Multi-Oriented Touching Text Character Segmentation in Graphical Documents using Dynamic Programming

Partha Pratim Roy[a], Umapada Pal[b], Josep Lladós[a], Mathieu Delalandre[c]

[a]*CVC, Universitat Autonoma de Barcelona, Spain*
[b]*CVPR Unit, Indian Statistical Institute, India*
[c]*LI Laboratory, François Rabelais University, France*

Abstract

The touching character segmentation problem becomes complex when touching strings are multi-oriented. Moreover in graphical documents sometimes characters in a single touching string have different orientations. Segmentation of such complex touching is more challenging. In this paper, we present a scheme towards the segmentation of English multi-oriented touching strings into individual characters. When two or more characters touch, they generate a big cavity region in the background portion. Based on the convex hull information, at first, we use this background information to find some initial points for segmentation of a touching string into possible primitives (a primitive consists of a single character or part of a character). Next, the primitives are merged to get optimum segmentation. A dynamic programming algorithm is applied for this purpose using the total likelihood of characters as the objective function. A SVM classifier is used to find the likelihood of a character. To consider multi-oriented touching strings the features used in the SVM are invariant to character orientation. Experiments were performed in different databases of real and synthetic touching characters and the results show that the method is efficient in segmenting touching characters of arbitrary orientations and sizes.

Keywords: Touching Character Segmentation, Multi-oriented Character Recognition, Dynamic Programming

1. Introduction

As electronic media becomes more and more accessible, the need for transferring offline documents to the electronic domain grows. Optical Character Recognition (*OCR*) works by scanning source documents and performing character analysis on the resulting images giving a transcription to ASCII text which can then be stored and manipulated electronically like any standard electronic document. As part of the OCR process, character segmentation techniques are applied to word images before individual characters images are recognized. The simplest way to perform character segmentation is to use the small space between characters as segmentation regions. This strategy fails when there are touching or broken characters, which often occur in degraded text images. Some examples of such documents are photocopies, faxes, historical documents, etc. and they are often degraded due to compression, bilevel conversion, aging or poor typing [3, 28]. In these situations, two or more characters may be segmented as one character component or one character may split into multiple pieces. Due to degradation, adjacent characters in a word touch together and they share common pixels in touching regions [27].

Besides the huge amount of documents having only horizontal direction text, there are many graphical documents such as maps, engineering drawings, etc. or artistic documents, where text lines appear frequently in different orientations other than usual horizontal direction. The purpose of such orientation and curvi-linearity is to catch people's attention at some particular words/lines or to annotate the location of graphical objects. Thus, a single document may contain strings with different inter-character spacing in the strings due to the annotation, style, etc. Also, the curvi-linear nature of the text makes the orientations of characters in a string different. As a result, it is difficult to detect the skew of such strings and hence character recognition of such documents is a complex task.

Segmentation of touching components is one of the difficulties to get higher recognition rates by OCR systems. The OCR systems available commercially do not perform well when words are multi-oriented in fashion in a document. When touching occurs in multi-oriented documents (e.g. artistic or graphical documents), it is much more difficult to segment such multi-oriented touching than touching segmentation of normal horizontal touching. Touching in curvi-linear string leads to false character segmentation and hence wrong recognition result occurs.

Text-lines could appear at different directions in the same document as illustrated in Fig.1. It can be seen from Fig.1(a), the word "PLANET" contains a touching string "LANE" of four characters. In Fig.1(b), we show a map where many characters in the word "Mayurakshi" are touching and they are oriented in different directions, although they belong to a same word. Orientation of two touching strings "ON" and "RE" of Fig.1(c) are perpendicular to each other. In Fig.1(d), it may be noted that orientations of "es" and "no" in the word "Couesnon" are not the same and such strings create difficulty for segmentation.

1.1. Related Work

There are many published papers towards the recognition and segmentation of the touching characters of horizontal direction [2, 15, 19, 30] and they are briefly reviewed here.

Among the earlier pieces of work on touching character segmentation, one class of approaches uses contour features of the connected components for segmentation [7, 15, 29]. When analyzing the contour of a touching pattern, valley and crest points are derived. Next, a cutting path is decided to segment the touching pattern by joining valley and crest points. Kahan et al. [13] used projection profiles as the objective function for touching character segmentation. They used the idea of joining adjacent characters that have minimum vertical projection. The segmentation function is calculated from the ratio of the second derivative of the projection-profile curve to its height. Later, Lu [18] introduced a peak-to-valley function to improve the segmentation approach. Fujisawa et al. [8] used profile features for touching numeral segmentation. Upper and lower profiles of the connected component are computed and the distance between upper and lower profiles are analyzed to detect the segmentation points.

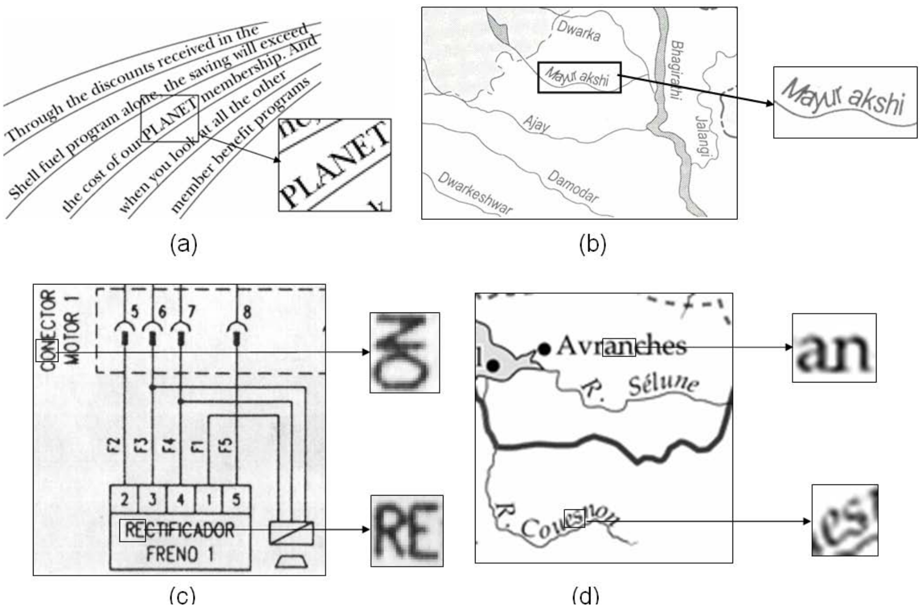


Figure 1: Example of documents showing multi-oriented touching characters in (a) an advertisement, (b) and (d) maps, and (c) electrical diagram.

Afterwards, Liang et al. [17] proposed discriminating functions for machine printed touching character segmentation. Pixel projection and profile projection techniques are employed as discrimination functions for solving heavily touching printed characters. Next, they applied forward segmentation along with a backward merge procedure based on the output of a character classifier. It works on the components generated by discriminating functions. Yu and Yan [32] presented a segmentation technique using structural features for single-touching hand-written numeral strings. At first, the touching region of the character components is determined based on its structural shape. Next, a candidate touching point is pre-selected using the geometrical information of special structural shapes. Finally, morphological analysis and partial recognition results are used for the purpose of segmentation. Dimauro et al. [5] applied contour based features along with some descending algorithms for the touching character segmentation purpose.

Another class of approaches is based on thinning [3, 19]. In these approaches, thinning of foreground and/or background pixels of the connected pattern are processed. End and fork points obtained by thinning are used for cutting points extraction. These methods are time consuming and in addition they generate protrusions. These protrusions sometimes give wrong results because they bring some confusions among the actual fork and end points.

A water reservoir based technique [21] is employed to locate inter-character spaces in touching numeral strings. Water reservoir is a metaphor to illustrate the cavity region of a component. In this sense, if water is poured from a side of a component, the cavity regions of the background portion of the component where water will be stored are considered as reservoirs of the component. Based on the size of water reservoirs, the segmentation zones of the touching string are selected. Next, segmentation is done using structural information of these reservoirs.

