent graphemes, if possible. The maximum number of graphemes that produce one character is four. The algorithm maps the graphemes of every word to characters sequentially. For each grapheme, the algorithm checks for the existence of combinations made up of 1, 2, 3, or 4 graphemes in the mapping table that include this grapheme. And then it maps these combinations to the corresponding characters with their weights. Finally, the characters of the highest weight are chosen to form the word.

3) Word Matching

This step is used to improve the recognition accuracy when the number of words is limited. Generic spell-checking engines find the closest match to an erroneous word from a list of possible words. Closest words are found through letter additions, deletions, and replacements and validating the speculated words in the list. A distance function is used to measure the degree of similarity between the word and valid speculated words. The speculated word that has the smallest distance is chosen as the correct word [18]. We use Hunspell implementation of this algorithm for its neat programming interface and its ability to give each character a set of preferred

replacement candidates [19]. This is very useful in Arabic OCR because we can easily suggest good replacement candidates for characters that are likely to be incorrectly recognized. For example, the final **Reh** (ر) is often written similar to the final **Lam** (ل) and thus is often miss recognized. The spell-checking engine can solve such problems as shown for the leftmost underscored letter in Fig. 2.

III. RESULTS

We evaluated our approach using 107 samples from the IFN/ENIT database of two writers. Table II summarizes the results obtained. Through inspection, we found that the segmentation algorithm produces the expected graphemes accurately. More than 96% of the samples are correctly segmented, only one sample is under segmented, and three samples suffer over segmentation. For the under-segmented sample, the algorithm doesn't segment two horizontally adjacent letters. The over-segmented samples have letters that are over segmented beyond the expected segmentation described in Subsection II.C.

TABLE II. SUMMARY OF RESULTS

Measure	Count	Percentage
Total words	107	100%
Under-segmented words	1	1%
Over-segmented words	3	3%
Total characters	882	100%
Characters correctly recognized	763	86.5%
Words correctly recognized	101	94%

To evaluate the recognition accuracy, we randomly selected 96 samples out of the 107 samples to train Tesseract's recognition engine. About 87% of the 882 characters included in the 107 samples are correctly recognized through the recognition engine and the grapheme to character mapping step. The recognition accuracy is further improved through the word matching step to more than 94% of the words. Note that the size of the IFN/ENIT corpus is 937 words.

However, when we increase the number of writers and the number of training samples, the accuracy of the recognition engine severely drops to low levels. As Tesseract is designed for recognizing printed text, it seems that it is unsuitable for recognizing handwritten shapes of high variability.

IV. CONCLUSIONS

We have built a system for recognizing handwritten Arabic script using efficient skeleton-based grapheme segmentation algorithm. By modifying an open-source OCR system, we were able to evaluate the accuracy of this segmentation algorithm. Our preliminary results are encouraging showing segmentation accuracies above 96% and competitive to the state-of-the-art work. We believe that this algorithm solves problems found in other algorithms such as baseline estimation accuracy, pitted and slanted strokes, and subtle and long letter connections.

Since the used OCR system doesn't give good results for large number of writers, we plan to drop this system and use other feature extraction and grapheme classification techniques that are more suitable for handwritten Arabic script.

REFERENCES

[1] M. Khorsheed, "Off-line Arabic character recognition - a review," Pattern Analysis and Applications, vol.5, no.1, pp. 31–45, 2002.

[2] V. Märgner, and H. El Abed, "ICDAR 2009 Arabic handwriting recognition competition," Proc. 10th Int'l Conf. on Document Analysis and Recognition (ICDAR'09), Catalonia, Spain, 2009, pp. 1383–1387.

[3] D. Motawa, A. Amin, and R. Sabourin, "Segmentation of Arabic cursive script," Proc. 4th Int'l Conf. on Document Analysis and Recognition (ICDAR'97), Ulm, Germany, 1997, pp. 625–628.

