Sādhanā Vol. 31, Part 6, December 2006, pp. 755–769.

Handwriting segmentation of unconstrained Oriya text

N TRIPATHY and U PAL

Computer Vision and Pattern Recognition Unit, Indian Statistical Institute, 203 B T Road, Kolkata 700 108, India
e-mail: umapada@isical.ac.in

Abstract. Segmentation of handwritten text into lines, words and characters is one of the important steps in the handwritten text recognition process. In this paper we propose a water reservoir concept-based scheme for segmentation of unconstrained Oriya handwritten text into individual characters. Here, at first, the text image is segmented into lines, and the lines are then segmented into individual words. For line segmentation, the document is divided into vertical stripes. Analysing the heights of the water reservoirs obtained from different components of the document, the width of a stripe is calculated. Stripe-wise horizontal histograms are then computed and the relationship of the peak–valley points of the histograms is used for line segmentation. Based on vertical projection profiles and structural features of Oriya characters, text lines are segmented into words. For character segmentation, at first, the isolated and connected (touching) characters in a word are detected. Using structural, topological and water reservoir concept-based features, characters of the word that touch are then segmented. From experiments we have observed that the proposed "touching character" segmentation module has 96·7% accuracy for two-character touching strings.

Keywords. Indian language; Oriya script; character segmentation; handwriting recognition.

1. Introduction

Segmentation of handwritten text into lines, words and characters is one of the important steps in the handwritten script recognition process. The task of individual text-line segmentation from unconstrained handwritten documents is complex because the characters of two consecutive text-lines may touch or overlap. These overlapping or "touching" characters complicate the line-segmentation task. Many techniques like global and partial projection analysis, techniques based on statistical modelling etc. (Casey & Lecolinet 1996; Liang *et al* 1999; Zahour *et al* 2001) are used for text-line segmentation from non-Indian scripts, but there is only one work, as we know, on the segmentation of unconstrained hand-written text of Indian script (Pal & Datta 2003). To the best of our knowledge there is no work on segmentation of unconstrained handwritten Oriya text. In this paper, we propose a scheme to segment unconstrained

Oriya handwritten text into lines, words and characters. Oriya is a popular and widely spoken language in the Indian sub-continent.

Although word segmentation from lines is relatively easy, character segmentation from words is difficult because (i) two consecutive characters of a word may touch, or (ii) two side-by-side non-touching characters are seldom vertically separable. Many techniques have been proposed on touching character segmentation (Casey & Lecolinet 1996). One class of approaches uses contour-tracing features of the component for segmentation (Kim *et al* 2000). Analysing the contour of a touching pattern, valley and mountain points are derived. Next, the cutting path is decided to segment the touching pattern by joining valley and mountain points. Valley and mountain points cannot be detected properly if the two numerals touch in a straight-line fashion. For such touching patterns, valley and mountain points cannot be detected correctly in proper touching positions by contour-tracing and hence cannot be segmented properly. This is the main drawback of the contour tracing-based method. Some researchers use profile features for touching string segmentation (Fujisawa *et al* 1992). Computing the upper and lower profile of the component and analysing the distance between upper and lower profiles, segmentation points are detected. Profile-based methods fail when the handwritten texts are strongly skewed or overlapping. This is because in these cases, proper segmentation points cannot be reached by the profiles. Thinning-based methods are also reported for touching characters segmentation (Chen & Wang 2000). In this approach, thinning of foreground and/or background pixels of the connected pattern is done. The end and fork points obtained by thinning are used for cutting-point extraction. This method is time-consuming and additionally generates protrusions. These protrusions sometimes give wrong results because they bring in some confusion among the actual fork and end points. Combined features-based methods are also used for touching-string segmentation. Oliveira *et al* (2000) used contour, profile and skeleton features for touching-character segmentation. Dimauro *et al* (1997) applied contour-based features along with some descending algorithms for the purpose.

In this paper, we propose a scheme to segment unconstrained handwritten Oriya text into lines, words and characters. For line segmentation, we divide the text into vertical stripes and determine horizontal histogram projections of these stripes. The relationship of the peak–valley points of the histograms is used to segment text lines. Based on vertical projection profiles and structural features of Oriya characters, lines are segmented into words. Segmentation of characters from handwritten words is difficult as the characters are mostly connected and/or overlap in words. For character segmentation we first detect isolated and touching characters in a word. Touching characters of the word are then segmented using structural, topological and water reservoir concept-based features (Pal *et al* 2003).

Overall organization of the rest of the paper is as follows. Section 2 deals with some properties of the Oriya script related to this work. The concept of the water reservoir principle is discussed in § 3. Text-line and word segmentation approach is discussed in § 4. Touching character detection and their segmentation approach is discussed in § 5. Finally, results and discussion are provided in § 6.

2. Properties of Oriya script

Oriya, an Indo-Aryan language spoken by about 31 million people mainly in the Indian state of Orissa, and also in West Bengal and Jharkhand. The alphabet of the modern Oriya script consists of 11 vowels and 41 consonants. These characters are called basic characters. The

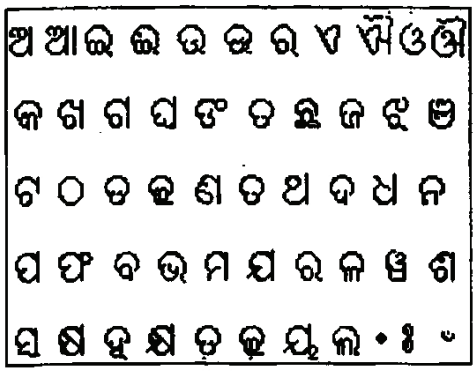

Figure 1. Basic characters of Oriya alphabet. (First 11 are vowels and the rest are consonants.)

basic characters of Oriya script are shown in figure 1. Writing style in the script is from left to right. The concept of upper/lower case is absent in Oriya script.