Yong et al. [31] proposed an approach using supervised learning on the labeled examples and a Markov Random Field (MRF) approach has been applied for this purpose. Further, a propagation minimization method is employed to select the candidate patches based on the compatibility of the neighbor patches. The output of the MRF after the iterative belief propagation forms a segmentation probability map. Finally, the cut position is extracted from the map. An accuracy rate of 94.8% is reported.

Methods based on combinations of features have also been used for touching segmentation. Oliveira et al. [20] used contour, profile and skeleton features to find a set of points for touching characters segmentation. First, local minima of contours and profile features are defined as basic point (BP). Second, a point with more than two pixels in its neighborhood is defined as an intersection point (IP). Afterwards, an Euclidean distance scheme is applied to determine proximity between IP and BP for segmentation.

The state-of-the-art approaches of touching character segmentation generally consider touching of characters in horizontal text strings. These methods assume the characters of strings are aligned horizontally and thus segmentation features are devised for such characters in horizontal strings. Also, the features used in most of the approaches for text character recognition are generally not rotation invariant. The characters along a touching portion may be in different orientations with respect to the baseline of the word. In graphical documents when characters touch, it is difficult to know the angle of alignment of characters in the touching regions. Moreover in Fig.1(b), we show examples where the characters in a single touching have different orientations. As a result, skew correction methods cannot make such touching horizontal and hence the methods that take care of horizontal touching cannot be used. For segmentation purpose, we need technique that can take care of size and rotation invariant touching strings. Hence, we propose here a segmentation approach that can handle touching strings in multiple orientations. Recently, we proposed a touching character segmentation approach in ICDAR-2009 [25] and the present work is its extended version. This paper elaborates the different steps of character segmentation method. Also, extensive experiments including comparative study are included in this version to prove the efficiency of this method.

1.2. Outline of the Proposed Approach

As mentioned earlier, many techniques are available for segmentation of horizontal touching characters but to the best of our knowledge there is no work towards multi-oriented touching character segmentation except our work. In this present paper, we propose an approach for multi-oriented n-character touching string segmentation scheme. The block-diagram of our approach is shown in Fig.2. The different steps used in this system are discussed as follows.

Recognition process: An important step in our system is the recognition of the isolated character. As we consider multi-oriented graphical document, features used in our system must be rotation invariant. Circular and convex hull ring based zoning approach has been used along with angular information of the contour pixels of the character to make the feature rotation invariant. A SVM classifier is used to find the likelihood of a character. The C1 and C2 modules of Fig.2 discuss about recognition.

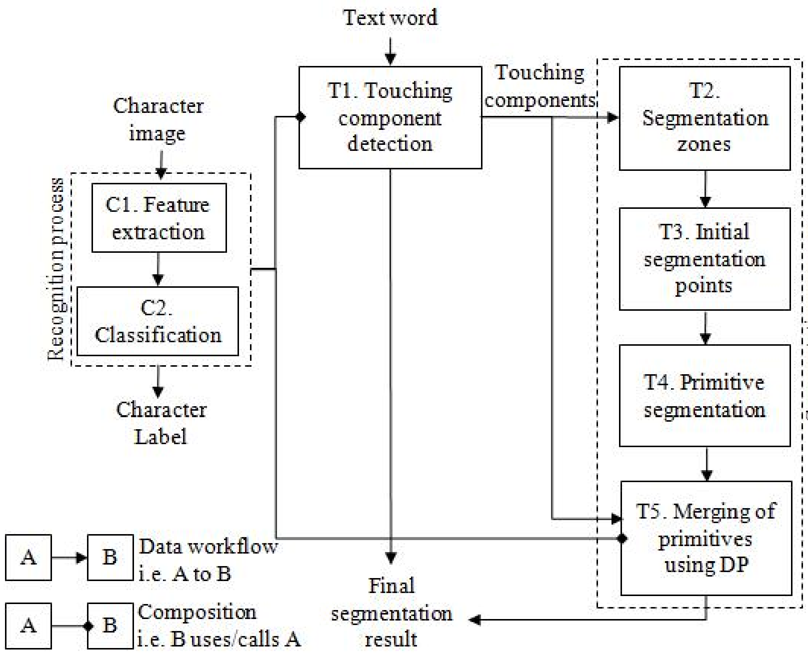


Figure 2: Block diagram of the proposed approach for touching character segmentation.

Touching component detection: There may exist touching or non-touching characters in a word. Before passing a component into our segmentation process (steps T2-T5 of Fig.2), we detect if the component is touching or isolated. We apply a Connected Component (CC) labeling to the word image and extract individual components. For each component, we compute the recognition confidence for all character class models using our recognition process (steps C1-C2). Based on this recognition score, the components are separated into touching and non-touching components. The touching components are processed next for segmentation. This module is noted by T1 in Fig.2 and discussed later.

Segmentation zones: When two or more characters touch, they generate a big cavity region at the background portion. This background portion is used to detect the segmentation zones. To handle the background information of a multi-oriented string, properties of the convex hull of the touching string have been used. This process is marked by T2 in Fig.2.

Initial segmentation points: The segmentation zones are used to find the segmentation points. A set of initial segmentation points are calculated in the contours of convex hull residua using the Douglas Peucker polyline approximation. This is denoted by T3 in Fig.2.

Primitive Segmentation: Next, segmentation lines are calculated from the initial segmentation points of the touching character. Based on these segmentation lines, the touching string is segmented into primitives. A primitive consists of a single character or a part of a single character. This step is mentioned by T4 in Fig.2.

Merging of primitive segments using dynamic programming: Some of the primitives obtained before are merged to get optimum segmentation. To do this, dynamic programming algorithm is applied using total likelihood of characters as the objective function. Based on the recognition rates of primitive segments, multiple hypothesis of segmentation are generated. A dynamic programming (DP) algorithm is applied to get the optimal solution for the touching character segmentation. This step is marked by T5 in Fig.2.

As discussed earlier, the main contribution of this paper is the multi-oriented n-character string segmentation for its recognition (i.e. steps T2-T5 in Fig.2). However, it is difficult to dissociate this part from the recognition process. So, we will present recognition procedure briefly before detail discussion of touching character segmentation.

The rest of the paper is organized as follows. In Section 2, we explain the feature extraction procedure as well as recognition for handling characters in multi-scale and multi-oriented environments (steps C1 and C2 of Fig.2). In Section 3, we present the proposed segmentation approach for n-touching strings (steps T1-T5 of Fig.2). Next, data details and the different experimental results are discussed in Section 4. Finally, the paper is concluded in Section 5.

2. Feature Description and Recognition of Text Character

Recognition of text characters in multi-rotation and multi-scale environment is a challenging task. Recognition of individual characters in multi-oriented and multi-sized environment drives the segmentation hypothesis of n-touching characters in our system. In the literature different shape descriptors like Angular Radial Transform (ART) [23], Hu's moments [12], Zernike moments [14], Fourier-Mellin [1], Angle based features [22] etc. are proposed for recognition and they are invariant to rotation, translation and scale. We noted that an angle based feature provides best performance among different rotation invariant shape descriptors. Because of the highest performance we have used an angle based feature for our work. The computation of such feature is briefly described in the following subsections.

2.1. Feature Descriptor

Our *Angle based feature* descriptor is a zone-wise feature descriptor to describe symbol/text characters. It is based on the histogram of angular information of the external and internal contour pixels of the characters. The relative angles obtained from all the contour pixels of a character are grouped into bins. Here, we consider 8 bins ($360°/45°$) of angular information.

To obtain local relative angle information, circular ring and convex hull rings are constructed. A set of circular rings is defined as the concentric circles considering their center as the center of Minimum Enclosing Circle (MEC) of the character and the minimum enclosing circle is the outer ring of the set. Similarly, convex hull rings are also constructed from the convex hull shape of the character. Angular slope of the contour pixels with respect to the center of the MEC is also included as rotation invariant features. The slope of contour pixels for each bin is computed and grouped into different sets. Details of feature explanation can be found in [22], so we are elaborating it here.

