IJCSC 0973-7391 Volume 8 • Issue 1 Sept 2016 - March 2017 pp. 63-68 available online at www.csjournalss.com

Segmentation of Overlapping Lines in Handwritten Devanagari Script

[1]Ramandeep Kaur, [2]Simpel Rani
[1`]Assistant Professor, Guru Teg Bhadur Khalsa Institute of Engineering &Tech., Punjab, India
[2]Associate Professor, Deptt. of Computer Science & Engineering, Yadavindra College of Engineering, Punjabi University Guru Kashi Campus, Talwandi Sabo, Bathinda, Punjab

Abstract: Overlapping of consecutive text lines can be seen in both printed as well as in handwritten documents. Overlapping lines in printed text can be seen mainly in newspapers, where some compression algorithms are used to fit large text in small space. In handwritten documents, overlapping of lines occurs due to irregularities of handwriting, which adds complexity to process of line Segmentation. The technique proposed in this paper provides the solution for segmenting horizontally overlapping lines. This technique is not language dependent, and also it can be applied to multilingual text documents. Additionally, it has no constraint on number of consecutive overlapping lines and on overlapping percentage. In this paper, the categorization of overlapping lines obtained by visual inspection of various handwriting samples of Devanagari script is also given.
Keywords: Overlapping Lines, Segmentation, Devanagari Script, Handwritten Document.

1. Introduction

Line segmentation is a critical stage of OCR (Optical Character Recognition), because once the lines get separated these can be subjected to development of lateral stages of OCR and if segmentation results are poor, recognition accuracy decreases. The line segmentation of handwritten documents is a challenging task because of variations in handwriting.

2. Overlapping Lines

When pixels of a line get grouped to the pixels of its adjacent line then text lines are said to be overlapping lines. There is considerable amount of work done on handwritten text segmentation. But there is not enough work on segmentation of overlapping lines. Bansal [1] presented a two-pass algorithm for segmentation of horizontally overlapping lines. Little amount of work on segmentation of horizontally overlapping lines of printed text is available in [2-4]. Takru et. al [5] presents a novel approach which uses structural features of different handwriting styles where overlapping occurs very frequently. Pal et. al [6] proposed a Water Reservoir Principle based technique for segmentation of unconstrained Bangla script. The work on segmentation of handwritten Hindi text is available in [7-8]. A study on various challenges during segmentation of overlapping lines is given in Kaur and Jindal [9].
After analyzing work on overlapping lines, handwriting samples have been collected showing different overlapping patterns. Figure 1 shows overlapping of modifiers in consecutive lines.

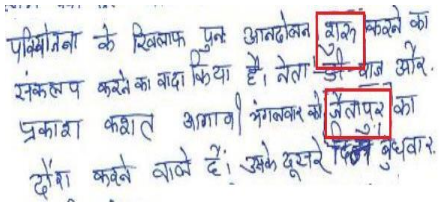


Figure 1: Overlapping lines due to modifiers

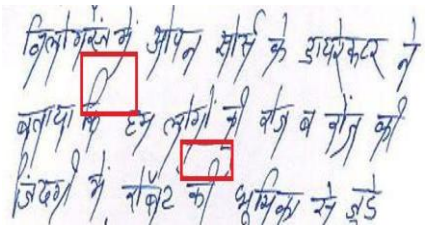


Figure 2: Text document with overlapping lines due to long vertical bars

The Figure 2 and Figure 3, shows overlapping due to handwriting habits. Sometimes overlapping occurs due shadow of text written on backside of page as shown in Figure 4.

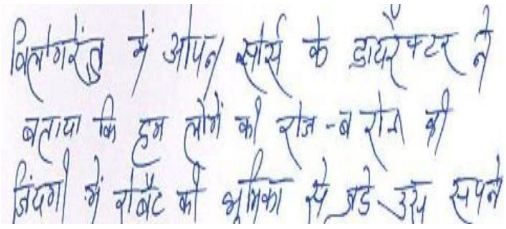


Figure 3: Overlapping due to Characters having curve portion

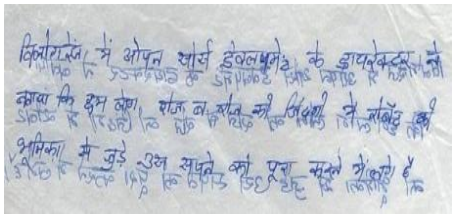


Figure 4: Overlapping of text with shadow of text on opposite side.