In Oriya script (as in some other Indian languages), a vowel following a consonant takes a modified shape, which, depending on the vowel, is placed at the left, right (or both) or bottom of the consonant. These are called modified characters. A consonant or vowel following a consonant sometimes takes a compound orthographic shape, which we call *compound character*. Such characters can be combinations of consonant and consonant, as well as consonant and vowel.

A text line may be partitioned into three zones. The *upper-zone* denotes the portion above the mean line, the *middle zone* covers the portion of basic (and compound) characters below the mean line and the *lower-zone* is the portion below the baseline. An imaginary line, where most of the uppermost (lowermost) points of characters of a text line lie, is referred as mean line (baseline). Examples of zoning are shown in figure 2. In this case, the mean line along with the baseline, partitions the text line into three zones.

There are some distinct differences between the English and Oriya scripts. In English there are 26 upper-case and 26 lower-case characters, whereas there is no concept of upper/lower case in Oriya. Another major difference is the shape of the characters. Most of the Oriya characters are round shape in the upper region. Owing to such rounded shapes, when two or more characters are written side by side to form a word, these rounded parts may touch at the upper part of the characters and generate touching patterns in most of the cases. From figure 1 it can be seen that out of 52 characters, 37 characters have a round shape at the upper part. From statistical analysis of 4000 touching components, we note that 72% of the touching components touch at the upper part (near the mean line), 11% touch in the lower zone and 17% touch mainly in the lower half of the middle zone. Based on this statistical analysis and the structural shape of the touching patterns, we have designed the segmentation scheme.

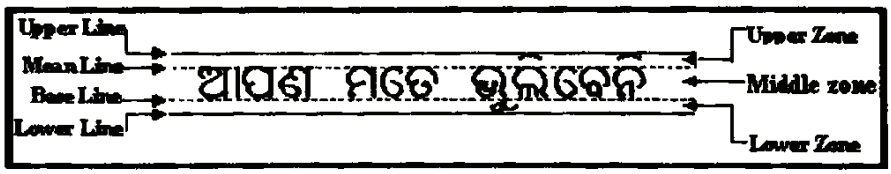

Figure 2. Different zones of an Oriya text line.

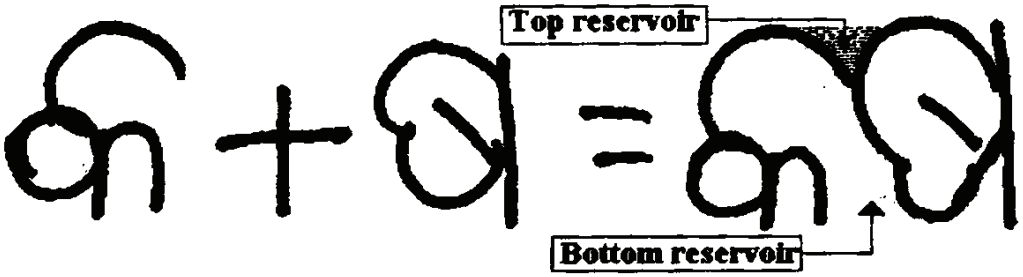

Figure 3. Examples of big reservoirs created by touching (because of the touching of two characters a big bottom reservoir is formed here).

3. Water reservoir principle

The water reservoir principle is as follows. If water is poured from the top (bottom) of a component, the cavity regions of the component where water is stored are considered the top (bottom) reservoirs (Pal *et al* 2003). For an illustration see figure 3. Here, two Oriya characters touch and create a large space which represents the bottom reservoir. This large space is very useful for touching character detection and segmentation. Owing to the shape of Oriya characters a small top reservoir is also generated due to touching (see figure 3). This small top reservoir also helps in touching character detection and segmentation.

All reservoirs are not considered for future processing. Reservoirs having heights greater than a threshold T_1 are selected for future use. For a component the value of T_1 is chosen as 1/9 times the component height. (The threshold is determined from experiment.)

We now discuss here some terms relating to water reservoirs that will be used in feature extraction.

Top reservoir: By top reservoir of a component, we mean the reservoir obtained when water is poured from the top of the component.

Bottom reservoir: By bottom reservoir of a component we mean the reservoir obtained when water is filled from the bottom of the component. A bottom reservoir of a component is visualized as a top reservoir when water is poured from the top after rotating the component by 180°.

Left (right) reservoir: If water is poured from the left (right) side of a component, the cavity regions of the component where water is stored are considered the left (right) reservoirs. A left (right) reservoir of a component is visualized as a top reservoir when water is poured from the top after rotating the by 90° clockwise (anti-clockwise).

Water reservoir area: By area of a reservoir we mean the area of the cavity region where water can be stored if water is poured from a particular side of the component. The number of pixels inside a reservoir is computed and this number is considered the area of the reservoir.

Water flow level: The level from which water overflows from a reservoir is called the water flow level of the reservoir (see figure 4).

Reservoir baseline: A line passing through the deepest point of a reservoir and parallel to the water flow level of the reservoir is called the reservoir baseline (see figure 4).

Height of a reservoir: By height of a reservoir we mean the depth of water in the reservoir. In other words, height of a reservoir is the normal distance between reservoir baseline and water flow level of the reservoir. In figure 4, H denotes the reservoir height.