Considering 7 circular rings and 7 convex hull rings, we have 56 (8 relative angular bins $\times$ 7 convex hull rings) features from the convex hull ring, 56 (8 relative angular bins $\times$ 7 circular rings) features from the circular ring and 64

(8 relative angular bins × 8 sets of angular slopes) features from angular slope with respect to the center of MEC. As a result, we have a 176 (56+56+64) dimensional feature vector for the classification. The numbers of bins, rings and sets have been selected based on the experiment. To obtain scale invariance, the feature vector is normalized. The feature vector is divided by the total number of contour pixels for this purpose. We have shown the plot of 176 dimensional features of characters between intra-class and inter-class characters in Fig.3. From the figure it can be seen that the corresponding features of same character classes are similar although the characters are multi-oriented.

2.2. Character Recognition

A Support Vector Machine (SVM) has been used to build the isolated character models. We employed the SVM software package *libSVM*[1] for this purpose. A Gaussian kernel with Radial Basis Function (RBF) has been chosen in our system to recognize multi-oriented text characters. Feature learning is done with datasets of multi-oriented characters to generate the text character models.

During the training process, the SVM generates character models according to pre-segmented text characters. After training, when an unknown character or a primitive segment is fed to this SVM classifier, the SVM provides its class label along with its weight. The value of the weight lies between 0 to 1. We consider this weight as recognition confidence and this value is used as the cost function in our dynamic programming approach for correct segmentation of multi-oriented touching characters.

3. Touching Character Segmentation

A touching component segmentation approach for two touching characters in a multi-oriented environment was presented in [24]. This segmentation was performed with the knowledge of the number of total characters in the touching component. To do this, touching components of only 2 characters were collected and the method was based on cavity regions of the background portion. The convex hull was used to find the cavity regions. Next, several hypothesis of segmentation lines were computed from these cavity regions. Each of these segmentation lines divided the touching component into 2 parts. Finally, all pair-wise segmented parts were fed to the SVM classifier to find the correct segmentation. The best segmentation line was selected based on the highest accumulated recognition confidence of the two parts of the touching component.

Continuing with this idea, a touching component of n-characters could be segmented if the number of character in the touching component were known before. This concept is restricted because we have to know the number of characters in the touching component apriori. It is a hard constraint, since it is not always possible to know the number of characters in a real touching component in a multi-scale and multi-rotation environment. Because of this, here in this paper, we propose an optimization algorithm to segment n-character touching components.

If a n-character string or word is rotated to a certain angle, we estimate a rough angle from the minimum rectangular bounding box of the string. We

[1] http://www.csie.ntu.edu.tw/ cjlin/libsvm/

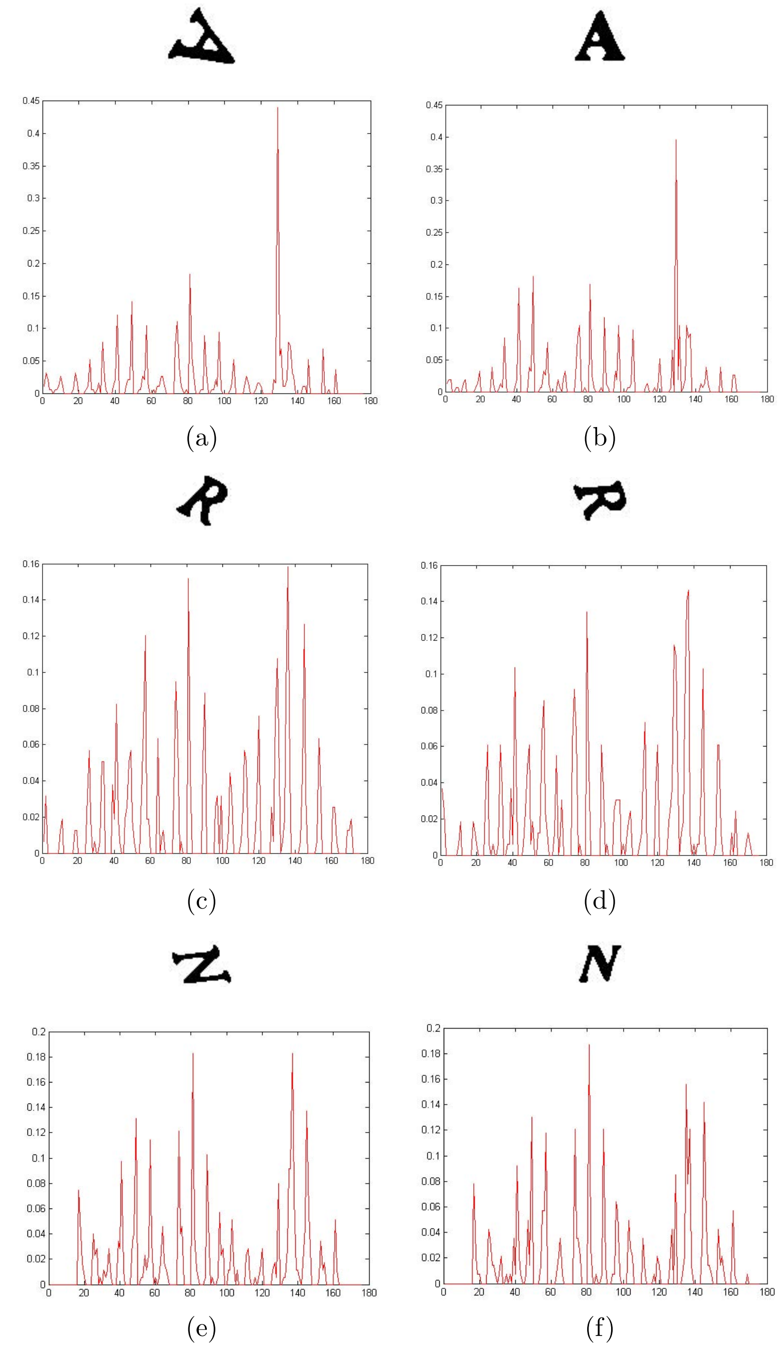


Figure 3: Feature vectors are plotted for different orientations of characters 'A', 'R' and 'N'. (Here, X axis (horizontal) denotes the feature vector and Y axis (vertical) denotes their values.)

compute the bounding box of the word and find the angle (α) of the major axis. Better approximation of the angle can be obtained if a word contain more number of characters. An approximate height of the word (H_w) is found from its bounding box (See Fig.4). It is to be noted that, when there are few characters in the word and it includes characters having ascenders and descenders, α indicates an approximated angle of the inclination of the word. In Fig.4, we show a multi-oriented word along with its bounding box and α and H_w are marked in this figure. It can be noted that if this word is rotated by α then all the components of the word will not be in horizontal mode. Hence, existing approaches of horizontal touching character segmentation can not be applied in such string after rotating it by α. Given a touching string of unknown orientation, our rough inclination angle α is used to arrange the primitive segments of the touching component in a sequential order, such that a dynamic programming algorithm can be applied to merge some of the primitive segments for proper segmentation.

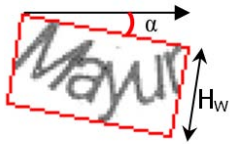


Figure 4: A multi-oriented word and its bounding box are shown. Here, α indicates the rough angle of inclination and H_w is the height of the bounding box.

Our proposed segmentation method is divided into five main steps (See Fig.2): touching component detection (T1), segmentation zones (T2), initial segmentation points (T3), primitive segmentation (T4) and merging of primitive segments using DP (T5). Details of the segmentation method are discussed in the following subsections.

3.1. Touching Component Detection

There may exist touching or non-touching characters in a word. A component is detected as touching or isolated before applying the segmentation approach on the touching string. For this purpose, at first, a Connected Component (CC) labeling is applied to extract individual components of the word. For each component, we compute the recognition confidence for all character class models using SVM and rank their confidence scores in descending order. If a component is recognized by the SVM with a high accuracy (more than 0.4), we assign it as a non-touching or isolated character. If the difference between the top two recognition scores of a component is high, it is also considered as an isolated character. The rest of the components are considered as touching. These touching components are processed for segmentation using our approach. We may get some false positive labelling due to such separation based on confidence score. For example, the difference between the top two recognition scores may be less for characters like 'D' and 'O'. Such mislabelled components do not affect the final segmentation result because the proposed dynamic programming approach takes care of such errors with the final optimal score.