3.1 Find optimal line height

To accomplish it, initially take the document and call Algorithm DetectTextArea, to find the starting, ending positions of various textual areas. To calculate the height of each textual area, subtract two consecutive ending positions. Then divide the document into two vertical strips and use Algorithm DetectTextArea in each strip and calculate the height of each textual area, Median, S.D (standard deviation) of all heights. Keep on incrementing the number of vertical strips by one and repeat the whole process. After a particular point S.D again starts increasing with increasing number of strips, as shown in Figure 5. From above procedure optimal number of strips can be calculated, which was 7 in our case.

Figure 5: Graph between S.D and number of strips.

The algorithm DetectTextArea and DetectLineHeight are used to each strip as shown in Figure 6 to find optimal line height

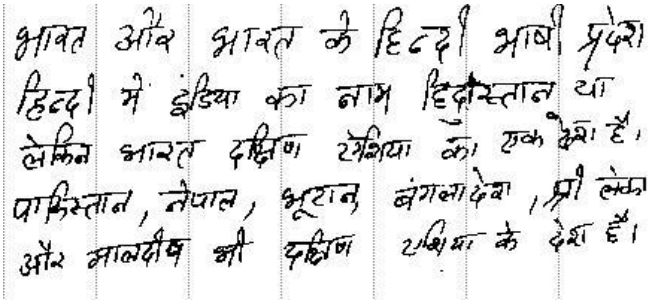


Figure 6: Document divided into strips.

Algorithm - DetectTextArea

[IN] binary Image as pixelArray[][], [OUT] startPositions[], [OUT] endPositions[]
booleanlineContinuations = false

1. For each pixel row in pixelArray
   1.1. Check if black pixel exists, and mark as blackPixelExists
   1.2. If blackPixelExists is true and lineContinuation as false
      1.2.1. Mark the position of row as start of Text Area (in startPositions[])
      1.2.2. Set lineContinuation as true
   1.3. else blackPixelExists is false and lineContinuation as true
      1.3.1. Mark the position of row as end of Text Area (in endPositions[])

Algorithm – detectLineHeight

[IN] binary Image as pixelArray[][],OUT] noOfStrips, [OUT]optimalLineHeight, [OUT]heightVariation, [OUT]whiteSpaceHeight,int stripCount = 1, int, distances[],intwhiteSpaceHeight[], intlastSD =99999999, sd = 99999999, intmaxWSHeight = 0, int median = 0

1. Divide pixelArray[][] into stripCount parts as subPixelArray[][].
2. Initialize distances[] and whiteSpaceHeight[] as empty.
3. For each subPixelArray
   3.1. call**detectTextArea**for [IN]subPixelArray [OUT] subStartPositions and [OUT] subEndPositions.
   3.2. For each i:0 to i <subStartPositions.Length
      3.2.1. Calculate subEndPositions[i+1] – subEndPositions[i], an accumulate into distances[]
      3.2.2. Calculate subStartPositions[i+1] – subEndPositions[i], an accumulate into whiteSpaceHeight[]
4. Calculate standard deviation as sd and Median as median from distances[]
5. Find maximum from whiteSpaceHeight[] as maxWSHeight
6. If sd<lastSD, increase StripCount, and repeat step 1 to 6.
7. Set
   7.1. noOfStrips = stripCount,
   7.2. optimallineHeight = median,
   7.3. heightVariation = lastSD
   7.4. whiteSpaceHeight = maxWSHeight

3.2 Perform Segmentation

For this purpose, we have developed a algorithm segmentLines. It compares height of textual areas with maximum line height (optimalLineHeight + heightVarition). The areas having height greater than maximum line height are considered as regions of overlapping lines. Now our aim is to handle these regions. In order to achieve this start-scanning image from top and find a minimum pixel row near to maximum line height. The regions where there are black pixels in this row are the locations of overlapping areas or we can say if draw a line at the position of minimum pixel row (minPixelRow1) the locations where this line strikes with any black pixel (obstacle) is the location of overlapping text. To handle this, we considered a small rectangular area around that black pixel (obstacle) as shown in Figure 7, find new minimum pixel row (minPixelRow2) in this area as shown in Figure 8 and join it with minpixelRow1 as shown in Figure 9.

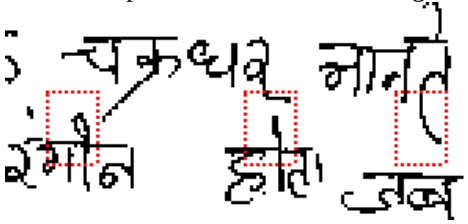


Figure 7: Rectangular area around overlapping regions.