Width of a reservoir: By width of a reservoir, we mean the normal distance between two extreme boundaries (perpendicular to base-line) of a reservoir.

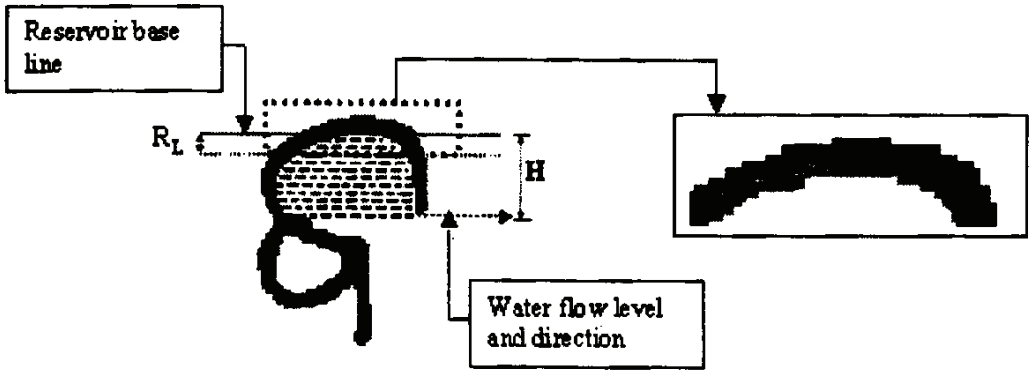

Figure 4. Illustration of different features obtained from water reservoir principle. 'H' denotes the height of bottom reservoir. Gray area of the zoomed portion represents reservoir base area.

In each selected reservoir we compute its base-area points. By base-area points of a reservoir we mean those border points of the reservoir which have height less than $2 \times R_L$ from the base-line of the reservoir. Base-area points for a component are shown in the zoomed-in version of figure 4. Here R_L is the length of the most frequently occurring black runs of a component. In other words, R_L is the statistical mode of the black run lengths of a component. The value of R_L is calculated as follows. The component is scanned both horizontally and vertically. If for a component we get n different run-lengths $r_1, r_2, \ldots r_n$ with frequencies $f_1, f_2 \ldots f_n$ respectively, then the value of $R_L = r_i$, where $f_i = \max(f_j)$, $j = 1 \ldots n$.

4. Line and word segmentation

The global horizontal projection method computes the sum of all black pixels on every row and constructs the corresponding histogram. Based on the peak/valley points of the histogram, individual lines are generally segmented. Although this global horizontal projection method is applicable for line segmentation of printed documents, it cannot be used in unconstrained handwritten documents because the characters of two consecutive text-lines may touch or overlap. For example, see the 4th and 5th text lines of the document shown in figure 5a. Here,

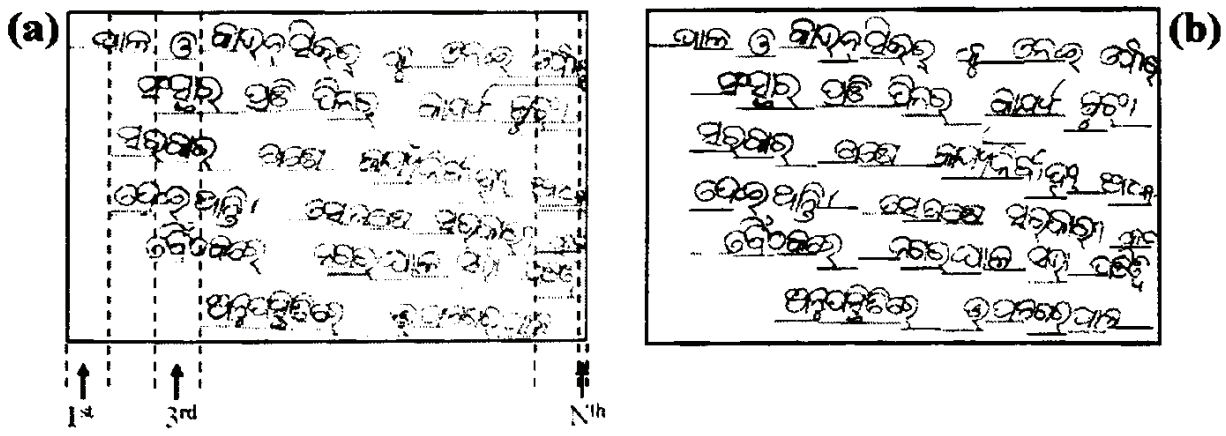

Figure 5. (a) N-stripes and PSL lines in each stripe are shown for a sample of Oriya handwritten text. (b) Potential PSLs of figure 5 (a) are shown.

these two lines are mostly overlapping. To take care of unconstrained handwritten documents, we use here a piece-wise projection method as below.

Here, at first, we divide the text into vertical stripes of width W (here we assume that a document page is in portrait mode). Width of the last stripe may differ from W. If the text width is Z and the number of stripe is N, the width of the last stripe is $[Z - W \times (N - 1)]$. Computation of W is discussed later. Next, we compute piece-wise separating lines (PSL) from each of these stripes. We compute the row-wise sum of all black pixels of a stripe. The row where this sum is zero is a PSL. We may get a few consecutive rows where the sum of all black pixels is zero. Then the first row of such consecutive rows is the PSL. The PSLs of different stripes of a text are shown in figure 5a by horizontal lines. All these PSLs may not be useful for line segmentation. We choose some potential PSLs as follows. We compute the normal distances between two consecutive PSLs in a stripe. So if there are n PSLs in a stripe we get $n - 1$ distances. This is done for all stripes. We compute the statistical mode (M_{PSL}) of such distances. If the distance between any two consecutive PSLs of a stripe is less than M_{PSL}, we remove the upper PSL of these two PSLs. PSLs obtained after this removal are the *potential PSLs*. The potential PSLs obtained from the PSLs of figure 5a are shown in figure 5b. We note the left and right co-ordinates of each potential PSL for future use. By proper joining of these potential PSLs, we get individual text lines. It may be noted that sometimes because of overlapping or touching of one component of the upper line with a component of the lower line, we may not get PSLs in some regions. Also, because of some modified characters of Oriya (e.g. *ikar, chandrabindu*) we find some extra PSLs in a stripe. We take care of them during PSL joining, as explained next.