3.2. Segmentation Zones

When two or more characters touch, generally they generate big cavity regions at the background portion between touching components. When components in a string are in the horizontal direction, the water reservoir concept can be used to find these cavity regions [21]. Water reservoir based algorithms may not be applied in the strings of multi-oriented nature. We have considered some properties of the convex hull to take care of this problem.

The Convex Hull [10] or convex envelope for a set of points X in a real vector space V is the minimal convex set containing X. In another way, a convex hull is a minimal convex shape entirely bounding an object. Some of the properties of convex hull residuum are described as follows.

1. Residuum area (R_A): The area of a residuum is defined by the number of pixels inside the residuum.
2. Residuum surface level (R_{SL}): The R_{SL} of a residuum is the line obtained by joining two endpoints of the open face of the residuum.
3. Residuum border pixels (R_{BP}): The border pixels of each residuum are defined as the contour pixels of the residuum excluding the R_{SL} pixels.
4. Residuum height (R_H): It is the depth of the farthest residuum border pixel from R_{SL}.

In Fig.5, convex hull residua and their different parameters for a text character 'S' are shown.

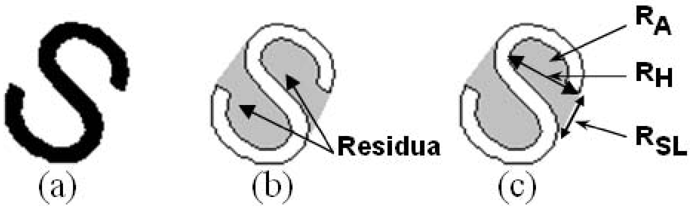


Figure 5: (a) Image of the character 'S'. (b) Two residua from the convex hull of 'S'. (c) Different parameters of convex hull are shown in a residuum.

Similarly, the cavity regions of the touching component are determined by finding residua of that component through convex hulls. These residua cover the cavity regions of the touching component and thus, they are used to determine the segmentation zone of touching characters. The residua found from the convex hull of a touching character are shown in Fig.6. For this touching component ("72"), we find a total of four cavity regions.

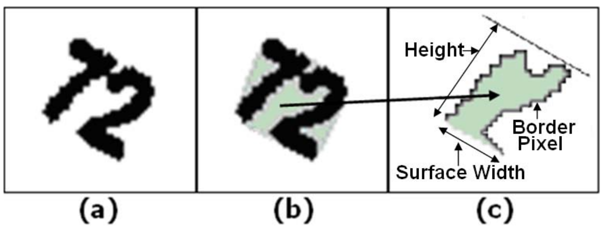


Figure 6: (a) Touching character. (b) The residua of the convex hull are shown and marked by grey shade. (c) Different parameters of the residuum.

Given a touching component, we may find many small cavity regions along with the segmentation zones due to the degradation of contour of the characters. Even, the presence of "Serif" in some fonts of text characters (e.g. Times New

Roman) also produces small cavity regions. These small cavity regions are considered as noise and thus, these are not considered for segmentation in this approach. To do that, the residua having height more than stroke-width are considered for segmentation purpose. The stroke-width (St_w) of the word is the statistical mode of object pixels' run lengths [2]. For a component, St_w is calculated as follows. The component is scanned row-wise (horizontally), column-wise (vertically) and then in two diagonal directions (45° and 135°). If rl different runs of lengths $r_1, r_2 \ldots r_{rl}$ with frequencies $f_1, f_2 \ldots f_{rl}$, respectively are obtained by the scanning the component, then value of St_w will be r_i if $f_i = \max(f_j)$, j = 1, 2 $\ldots rl$.

3.3. Initial Segmentation Points

To segment a touching string into possible primitive segments, segmentation points are next computed from the segmentation zones extracted previously. To do it, we employ a polygonal approximation method to the contour pixels of residuum borders. Polygonisation provides key-points which are at the corner of edges in the corresponding segmentation zones. Among existing algorithms of the literature [16], we have selected the Douglas and Peucker [6] polygonal approximation algorithm. A short presentation of this algorithm is given in Appendix A. This algorithm is well adapted to localize hard curvature points along a border.

The two end points of the residuum surface level (R_{SL}) of each selected residua are regarded as the initial rough estimate of the polyline. Using this initial guess, the other vertices are approximated using a tolerance threshold ϵ. The value of this tolerance threshold is selected with stroke width (St_w) precision. After approximation, the list of vertices are treated as key-points. The advantage of using polygonal approximation is that it provides the key-points which are at the corner of the edges in the corresponding segmentation zones. These key points are necessary for touching character segmentation because, usually when the characters touch, they form a corner at the touching region.

We have noted that some of the key-points might appear very near to R_{SL} due to the appearance of hard curvature or degradation in the contours of these zones. These key-points do not provide the segmentation lines and thus are selected for removal. We remove such key-points based on the corresponding residuum height (R_H). The key-points which are having height more than ($0.5 \times R_H$) from the corresponding R_{SL} are kept for the initial segmentation points. Remaining key-points are considered as the initial segmentation points. In Fig.8(b), we have shown the initial segmentation points for the image Fig.8(a). Similarly, these initial segmentation points are shown for a touching component of fonts having serifs in Fig.7.

Figure 7: A touching component of characters "UT" is shown with its initial segmentation points. These initial segmentation points are marked in dark gray (red in pdf).

3.4. Primitive Segmentation

Once we get initial segmentation points, for each initial segmentation point (S_i) we find another point (S_j) through which the touching component can be cut into 2 parts. Computation of S_j is done as follows. We know the angle of the direction of the R_{SL} with the horizontal axis. For each initial segmentation point, we find a segmentation line passing through this point and perpendicular to the respective R_{SL}. This line segments a touching component at the point S_i. The perpendicular line to the R_{SL} generally gives us a clue to the direction of the segmentation line. We draw a line from the point S_i in the opposite side of R_{SL} and perpendicular to the R_{SL} until it passes through the object pixels. Let, the last object pixel on this line be S_c. The line from S_i to S_c may be considered as a segmentation line. This segmentation line may not give the best segmentation always and hence to get better segmentation the point S_c is tuned to be a better segmentation point. This tuning is done by considering some neighbor contour points of the point S_c. To get neighboring pixels, the contour is traced upto a length of stroke-width (S_w) clockwise and anti-clockwise starting from S_c. These traced pixels are neighbor pixels. The neighbor pixel having the minimum distance from S_i is chosen as S_j. The line obtained by joining S_j and S_i is the segmentation line and the distance between S_j and S_i is called the length of segmetnation line. We explain the segmentation line computation process in Algorithm 1.

Algorithm 1 Segmentation Lines of Touching Component

Require: Touching component (C_T)
Ensure: A set of segmentation lines from C_T
  Compute convex hull of C_T and find the residua.
  //create a list (L_S) of segmentation lines
  $L_S \Leftarrow \oslash$
  for all residua R_k of C_T **do**
    Generate the initial segmentation points using polyline approx. in the contour of R_k
    for all initial segmentation points S_i **do**
      Compute segmentation line (L_{ik}) at S_i perpendicular to R_{SL} of R_k and tune L_{ik}
      $L_S \Leftarrow L_S \cup L_{ik}$
    end for
  end for

It may happen that the touching portion of the components creates segmentation zones in both sides (top and bottom) of the characters. Such touching creates multiple hypotheses of segmentation points in both sides. These points generate segmentation lines for the components. As a result, some of the segmentation hypotheses may lie very close to other. To reduce the choices of hypotheses we remove some of the lines which are very closed. If the distance between two segmentation lines is less than St_w, the segmentation line with bigger length is not considered for segmentation. Moreover, if the length of a segmentation line is greater than $0.75 \times H_w$, that segmentation line is also not considered. This value is determined from the experimental result.

For each initial segmentation point we get the corresponding segmentation line. If we have n segmentation lines, the image is segmented into $(n + 1)$

sub-images. The (candidate) segmentation lines of Fig.8(a) have been shown in Fig.8(c). Note that, there were 7 initial segmentation points (See Fig.8(b)) and we have got 5 segmentation lines (these 5 segmentation lines are shown in Fig.8(c)). These segmentation lines split the touching component into primitive segments. In Fig.8(c) there are 6 primitive segments. These primitive segments are arranged in a sequence following the direction of angle α. Now, we will merge some primitive segments for correct segmentation. A dynamic programming technique is used for the purpose.