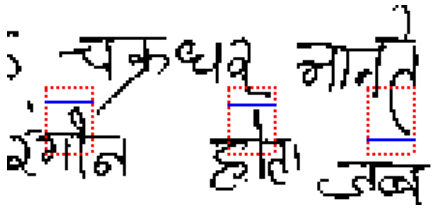


Figure 8: Individual minimum pixel rows (minPixelRow2).

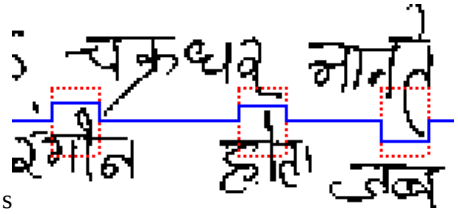


Figure 9: Individual minimum pixel rows (minPixelRow2) joined to minPixelRow1.

Algorithm: segmentLines

[IN] binary Image as pixelArray[][]
Int noOfStrips, int optimalLineHeight, int heightVariation, int whiteSpaceHeight, int distances[],int lastPosition, int contourStripWidth, int minPixelRow1, int minPixelRow2

1. call optimallineHeight for [IN] pixelArray [OUT] noOfStrips, [OUT] optimalLineHeight, [OUT] heightVariation, [OUT] whiteSpaceHeight
2. call detectTextArea for [IN] subPixelArray [OUT] startPositions and [OUT] endPositions
3. Set contourStripWidth = documentWidth / (5 * noOfStrips);
4. Set lastPosition = startPositions[0];
5. for each endPosition in endPositions[]
   - 5.1. while ((endPosition – lastPosition) > (optimalLineHieght + heightVariation))
     - 5.1.1. search for minimum Pixel Row near to lastPosition + optimalLineHeight, and mark as minPixelRow1, and update lastPosition = minPixelRow1
     - 5.1.2. Start drawing horizontal line pixel by pixel at minPixelRow1, till blackPixel is encountered
     - 5.1.3. if blackPixel is encountered, store position as x, y
       - 5.1.3.1. Search for minimum pixel row in small rectangle starting at co-ordinates (x – (contourStripWidth/2)), (minPixelRow1 – (whitespaceHieght/2)) With width= contourStripWidth and height=whitespaceheight and mark as mixPixelRow
       - 5.1.3.2. Join minPixelRow1 and minPixelRow2 (length of minPixelRow1 component will be stripWidth)
   - 5.2. Draw horizontal line at endPosition and update lastPosition = endPosition

4. Results and Discussions

The proposed method is tested on handwriting of different users in Devanagari. Figure 10 shows a test document from our database. Figure 11(a) shows line segmentation of this document and a segmented view for this document is shown in Figure 11(b), where separated lines are shown in different colors.