Joining of PSLs is done in two steps. In the first step, we join PSLs from right to left and, in the second step, we first check whether line-wise PSL joining is complete or not. If for a line it is not complete, joining from left to right is done to obtain complete segmentation. We say PSLs joining of a line is complete if the length of the joined PSLs is equal to the column (width) of the document image. This two-step approach is done to get good results even if two consecutive text lines are overlapping or connected.

To join a PSL of the ith stripe, say K_i, to a PSL of $(i - 1)$th stripe, we check whether any PSL, whose normal distance from K_i is less than $M_{PSL_{i-1}}$, exists or not in the $(i - 1)$ stripe. If it exists, we join the left co-ordinate of K_i with the right co-ordinate of the PSL in the $(i - 1)$th stripe. If it does not exist, we extend the K_i horizontally in the left direction until it reaches the left boundary of the $(i - 1)$th stripe or intersects a black pixel of any component in the $(i - 1)$th stripe. If the extended part intersects the black pixel of a component of the $(i - 1)$th stripe, we decide the "belongingness" of the component in the upper line or lower line. Based on the belongingness of this component, we extend this line in such a way that the component falls in its actual line. Belongingness of a component is decided as follows.

We compute the distances from the intersecting point to the topmost and bottommost point of the component. Let d_1 be the top distance and d_2 the bottom distance. If $d_1 < d_2$ and $d_1 < (M_{PSL}/2)$ then the component belongs to the lower line. If $d_2 \leq d_1$ and $d_2 < (M_{PSL}/2)$ then the component belongs to the upper line. If $d_1 > (M_{PSL}/2)$ and $d_2 > (M_{PSL}/2)$ then we assume the component touches another component of the lower line. If the component belongs to the upper-line (lower-line) then the line is extended following the contour of the lower part (upper part) of the component so that the component can be included in the upper line (lower line). The line extension is done until it reaches the left boundary of the $(i - 1)$th stripe. If the component is touching, we detect possible touching points based on the structural shape of the touching component. From the experiment, we notice that in most of the touching cases there exist junction/crossing shapes or there exist some obstacle points in the middle

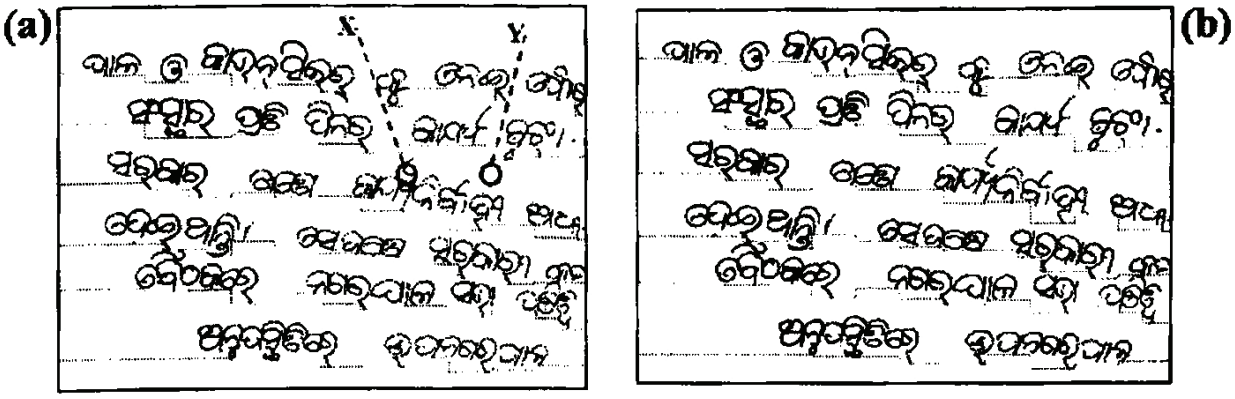

Figure 6. Line-segmented result of text shown in figure 5. Text line segmentation is shown by solid lines. **(a)** Two end points of a mis-segmented line XY are marked by circles. **(b)** Correct segmentation is shown.

portion having low black pixel density of the touching component. These obstacle points and the junction/crossing shape help to find touching position. Extension of PSL is done through this touching point to segment the component into two parts.

Sometimes because of some modified Oriya characters we may get some wrongly segmented lines. For example, see the line marked XY (see figure 6a). To take care of such wrong lines we compute the density of black pixels and compare this value with the candidate length of the line. (By candidate length of a line we mean the distance between the leftmost column of the leftmost component and the rightmost column of the rightmost component of that line.) Let L be the candidate length of a line. Now we scan each column of the portion of the line that belongs to the candidate length to check the presence of black pixels. If a black pixel does not exist in at least 50% of the column of that line then the line is not a valid line and we delete the lower boundary of this line to merge this line with its lower line. Thus a mis-segmented line like XY of figure 6a is corrected. The corrected line segmentation result is shown in figure 6b.

In short, line segmentation algorithm (LINE-SEGM) is as follows.