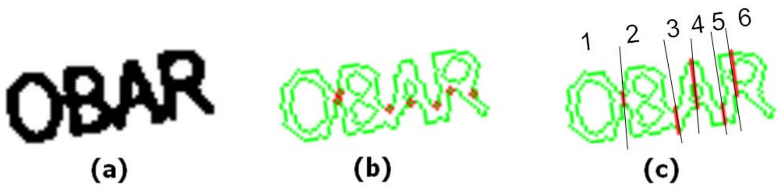


Figure 8: (a) A touching string of four characters. (b) Initial segmentation points found from concave residua. (c) Candidate segmentation lines and primitive segments obtained from selected segmentation points.

3.5. Merging of Primitive Segments using Dynamic Programming

The key idea behind dynamic programming (DP) is quite simple and the core of a DP algorithm [9] is the module that takes a set of symbols (list of primitive segments in our case) and a set of labels (possible characters) and returns the optimum assignment of labels to symbols assuming that an optimum assignment is the sum of the sub-assignments (sub-problems). DP is a very powerful algorithmic paradigm in which a problem is solved by identifying a collection of sub-problems and tackling them one by one, smallest first, using the answers to small problems to help figure out larger ones, until the whole lot of them is solved. The DP seeks to solve each sub-problem only once, thus reducing the number of computations. Given a touching image, the primitive segments are merged so that the average character likelihood is maximized using DP and in our case, the likelihood of each character is calculated using a recognition accuracy obtained by the SVM. Generally the order of time complexity of DP is $O(N \times M \times M)$, where N is the length of the touching string and M is maximum number of primitive segments for a character. From, our experiment we noted that a single character can take at most four primitive segments and hence, M reduces to four in our case.

To apply the DP algorithm, the primitive segments are arranged from left to right following the direction of α. Let $S_1, S_2, ..S_n$ be a list of n primitives. In Fig.8(c) the six primitive segments (n = 6) are indexed according to their sorting order. We use two tables to store the character likelihood of primitive segments after merging (see Fig.9). In the Score Table ST, we enter the classification score $\{c_{uv}\}$ and in the Label Table LT, we enter the character classification label $\{l_{uv}\}$ where $1 \leq u \leq v \leq n$. The classification scores and classification labels are defined as follows:

$$c_{uv} = \bigcup_{i=u}^{v} S_i \quad \texttt{, score from ST}$$

$$l_{uv} = \bigcup_{i=u}^{v} S_i \quad \texttt{, label from LT}$$

In these tables, the cells correspond to the recognition result of a cumulative grouping of primitive segments. The possible merging result of the primitive segments of Fig.8(c) are shown in Fig.9(a) and Fig.9(b). For example, in Fig.9(b), the cell l_{35} represents the character likelihood of merging the primitive segments S_3, S_4 and S_5. The label obtained by our SVM is 'm'. In table ST, the cell c_{35} indicates the corresponding classification score (0.183) to obtain the label 'm'. If the classification score of merged segments is very low (a threshold value of 0.1 is decided empirically), we do not consider it. Cumulation of primitive segments is continued until the width of the resultant image is less than $1.2\times H_w$. This value is chosen based on the size of the Latin alphabet. Also, characters like 'M' can be segmented into multiple hypothesis of '1'. So, if we find two or more consecutive character shapes of '1', 't', etc., we check the hypothesis of the combination of these shapes and based on the recognition confidence, the character is selected.

v (merged primitives)

u (starting primitive)

	S1	S2	S3	S4	S5	S6
S1	*0.969*					
S2		*0.856*	*0.705*			
S3			*0.566*	*0.380*	*0.183*	
S4				*0.266*	*0.645*	*0.245*
S5					*0.158*	*0.839*
S6						*0.103*

(a)

v (merged primitives)

u (starting primitive)

	S1	S2	S3	S4	S5	S6
S1	*O*					
S2		*B*	*a*			
S3			*t*	*A*	$\boxed{m}$	
S4				*t*	*n*	*m*
S5					*l*	*R*
S6						*2*

(b)

Figure 9: (a) Score Table ST and (b) Label Table LT of character string "OBAR".

Next, we check the total likelihood of the character groups. The group having the maximum likelihood is chosen and the corresponding merged primitives are their segmentation result. In Fig.8(c), the first segment corresponds to the letter 'O', the second segment corresponds to the letter 'B', the third and the fourth segments correspond to the letter 'A' and the fifth and sixth segments correspond to the letter 'R'. The assignment of primitive segments for the characters $O\ \ B\ \ A\ \ R$ is also represented by:

$i \rightarrow 1\ \ 2\ \ 3\ \ 4$ and $\quad j(i) \rightarrow 1\ \ 2\ \ 4\ \ 6$

where i denotes the letter number, j(i) denotes the number of the last primitive corresponding to the i-th letter. Note that the number of the first primitive segment corresponding to the i-th letter is $j(i-1)+1$. Given $j(i)$, (i = 1…n), the total likelihood of characters is represented by

$$L = \sum_{i=1}^{n} l(i, j(i-1)+1, j(i)) \tag{1}$$

where $l(i, j(i-1)+1, j(i))$ is the likelihood for the i-th letter. The optimal assignment (the optimal segmentation) that maximizes the total likelihood is found in terms of dynamic programming as follows. The optimal assignment $j(n)^*$ for the n-th letter is the one such that

$$L^*_{j(n)} = L(n, j(n)^*) = \mathtt{Max} L(n, j(n)) \tag{2}$$

where $L(k, j(k))$ is the maximum likelihood of partial solutions given $j(k)$ for the k-th letter. This is defined and calculated recursively by
$L(k, j(k))= \mathtt{Max}_{j(1),j(2)..j(k-1)} \sum_{i=1}^{k} l(i, j(i–1)+1, j(i))$

$$= \mathtt{Max}_{j(k-1)}[l(k, j(k-1)+1, j(k)) + L(k-1, j(k-1))] \tag{3}$$

$$\texttt{and} \quad L(0, j(0)) = 0 \texttt{ for } j(0) = 1, 2, ...m \tag{4}$$

Starting from (4), all $L(k, j(k))$'s are calculated for $k = 1, 2, ..., n$ using (3) to find $j(n)^*$ using (2). The rest of $j(k)^*$'s ($k = n-1, n-2, ..., 1$) are found by back tracking a pointer array representing the optimal $j(k-1)^*$'s which maximizes $L(k, j(k))$ in (3).

Given a segment group, the feature vector is calculated for a character class. Based on the character likelihood, the total likelihood of a word is found in terms of the dynamic programming technique discussed above. In Fig.10 we have shown the final segmentation result of the touching characters of Fig.8(a).

Figure 10: Final segmentation lines are drawn on the touching string shown in Fig.8(a) after applying our proposed approach.

4. Data Collection and Experimental Results

In this section we detail the performance of the proposed approach. First we describe the dataset generation and then we show experimental results of our touching character segmentation approach.

4.1. Dataset Generation

To the best of our knowledge, there exists no standard database to evaluate character segmentation methods in a multi-oriented and multi-size context. For our experiments, we have constituted our own database using real as well as synthetic data. Synthetic data is available online[1] for the use of other researchers.

The real data is collected from maps, newspapers and magazines. We have considered 10 different real geographical maps for evaluation of OCR in graphical documents. The average size of these map images are 1200×1200. There are approximately 35-40 words in each document. Documents are digitized in grey tone at 300 dpi and we have used a histogram based global binarization algorithm for their two tone conversion. A text separation method [26] has been used to extract characters from documents, and the groundtruth has been generated manually.

Synthetic data is generated from Arial and Times New Roman fonts. Some of them are shown in Fig.12(a) and (b). These datasets have been produced using the system described in [4]. The data are produced at first in vector graphics form with the corresponding ground-truth. Next, vector graphics data are rasterized to obtain the test images. The touching strings are composed of single-word images with different scales, orientations and fonts, with corresponding groundtruth at the character level. The words are selected from a dictionary (of 52 country names), with random scaling and rotation parameters. The average number of characters in the word is 7-8. In each word, this data generation method looks for the pairs of successive characters, and makes them connected according to a boolean value. The overlapping between characters is controlled using a Gaussian function.