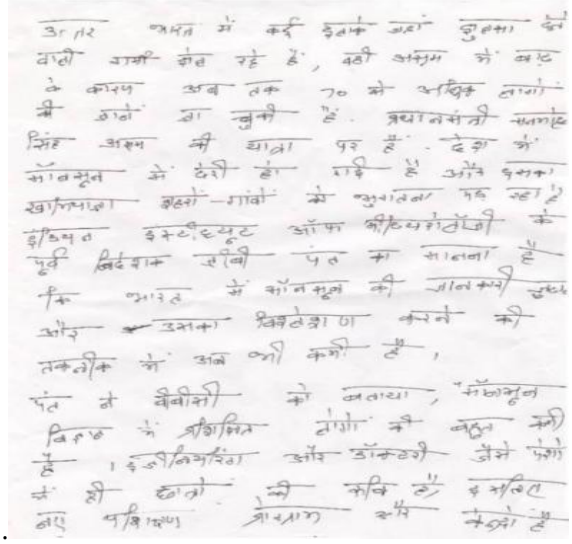
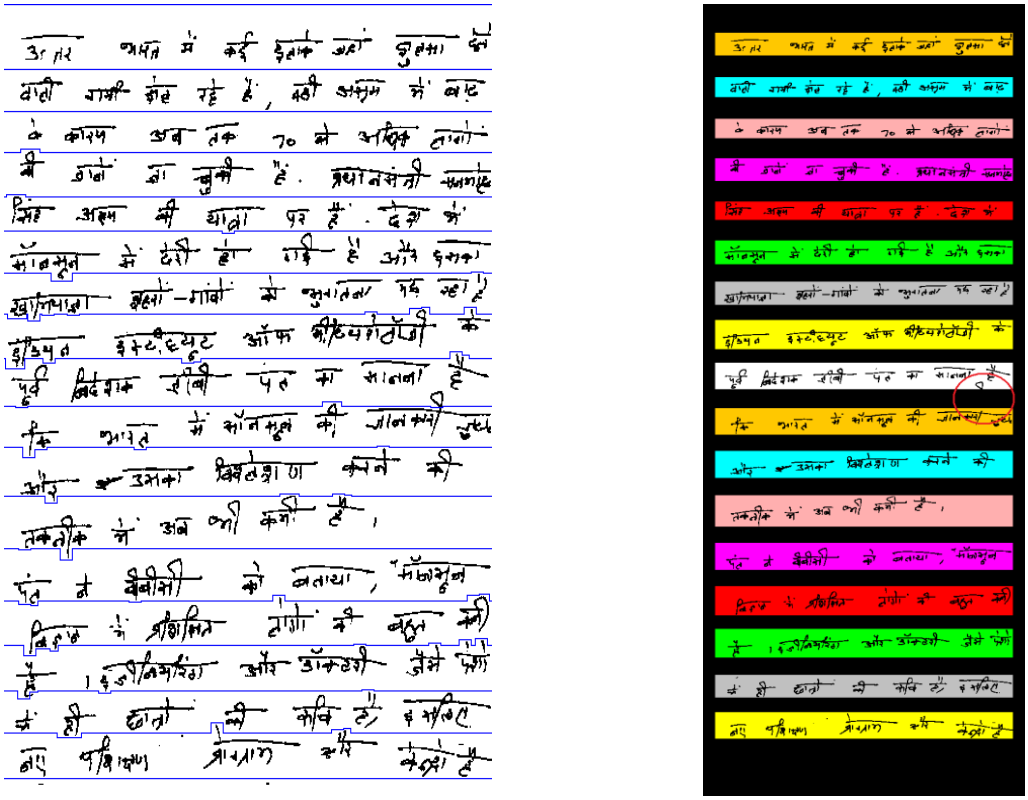


Figure 10: Part of database-Doc 1.

Results on various handwritten documents of Devanagari script are shown in Table 1. In this table test documents are various handwriting samples tested, actual line count denotes total number of text lines present in the document, correctly segmented lines field shows count of lines which are accurately segmented and accuracy denotes the extent to which proposed algorithm is successful. This algorithm is also working well on handwritten Gurumukhi script



(a) (b)

Figure 11: (a) Line segmentation-Doc 1, (b) Segmented View –Doc1

TABLE I: Sample Results

Test Document	Actual line Count	Correctly Segmented lines	Accuracy*
Doc-1	17	16	94%
Doc-2	14	13	93% (appx.)
Doc-3	10	9	90%
Doc-4	17	16	94%

*Accuracy = (Actual line Count /Correctly Segmented line Count) x 100

6. References

[1] V. Bansal, "Integrating knowledge Resources in Devanagari text recognition", Ph. d. Thesis, 1999.

[2] M. K. Jindal, R. K. Sharma and G. S. Lehal, "Segmentation of horizontally overlapping lines in Printed Indian scripts", International Journal of computational intelligence and Research Vol. 3, No. 4, 2007, pp. 227-886.

[3] S. Priyanka, S. Pal and R. Mandal, "Line and Word Segmentation Approach for Printed Documents", IJCA Special Issue on Recent Trends in Image Processing and Pattern Recognition (RTIPPR'10), 2010, pp. 30-36.

[4] M. S. Das, C. Reddy, A. Govardhan and G. Saikrishna, "Segmentation of overlapping Text Lines, Characters in Printed Telugu Text Document images", International Journal of Engineering Science and Technology, Vol. 2(11), 2010, pp. 6606-6610.

[5] K. Takru and G. Leedham, "Separation of touching and overlapping Words in adjacent lines of handwritten text", 8th International workshop on frontiers in Handwriting Recognition, 2002, pp. 496-501.

[6] U. Pal and S. Datta, "Segmentation of Bangla Unconstrained Text", 7th International Conference on Document Analysis and Recognition, 2003, pp. 1128-1132

[7] N. K. Garg, L. Kaur and M. K. Jindal, "Segmentation of Handwritten Hindi Text", International Journal of Computer Applications, Vol. 1(4), 2010, pp. 19-23.

[8] N. K. Garg, L. Kaur and M. K. Jindal, "A new method for Line Segmentation of Handwritten Hindi Text", International Conference on information technology, 2010, pp. 392-397

[9] R. Kaur and S. Jindal, " Problems During Line Segmentation of Overlapping Lines in Handwritten Devanagari Script Documents", IJICT, Vol.(1-2), 2013, pp: 349-353