Algorithm LINE-SEGM:

Step 1. Divide the text into vertical stripes of width W.
Step 2. Compute piece-wise separating lines (PSL) from each of these stripes as discussed earlier.
Step 3. Compute potential PSLs from the PSLs obtained in step 2.
Step 4. Chose the rightmost top potential PSL and extend (from right to left) this PSL up to the previous stripe as discussed in § 4.
Step 5. Continue this PSL joining from right to left until we reach the left boundary of the left-most stripe.
Step 6. Check whether the length of the line drawn equals to the width of the document. If yes, go to step 7. Else, PSL line extension is done to the right until we reach the right boundary of the document.
Step 7. Repeat steps 4 to 6 for the potential PSLs not considered for joining so far. If there is no more PSL for joining, stop.

To get a size-independent measure, computation of W is done as follows. We compute the statistical mode (m_d) of the widths of the bottom reservoirs obtained from the text. This

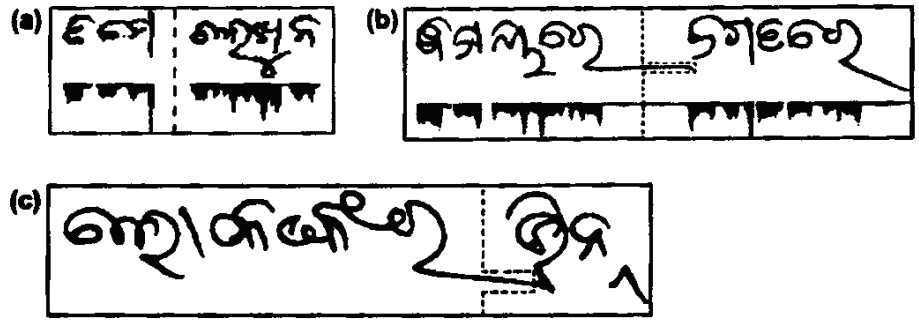

Figure 7. Example of word segmentation. Segmentation is shown by dotted lines. **(a)** Two words with enough space between them. **(b)** Two words with insufficient space between them. **(c)** Two words that touch.

mode is generally equal to character width. Since average character in an Oriya word is four, the value of W is assumed as $4 \times m_d$ to make the stripe width the word width. We computed word-length statistics from 7500 Oriya words.

The proposed line-segmented method does not depend on size and style of the handwriting. Even if the handwritten lines overlap, touch or are curved, the proposed scheme works.

For word segmentation from a line, we compute vertical histograms of the line. In general, the distance between two consecutive words of a line is greater than the distance between two consecutive characters in a word. Taking the vertical histogram of the line and using the above distance criteria we segment words from lines. For example, see figure 7a.

Owing to the characteristics of some Oriya letters in some of the handwriting samples we may not get enough white space between two consecutive words. For example, see figure 7b. Here, we cannot get a large gap between two words because of the long horizontal line-like structure of the rightmost character of the left word. For the segmentation of such words, we use a modified technique as follows. We scan each column of a text line staring from the top row of the line and for each of them we note the position of the topmost black pixel. Similarly, each column is scanned from the bottom and the position of its bottommost pixel is noted. If, for any column, the distance between the top and bottommost points is less than $2 \times R_L$ then we mark that column as zero (white). Else it is marked as one (black). Hence, after the completion of column scanning we get a row of ones and zeros. If the length of a run of zero in this row is greater than W (value of W is computed earlier) then we assume that this run corresponds to a separator of two words and the column (say Y^l) of the image corresponding to the midpoint of this run is noted. If all the pixels of Y^l are white then we consider Y^l as word boundary. If there is any black pixel in Y^l, starting from the topmost black pixel of Y^l the border of the component, from which the black pixel is obtained, is traced clockwise and the path obtained by this tracing is considered the separator of the two words. For example, see figure 7b. Here, the segmentation path is marked by dots.

Sometimes, because of handwriting styles, two consecutive words may touch. For example, see figure 7c. In such cases, from the topmost black pixels of Y^l, we trace clockwise the border of the component to find an obstacle point for segmentation. During tracing, the length of vertical black run at each tracing point is computed. The border point where this run length is greater than $1{\cdot}5 \times R_L$ is considered an obstacle point (R_L is described earlier). We disconnect the touching by replacing the black pixels of the vertical run where the obstacle point lies. The path obtained by tracing along with the disconnected portion is considered the separator of the two words.

5. Character segmentation from word

To segment characters from words, we first detect isolated and connected (touching) characters within each word. Next, connected components are segmented into individual characters.

5.1 *Isolated or connected (touching) component detection*

In principle, when two or more characters in Oriya get connected, one of the four following situations happens in most of the cases: (a) Two consecutive characters create a large bottom reservoir (see figure 3); (b) the number of reservoirs and loops in a connected component is greater than that in an isolated component; (c) two consecutive characters create a small top reservoir near the mean line; (d) the shape of the touching character is more complex than those of isolated characters. By computing different features obtained by the above observations, isolated and touching characters are segmented as follows.

For a component C, let

N = number of close loops;
h = height of C;
w = width of C;
N_T = number of top reservoirs of C;
N_B = number of bottom reservoirs of C;
N_L = number of left reservoirs of C;
N_R = number of right reservoirs of C;
W_i = width of the ith bottom reservoir, $i = 1, 2 \cdots N_B$;
H_C = row-wise maximum number of horizontal black runs of C;
W_M = maximum$(W_i, i = 1 \cdots N_B)$;
$T = N + N_T + N_B + N_L + N_R$.