In our experiment on touching character segmentation, we tested our scheme on 450 words (200 real and 250 synthetic). The synthetic data contains touching as well as non-touching characters. There were 880 touching components in these 450 words. Also we noted that 2050 characters touched in these 880 touching strings. The touching strings are of different sizes and orientations. Some of the data are up side down to check the rotation invariance nature of our method.

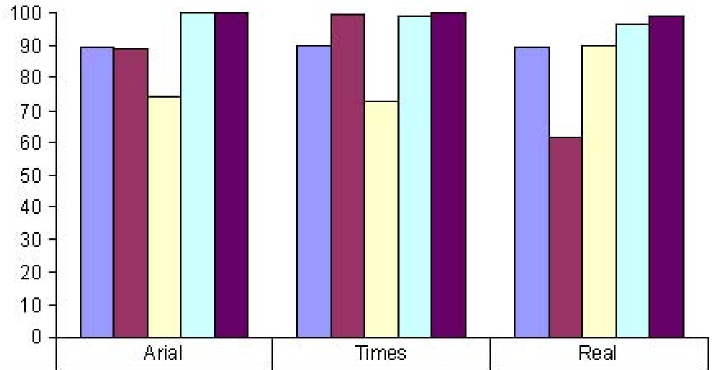
4.2. Isolated Character Recognition using Different Shape Descriptors

Isolated text recognition in a multi-scale and multi-oriented environment drives the solution towards the touching character segmentation problem. Text character recognition is a challenging task in such an environment. To overcome such problems, multi-scale and multi-rotation shape features are used as discussed in Section 2. A SVM classifier with RBF function is employed for isolated text character recognition.

In our experiment both English uppercase and lowercase alpha-numeric characters are considered, so we should have 62 classes (26 for uppercase, 26 for lowercase and 10 for digits). But because of shape similarity due to orientation some of the characters like 'd' and 'p'; 'b' and 'q'; etc. are grouped together. Hence, in our approach we considered 40 classes of character shapes. Different fonts of characters including Times New Roman and Arial have been used for the experiment.

[1] http://mathieu.delalandre.free.fr/projects/sesyd/charseg.html

A comparison is done with other rotation invariant feature descriptors used in the literature, namely: ART, HU, Zernike moments and Fourier Mellin. We considered three different sets of data consisting of multi-oriented and multi-scale text characters to perform this character recognition evaluation. One of the datasets is from graphical documents and its size is 8250. The groundtruth of this dataset has been generated manually for the performance evaluation. The other two datasets are synthetic data, constructed from Arial and Times New Roman fonts characters. The size of both of these datasets is 1850. The feature vectors obtained from different descriptors are passed to a SVM classifier to get the recognition accuracy. The classification is done with 5-fold cross-validation. Comparative results of different descriptors are shown in Fig.11. It is noted that angle based features (HU moments) perform the best (worst) among these descriptors to classify text characters.



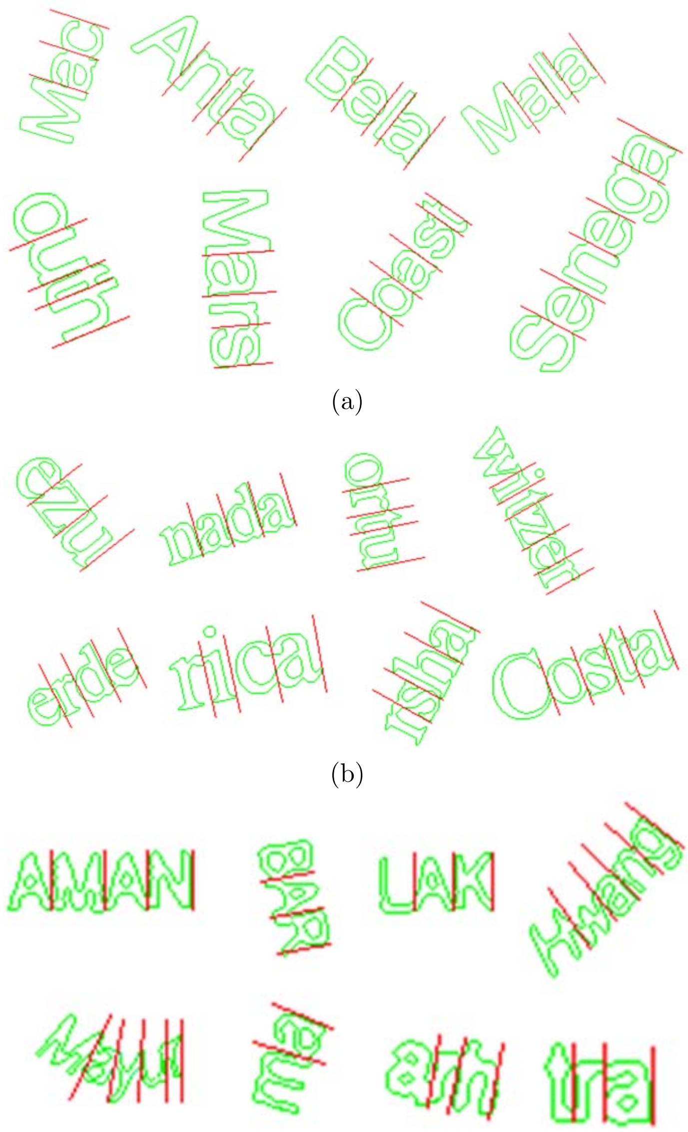
	Arial	Times	Real
ART	89,05	89,97	89,06
HU	88,79	99,57	61,71
Zernike	74,12	72,33	90,19
Fourier-Mellin	100,00	99,09	96,52
Angle-Based Feature	100,00	100,00	98,89

Figure 11: Text character recognition accuracy with different shape feature descriptors like ART, HU, Zernike, Fourier Mellin and Angle based feature.

4.3. Performance Evaluation of n-Touching Characters

In our experiment, here at first, we have provided some qualitative results to show how segmentation is done with our approach. Next, we evaluate the performance of our method in the datasets discussed in Section 4.1. To get an idea about the segmentation results, we have shown some touching images with their segmentation results in Fig.12. In Fig.12(c), some of the words are touching in a curvilinear fashion. Our method also segmented them correctly.

We compared the touching character segmentation results using 2 different multi-oriented text descriptors namely: angle based features and Fourier-Mellin moments. The Fourier-Mellin moment shape descriptor has been chosen due to its good performance in recognizing isolated multi-oriented characters [1]. We obtained 91.36% and 88.38% segmentation accuracy in overall experiment using angle based features and Fourier-Mellin, respectively. In Table 1, we provide



(a)

(b)

(c)

Figure 12: Some segmentation results of different datasets when angle based features are applied : (a) Arial font (b) Times New Roman font (c) Real data.

the accuracy of touching character segmentation based on the number of characters present in a touching component. We noted that, our system provides better results on 2-character touching strings than 3 or more character touching strings. Fig.13 provides the comparative results of different datasets using these two different features. It can be noted that angle based features provides better segmentation results than that of Fourier Mellin in all these datasets.

Error Analysis: Fig.14 shows some wrong segmentation of touching characters from our method. It is noted that most of the segmentation errors are due to following. (a) When a touching string can be segmented in more than two ways to get the valid segmented characters. For example, in Fig.14(a) the touching string was formed from the characters ‘r’ and ‘m’. But this touching string can be visualized as ‘r-r-n’ also, and our system segmented this string into ‘r’, ‘r’ and ‘n’ instead of ‘r’ and ‘m’ which we consider as erroneous. (b) The character shapes like ‘h’ (Fig.14(b)) may be split in two parts and our system segments this character into ‘t’,‘l’. We also considered it as wrong segmentation. (c) Since our method is based on convex hulls, when touching is made in two or more positions, we may not find any segmentation point in the touching cavity region. Hence we get erroneous results.

Table 1: Segmentation results on touching string of different length.