[4] M. Mansour, M. Benkhadda, and A. Benyettou, "Optimized segmentation techniques for Arabic handwritten numeral character recognition," Proc. 1st Int'l Conf. on Signal-Image Technology and Internet-based Systems (SITIS'05), Yaoundé, Cameroon, 2005, pp. 96–102.

[5] T. Sari, L. Souici, and M. Sellami, "Off-line handwritten Arabic character segmentation algorithm: ACSA," Proc. 8th Int'l Workshop on Frontiers in Handwriting Recognition (IWFHR'02), Ontario, Canada, 2002, pp. 452–457.

[6] L. Lorigo, and V. Govindaraju, "Segmentation and pre-recognition of Arabic handwriting," Proc. 8th Int'l Conf. on Document Analysis and Recognition (ICDAR'05), Seoul, Korea, 2005, pp. 605–609.

[7] P. Xiu, L. Peng, and X. Ding, "Multi-queue merging scheme and its application in Arabic script augmentation," 2nd Int'l Conf. on Document Image Analysis for Libraries (DIAL'06), Lyon, France, 2006, pp. 24–29.

[8] R. Bentrcia and A. Elnagar, "Handwriting segmentation of Arabic text," Proc. 5th IASTED Int'l Conf. on Signal Processing, Pattern Recognition, and Applications, Innsbruck, Austria, 2008, pp. 122–127.

[9] S. Wshah, Z. Shi, and V. Govindaraju, "Segmentation of Arabic handwriting based on both contour and skeleton segmentation," Proc. 10th Int'l Conf. on Document Analysis and Recognition (ICDAR'09), Catalonia, Spain, 2009, pp. 793–797.

[10] F. B. Samoud, S. S. Maddouri, and N. Ellouze, "A hybrid method for three segmentation level of handwritten Arabic script," Proc. Int'l Workshop on Multilingual OCR (MOCR'09), Barcelona, Spain, 2009, pp. 1–6.

[11] M. Pechwitz, S. Snoussi Maddouri, V. Märgner, N. Ellouze, and H. Amiri, "IFN/ENIT-database of handwritten Arabic words," Proc. 7th Colloque Int'l Francophone sur l'Ecrit et le Document, (CIFED'02), Hammamet, Tunis, 2002, pp. 129–136.

[12] R. Smith, "An overview of the Tesseract OCR engine," Proc. 9th Int'l Conf. on Document Analysis and Recognition (ICDAR'07), Parana, Brazil, 2007, pp. 629–633.

[13] A. AbdulKader, "Two-tier approach for Arabic offline handwriting recognition," Proc. 10th Int'l Workshop on Frontiers in Handwriting Recognition, La Baule, France, 2006, pp. 161–166.

[14] S. S. Maddouri, F. B. Samoud, K. Bouriel, N. Ellouze, and H. El Abed, "Baseline extraction: comparison of six methods on IFN/ENIT database," Proc. 11th Int'l Conf. on Frontiers in Handwriting Recognition (ICFHR'08), Montréal, Québec, Canada, 2008.

[15] G. Abandah, and M. Khedher, "Analysis of handwritten Arabic letters using selected feature extraction techniques," Int'l Journal of Computer Processing of Languages (IJCPOL), vol. 18, no.1, pp. 49–73, 2009.

[16] F. Menasri, N. Vincent, M. Cheriet, and E. Augustin, "Shape-based alphabet for off-line Arabic handwriting recognition," Proc. 9th Int'l Conf. on Document Analysis and Recognition (ICDAR'07), Parana, Brazil, 2007, pp. 69–73.

[17] E. S. Deutsch, "Thinning algorithms on rectangular, hexagonal, and triangular arrays," Communications of the ACM, vol. 15, no. 9, pp. 827–837, 1972.

[18] V. J. Hodge, and J. Austin, "A comparison of standard spell checking algorithms and a novel binary neural approach," IEEE Trans. on Knowledge and Data Engineering, vol. 15, no. 5, pp. 1073-1081, 2003.

[19] Hunspell Official Website, http://hunspell.sourceforge.net.