We define the following Boolean functions based on different characteristics of Oriya connected characters.

$S_1 = f(N_B, h) = 1$, if $N_B \geq 3$, if any of these three bottom reservoirs are adjacent and vertically non-overlapping, and if the height of at least two of these reservoirs is $\geq 50\%$ of h.
$= 0$ otherwise,

$S_2 = f(W_M, w, N_B) = 1$, if $W_M \geq 50\%$ of w and $N_B \geq 3$ and if any of these three bottom reservoirs are adjacent and vertically non-overlapping,
$= 0$ otherwise.

Please note that when two or more characters touch, number of loops and reservoirs of a touching component is bigger than that of an isolated character. Combining measure (T) of loops and reservoirs of a touching character largely differs from that of isolated characters. Owing to this T has been used as one of the criteria for the identification of touching and isolated characters in the proposed scheme. Based on the values obtained from different Boolean functions and the combining measure T, isolated and touching character detection algorithm (ISO-CONN) is developed as follows.

Algorithm ISO-CONN

If $T \geq 7$ then C is touching.
Else if $H_C > 5$ then C is touching.
Else if $S_1 = 1$ then C is touching.
Else if $S_2 = 1$ then C is touching.
Else C is isolated.

The advantage of the proposed isolated and touching component classification method is that it is size-independent and there is no need for any normalization of the component.

5.2 *Segmentation of touching characters into isolated characters*

If a component is detected as touching then we segment the touching component to get its individual characters. For the segmentation of a touching pattern, at first, the touching position is found. Next, based on the touching position, reservoir base-area points, topological and structural features, the touching component is segmented.

From statistical analysis, we note that Oriya characters may touch in three positions: (a) Top, (b) middle, and (c) lower portion. All touching that occurs between mean-line and upper half of middle zone is considered as top touching. From statistical analysis, we note that 72% of the touching is top touching. If the touching occurs in the lower half of the middle zone and above the base-line we mark that as middle touching. About 17% of Oriya touching components is middle touching. Touching below the base-line is lower touching and 11% of the touching components occurs in the lower zone.

5.2a *Top-touching segmentation:* For the top-touching patterns we noted that in most of the cases, a small top reservoir is created near the mean line in the touching patterns (see figure 3). The bottom-most point of the top reservoir is considered one of the extreme points of the cutting path for segmentation. Let this extreme point be U_1. In order to get the other extreme points (U_2) of the cutting path, we consider all the bottom reservoirs that are adjacent to the small top reservoir. If there is any bottom reservoir that lies just below the top reservoir, we consider the base point of this bottom reservoir as the other extreme point U_2. Joining these two extreme points U_1 and U_2, we segment the touching. For example, see figure 8a. Segmentation results of the touching component shown on the left hand side of figure 8a is shown on the right hand side of figure 8a.

If no bottom reservoir is found below the small top reservoir, we choose the adjacent bottom reservoir situated to the left of the top reservoir. From the base point of the chosen bottom reservoir, we trace clockwise the border pixels of the reservoir. While tracing a border pixel if

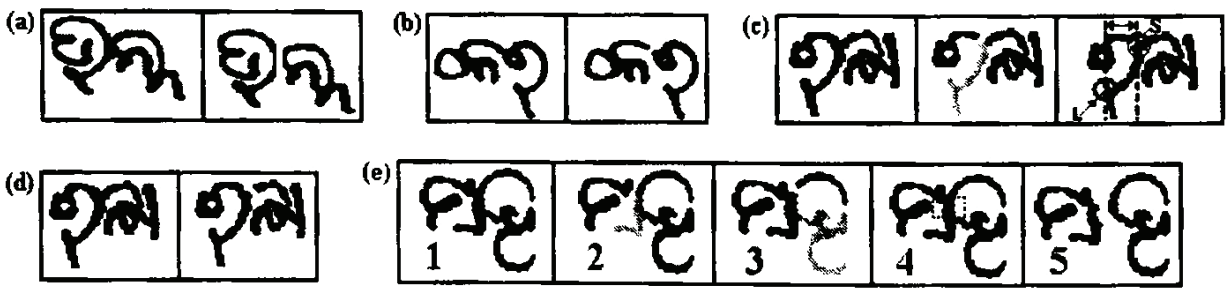

Figure 8. Illustrations of touching character segmentation. Here components of the leftmost box of each rectangular region are the original string to be segmented.

we reach the bottom-most point of the top reservoir from the tracing pixel without crossing any white pixel then that border pixel is marked as the extreme point U_2 for segmentation. If there exist many such border pixels, then the border pixel from which the distance of the base point of the top reservoir is minimum is considered as the extreme point U_2. See figure 8b where an example of such segmentation is shown. Sometimes, this method may produce wrong segmentation. For example, see figure 8c. Here, some part of the left component is wrongly associated with the right component. The wrongly associated part is marked by gray shade in the middle component of figure 8c. To identify this wrong segmentation we check the following validity condition. The position of the leftmost point (L) of the right component with respect to the segmentation point is noted. Let the segmentation point be S. S and the leftmost point L are shown in the right component of figure 8c. If the point L situated in the left side of S and if the column difference of L and S is more than $2 \times R_L$, then we assume there is a mistake in the segmentation and we ignore this segmentation. For illustration, see the rightmost image of figure 8c. Column difference between L and the segmented point S is shown by an arrow in this figure. Since this column difference is more than $2 \times R_L$ we ignore this segmentation and we choose the bottom reservoir which is adjacent and situated to the right of the top reservoir. From the base point of the chosen bottom reservoir, we trace the border pixels of the reservoir anti-clockwise. While tracing this, if from any border pixel of the chosen bottom reservoir we can reach the bottom-most point of the top reservoir without hitting any white pixel, then that border pixel is marked as another extreme point U_2 for segmentation. We also check the validity condition of this segmentation point, as discussed above. If the validity condition satisfies, we segment it accordingly. See figure 8d where an example of such segmentation is shown. If a component is not segmented by the above procedure, we assume that there is an overlapping portion between the two components of the touching pattern. The overlapping region is marked by a dotted rectangle in the fourth figure of figure 8e. The second and third figures of 8e show the wrong segmentation results obtained by top and left reservoirs, and top and right reservoirs respectively. Mis-segmented portions are marked by shading in these two figures.