No. of characters in a touching string	Total number of touching strings	Segmentation Accuracy	
		Angle-based	Fourier-Mellin
2	635	92.60%	91.18%
3	206	89.97%	86.25%
≥ 4	38	86.18%	73.68%
Total	879	91.36%	88.38%

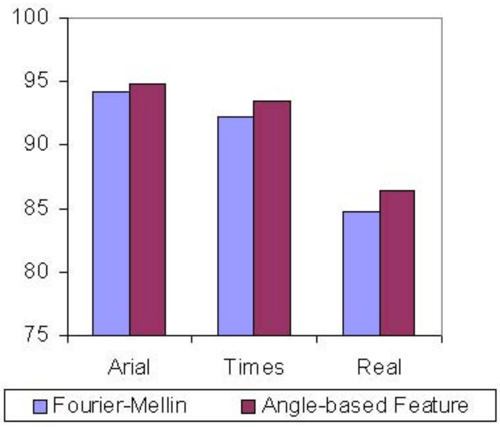


Figure 13: Percentage of touching character segmentation accuracy in datasets of “Arial”, “Times New Roman” fonts and Real dataset.

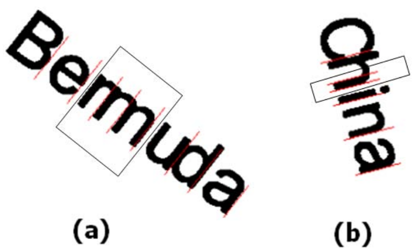


Figure 14: Two examples of wrong segmentation results.

4.4. Experiment on 2-touching Components

Though our objective is to segment n-touching components into their corresponding characters, we have performed an experiment with the additional knowledge of the number of components present in a touching component. This test is performed to check whether the information of the number of character of a touching string improves the performance of touching component segmentation or not. For the experiment, we have created a restricted dataset of touching components containing 2 characters only. The dataset contains 635 components (as mentioned in the 2nd row of Table 1) and the characters are in multi-scale and multi-orientation fashion. The segmentation on this dataset is done based on the concept discussed in the first paragraph of Section 3. We have obtained 95.74% character segmentation accuracy in this experiment. In Fig.15, we have shown some touching images with their segmentation results obtained using this algorithm. From the Table 1, it can be noted that we obtain 92.60% accuracy on two character touching strings when the number of characters in a touching string was not known. Thus, it is to be noted that we achieved 3.14% (95.74% - 92.60%) higher accuracy than the dynamic programming based approach when additional information of the number of characters in a touching component is used.

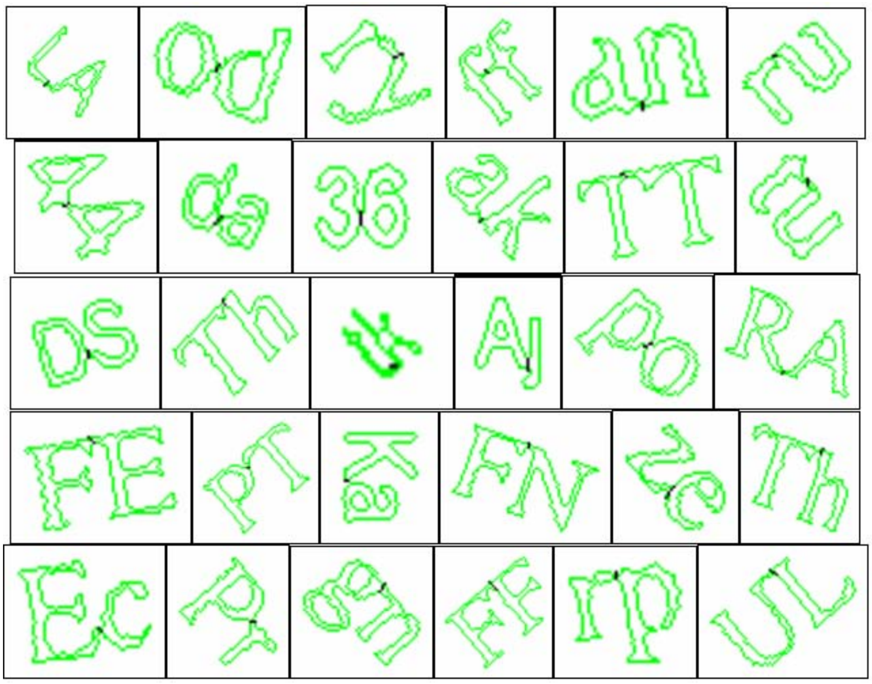


Figure 15: Few images showing segmentation in 2-character touching.

4.5. Experiment on Graphical Documents

We have integrated the touching character segmentation method in OCR of graphical documents. The details of these graphical documents are mentioned in Section 4.1. We have performed the character recognition in 10 maps. We show a map in Fig.16. These maps contain text characters at different scale and orientation. There were long graphical lines that touch or overlap with text in these documents. To separate the text components, we remove the long graphical objects that are present in the document using [26]. Also, the text characters in a string sometimes touch together. So, our method of touching character segmentation is applied here for improving character recognition in these documents. The extracted text characters are recognized initially using only isolated character recognition approach. Next, we integrated the touching character segmentation method and compared the recognition accuracy. We show in Table 2 the improvement of OCR in such documents when segmentation based recognition method is used.

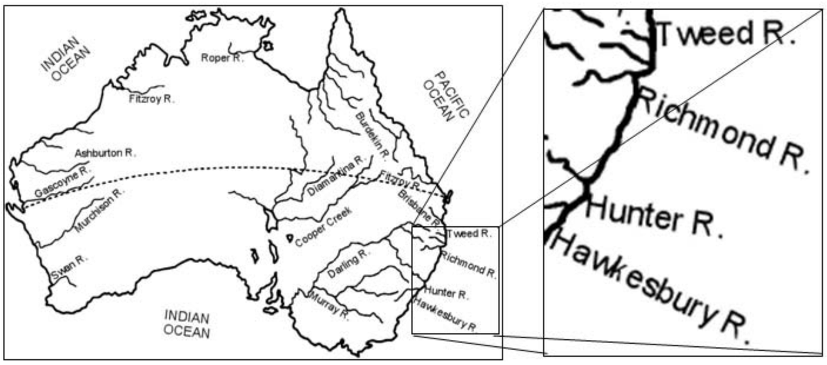


Figure 16: Part of a map showing orientation and touching of characters.

Table 2: Comparison of OCR accuracy by adding touching character segmentation approach

Map Image	Total Number of Characters in the Document	OCR Accuracy without Touching Segmentation	OCR Accuracy with Touching Segmentation
1	251	91.24%	96.81%
2	236	91.95%	95.76%
3	318	92.77%	95.60%
4	458	93.45%	96.07%
5	173	91.33%	96.53%
6	185	87.02%	92.97%
7	201	88.56%	97.01%
8	362	89.50%	95.30%
9	277	97.47%	98.19%
10	180	86.67%	93.89%

5. Conclusions

In this paper we have proposed a scheme towards segmentation of multi-oriented and multi-sized n-character touching strings. The algorithm, at first, segments the touching characters into primitives and then finds the best sequence of characters shapes based on a dynamic programming approach using these primitive segments.

We also performed an adhoc segmentation approach of touching characters based on the knowledge of number of characters in the touching string. In such a restrictive dataset, we have obtained better performance. But, in a real environment, it is not possible to know a priori the number of characters in the touching component. Hence, the proposed approach based on dynamic programming explores the full idea for such segmentation.

To the best of our knowledge, this work is pioneer towards multi-oriented and multi-sized n-character touching string segmentation. We have tested our method on various multi-oriented touching character data with different fonts and scale. As the features of text characters are devised for multi-scale and multi-orientation, some touching characters are not segmented properly due to different possibility of segmentation. Such situation can be taken care of by using a word dictionary in the dynamic programming algorithm and by using contextual information.

6. Acknowledgements

This work has been partially supported by the Spanish projects TIN2009-14633-C03-03, TIN2008-04998 and CONSOLIDERINGENIO 2010 (CSD2007-00018). It has also been supported in part by the AAP program 2010-2011 of François Rabelais University, Tours City, France. The authors would like to thank Dr. Sebastien Adam for providing the Fourier-Mellin shape descriptor code for our research.

Appendix

Douglas-Peucker polyline-approximation algorithm : The classical Douglas-Peucker [6, 11] polyline-approximation algorithm generates a set of points to represent the original line. It works from top to bottom by starting with a rough initial approximation at a simplified polyline, namely the single edge joining the first and the last vertices of the polyline. Then the remaining vertices are tested for closeness to that edge. If there are vertices further than a specified tolerance, ϵ, away from the edge, then the vertex furthest from it is added to the simplification. This creates a new guess for the simplified polyline. Using recursion, this process continues for each edge of the current guess until all vertices of the original polyline are within the tolerance of the simplification. Fig.17 explains a few steps for obtaining approximated polyline. For a polyline shown in Fig.17(a), the contour pixel (V_t) has the maximum distance from the line joining the farthest points (V_1 and V_n) of the polyline. Next, the polyline is simplified with lines V_1V_t and V_tV_n as shown in Fig.17(b). In Fig.17(c), this process is iterated in polyline segment V_tV_n and V_u is selected having the maximum distance between points V_t and V_n.

References

[1] S. Adam, J. M. Ogier, C. Carlon, R. Mullot, J. Labiche, and J. Gardes. Symbol and character recognition: application to engineering drawing. *International Journal on Document Analysis and Recognition*, 3:89–101, 2000.

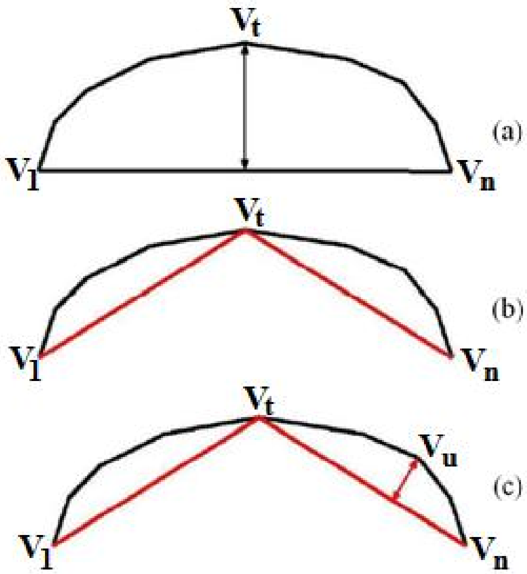


Figure 17: Stages of Douglas-Peucker for Polyline Approximation.

[2] T. C. Chang and S. Y. Chen. Character segmentation using convex-hull techniques. *International Journal of Pattern Recognition and Artificial Intelligence*, 13(6):833–858, 1999.

[3] Y. K. Chen and J. F. Wang. Segmentation of single- or multiple-touching handwritten numeral string using background and foreground analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1304–1317, 2000.

[4] M. Delalandre, E. Valveny, T. Pridmore, and D. Karatzas. Generation of synthetic documents for performance evaluation of symbol recognition and spotting systems. *International Journal on Document Analysis and Recognition*, online first articles, May 2010.

[5] G. Dimauro, S. Impedovo, and G. Pirlo. From character to cursive script recognition: Future trends in scientific research. In *11th International Conference on Pattern Recognition*, volume 2, page 516, 1992.

[6] D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973.

[7] R. Fenrich. Segmentation of automatically located handwritten words. In *Second International Workshop on Frontiers on Handwriting Recognition*, pages 33–44, 1991.

[8] H. Fujisawa, Y. Nakano, and K. Kurino. Segmentation methods for character recognition from segmentation to document structure analysis. In *Proc. IEEE*, volume 80, pages 1079–1092, 1992.

[9] S. Gnesi, U. Montanari, and A. Martelli. Dynamic programming as graph searching: An algebraic approach. *Journal of ACM*, 28:737–751, 1981.

[10] R. Graham. An efficient algorithm for determining the convex hull of a finite point set. *Info. Proc. Letters*, 1:132–133, 1972.

[11] J. Hershberger and J. Snoeyink. Speeding up the douglas-peucker line-simplification algorithm. In *Proceedings of the 5th International Symposium on Spatial Data Handling*, volume 1, pages 134–143, Charleston, South Carolina, 1992.

[12] M.-K. Hu. Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory*, IT-8:179–187, 1962.

[13] S. Kahan, T. Pavlidis, and H. Baird. On the recognition of printed characters of any font and size. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9:274288, 1987.

[14] A. Khotanzad and Y. Hong. Invariant image recognition by zernike moments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:489–497, 1990.

[15] K. Kim, J. Kim, and C. Suen. Recognition of unconstrained handwritten numeral strings by composite segmentation method. In *15th International Conference on Pattern Recognition*, pages 594–597, 2000.

[16] A. Kolesnikov. *Efficient algorithms for vectorization and polygonal approximation.* PhD thesis, University of Joensuu, Finland, 2003.

[17] S. Liang, M. Shridhar, and M. Ahmadi. Segmentation of touching characters in printed document recognition. *Pattern Recognition*, 27(6):825–840, 1994.

[18] Y. Lu. Machine printed character segmentation: An overview. *Pattern Recognition*, 28:67–80, 1995.

[19] Z. Lu, Z. Chi, W. Siu, and P. Shi. A background-thinning based approach for separating and recognizing connected handwritten digit strings. *Pattern Recognition*, 32:921–933, 1999.

[20] L. E. Oliveira, E. Lethelier, F. Bortolozzi, and R. Sabourin. A new segmentation approach for handwritten digits. In *15th International Conference on Pattern Recognition*, pages 2323–2326, 2000.

[21] U. Pal, A. Belaid, and C. Choisy. Touching numeral segmentation using water reservoir concept. *Pattern Recognition Letters*, 24 (1-3):261–272, 2003.

[22] U. Pal, P. P. Roy, N. Tripathy, and J.Lladós. Multi-oriented bangla and devnagari text recognition. *Pattern Recognition*, 43:4124–4136, 2010.

[23] J. Ricard, D. Coeurjolly, and A. Baskurt. Generalizations of angular radial transform for 2d and 3d shape retrieval. *Pattern Recognition Letters*, 26:2174–2186, 2005.

[24] P. P. Roy, U. Pal, and J. Lladós. Recognition of multi-oriented touching characters in graphical documents. In *Indian Conference on Computer Vision, Graphics and Image Processing*, volume 1, pages 297–304, India, 2008.

[25] P. P. Roy, U. Pal, J. Lladós, and M. Delalandre. Multi-oriented and multi-sized touching character segmentation using dynamic programming. In *10th International Conference on Document Analysis and Recognition*, pages 11–15, Barcelona, Spain, 2009.

[26] P. P. Roy, E. Vazquez, J. Lladós, R. Baldrich, and U. Pal. A system to segment text and symbols from color maps. *Revised Selected Papers of Workshop on Graphics Recognition, Lecture Notes in Computer Science*, 1:245–256, 2008.

[27] T. Saba, G. Sulong, and A. Rehman. A survey on methods and strategies on touched characters segmentation. *International Journal of Research and Reviews in Computer Science*, 1:103–114, 2010.

[28] J. Song, Z. Li, M. R. Lyu, and S. Cai. Recognition of merged characters based on forepart prediction, necessity-sufficiency matching, and character-adaptive masking. *IEEE Trans. on SMC B*, 35:2–11, 2005.

[29] N. Strathy and C. Suen. A new system for reading handwritten zip codes. In *3rd International Conference on Document Analysis and Recognition*, volume 1, pages 74–77, 1995.

[30] E. Vellasques, L. S. Oliveira, A. S. Britto, Jr., A. L. Koerich, and R. Sabourin. Filtering segmentation cuts for digit string recognition. *Pattern Recognition*, 41(10):3044–3053, 2008.

[31] G. Yong, Z. Yan, and H. Zhao. Touching string segmentation using MRF. In *Proceedings of International Conference on Computational Intelligence and Security*, pages 520–524, 2009.

[32] D. Yu and H. Yan. Separation of single-touching handwritten numeral strings

based on structural features. *Pattern Recognition*, 31:1835–1847, 1998.