Height of the overlapping area is detected as follows. In most of the cases, overlapping and touching strings have a top reservoir and a bottom reservoir. The height of the overlapping area is the distance between the base points of the top and bottom reservoirs. For segmentation of an overlapping touching string we associate common portions to each of the segmented parts and proceed as follows. To separate left (right) component from the touching string, starting from the leftmost (rightmost) black pixel of the top row of the overlapping zone, we trace the contour of the tracing pattern clockwise (anti-clockwise) and tracing is done up to the bottom row of the overlapping zone. From each traced contour point, a connected black run of length upto R_L is marked and assigned this run to the left (right) component to get it segmented. An example of overlapped touching segmentation results of the first component of figure 8e is shown in fifth component of figure 8e.

In some touching components we may not get a small top reservoir near the mean-line position. This situation is shown in the figure 9a. In order to segment this component, we find the largest bottom reservoir. If such a reservoir exists, we consider that reservoir for segmentation. From the base point of this reservoir we trace the reservoir border points clockwise to find an obstacle point. If there is any obstacle point, we segment that component at that obstacle point. The detection of obstacle point is done as follows. While tracing a point, we calculate a horizontal run-length of black pixels connected to the tracing point. If for a tracing point, the length of the black run exceeds $2 \times R_L$ then we assume that an obstacle point occurs at the tracing point. The touching component is segmented at that point.

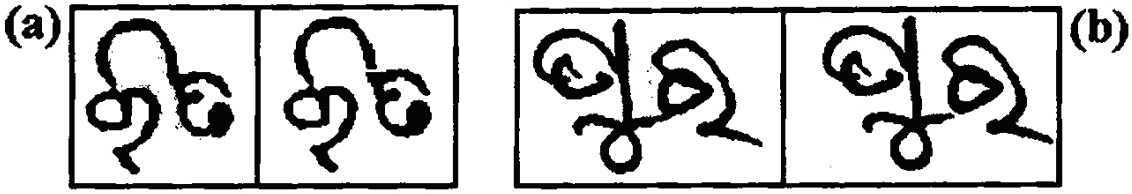

Figure 9. Examples of touching component segmentation using (a) largest bottom reservoir, (b) largest top reservoir. Here reservoirs are marked by gray shades.

A segmentation result of the component shown to the left of figure 9a, is shown on the right hand side of the figure.

5.2b *Middle-touching segmentation:* For segmentation of middle-touching patterns we follow similar techniques of top-touching patterns.

5.2c *Bottom-touching segmentation:* For bottom-touching patterns we find the largest top reservoir whose base-line lies in the lower zone. Starting from the base point of this reservoir we trace its boundary anti-clockwise to find an obstacle point and segmentation is done at that obstacle point. Obstacle point detection procedure is similar to the method discussed above. Instead of a horizontal run length, we compute here the vertical run length of the tracing point. Example of a bottom-touching component and its segmented result is shown in figure 9b.

We reject other touching patterns that appear during the experiment but are not discussed above.

After segmentation, two segmented parts are passed to the isolated and connected component detection module to check whether any of these segmented parts is connected. If any part is detected as connected, it is then sent to the segmentation module for further segmentation. This procedure is repeated until both the segmented parts of a component are detected as isolated by the isolated and connected component detection module. This iterative procedure is done to segment connected characters created by multiple touching, or by touching of 3 or more characters.

6. Results and discussion

Results of different modules of our system are as follows.

6.1 *Results on line segmentation*

For experiments of line segmentation algorithm, landscape documents were considered from individuals following different professions, like students and teachers, bank and post-office employees, businessmen etc. The line segmentation module was tested on 1627 text lines. We ensured that the data sets contained varieties of writing styles. For the experiment we considered only single-column document pages.

To check whether a text line is segmented correctly or not, we draw boundary markers between two consecutive text lines (as shown in figure 6b). By viewing the results on the computer's display we calculate line segmentation accuracy manually. Accuracy of line extraction module is measured according to the following rule. If out of N components of a line, M components are extracted in favour of that line by our scheme, then the accuracy for that line

Table 1. Distributions of line segmentation results.

Number of lines on which experiment is made	Percentage of components that fall in their correct lines
984	100
290	97–99·9
353	< 97

is $(M \times 100)/N\%$. So if all components of a text line are extracted correctly by the proposed algorithm we say accuracy for the line is 100%. Distributions of experimental results of line segmentation module are given in table 1. From the table it is seen that out of 1627 lines, 984 lines are segmented correctly. In other words, all components of these 984 lines are grouped in their individual lines correctly. In 1274(984 + 290) lines, at least 97% characters are grouped correctly in their respective lines.

6.2 *Results of word segmentation*

Our word segmentation module is tested on 3700 words and we notice that the word segmentation module has 98·2% accuracy. From experiments we notice that most of the errors come, when (a) two consecutive words touch, (b) distance between two consecutive words is very small.

6.3 *Results on isolated and connected character identification*

Performance evaluation of the isolated and connected character identification scheme is done manually on 3200 components of Oriya handwritten text. The proposed method has an average accuracy of 96·3%. From the experiment we notice that isolated characters fall into the isolated group in most of the cases (98·6%). Most of the errors arise from connected characters identified as isolated characters.

6.4 *Results on touching component segmentation*

Evaluation of the segmentation scheme was done with 1840 landscape images of Oriya script with touching strings. Out of these 1840 touching components, 1458 components were two-character touching and 311 were three-character touching components, and the rest were generated by four or more components. Some of the two-character touching components were multi-touching components. The segmentation results are verified manually and the distributions of the result are provided in table 2. Some segmentation results of the proposed

Table 2. Touching character segmentation results.

Touching type (number of data)	Segmentation accuracy (%)
Two-character touching (1458)	96·7
Three-character touching (311)	95·1
Four or more character touching (71)	93·3

Figure 10. Example of touching character segmentation. Touching characters are shown on the left of the rectangle and the segmented result is shown on the right. Characters are segmented, replacing black pixels by white pixels in the segmented part.

touching component segmentation module are shown in figure 10 and some mis-segmented results are shown in figure 11. From the experiment, it is seen that maximum accuracy (96·7%) is obtained from two-character touching components. If the number of characters in a touching string increases, the shape complexity of the touching pattern increases and segmentation accuracy reduces in the string. From the experiment, we also notice that in most of the cases mis-segmentation occurs due to an undesired position of the reservoir. Also, some error occurs when the touching area of a touching string is large. Some errors are also generated from multi-touching components. Rejection rate of the proposed method is 4·7% and most of the rejection cases are from multi-touching patterns.

One of the significant advantages of the proposed method is its flexibility. The proposed scheme is size-independent and there is no need for any normalization of the component. Also, the proposed method can handle touching strings of any number of characters.

To the best of our knowledge, there is no work on Oriya unconstrained handwritten segmentation. Thus, we cannot compare our results with any other work.

7. Conclusions

A scheme for segmentation of unconstrained Oriya handwritten text into lines, words and characters is proposed in this paper. Here, at first, the text image is segmented into lines, and then lines are segmented into individual words. Next, for character segmentation from words, initially, isolated and connected (touching) characters in a word are detected. Using structural, topological and water reservoir concept-based features, touching characters of the word are then segmented into isolated characters. To the best of our knowledge, this is the first work of its kind on Oriya text. The proposed water reservoir-based approach can also be used for other Indian scripts where touching patterns show similar behaviour.

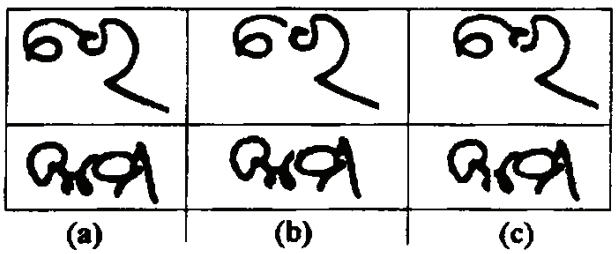

Figure 11. Examples of some mis-segmented results. (**a**) Original images. (**b**) Actual segmentation points are shown on the components. (**c**) Segmentation results obtained by the proposed method.

References

Casey R G, Lecolinet E 1996 A survey of methods and strategies in character segmentation. *IEEE Trans. Pattern Anal Mach. Intell.* 18: 690–706

Chen Yi-Kai, Wang Jhing-Fa 2000 Segmentation of single- or multiple-touching handwritten numeral string using background and foreground analysis. *IEEE Trans. Pattern Anal. Mach. Intell.* 22: 1304–1317

Dimauro G, Impedevo S, Pirlo G, Salzo A 1997 *Automatic bankcheck processing: A new engineered system. Automatic bank check processing* (eds) S Impedovo, P S P Wang, H Bunke (Singapore: World Scientific) pp 5–42

Fujisawa H, Nakano Y, Kurino K 1992 Segmentation methods for character recognition from segmentation to document structure analysis. *Proc. IEEE* 80: 1079–1092

Kim K K, Kim J H, Suen C Y 2000 Recognition of unconstrained handwritten numeral strings by composite segmentation method. In *Proc.15th Int. Conf. on Pattern Recognition* (Los Alamitos, CA: IEEE Comput. Soc.) pp 594–597

Liang J, Philips I, Haralick R M 1999 A statistically based highly accurate text-line segmentation method. *Proc. 5th Int. Conf. on Document Analysis and Recognition* (Los Alamitos, CA: IEEE Comput. Soc.) pp 551–554

Oliveira L S, Lethelier E, Bortolozzi F, Sabourin R 2000 A new approach to segment handwritten digits. *Proc. of 7th Int. Workshop on Frontiers in Handwriting Recognition* (Los Alamitos, CA: IEEE Comput. Soc.) pp 577–582

Pal U, Belaïd A, Choisy Ch 2003 Touching numeral segmentation using water reservoir concept. *Pattern Recog. Lett.* 24: 261–272

Pal U, Datta S 2003 Segmentation of Bangla unconstrained handwritten text. *Proc.7th Int. Conf. on Document Analysis and Recognition* (Los Alamitos, CA: IEEE Comput. Soc.) pp 1128–1132

Zahour A, Taconet B, Mercy P, Ramdane S 2001 Arabic hand-written text-line extraction. *Proc. 6th Int. Conf. on Document Analysis and Recognition* (Los Alamitos, CA: IEEE Comput. Soc.) pp 281–285