International Journal of Computer Science and Applications,
©Technomathematics Research Foundation
Vol. 14, No. 2, pp. 64 – 78, 2017

# SEGMENTATION OF OVERLAPPING, SKEWED, AND TOUCHING LINES IN HANDWRITTEN MODI SCRIPT DOCUMENTS

PARAG A. TAMHANKAR

*Department of Computer Science, MES Abasaheb Garware College, Savitribai Phule Pune University, Ganeshkhind Road,*
*Pune,Maharashtra 411004, India*
*paragishere@gmail.com*

KRISHNAT D. MASALKAR

*Department of Mathematics, MES Abasaheb Garware College, Savitribai Phule Pune University, Ganeshkhind Road,*
*Pune,Maharashtra 411004, India*
*krishna_masalkar@rediffmail.com*

SATISH R. KOLHE

School of Computer Sciences, North Maharashtra University, Umavi Nagar
Jalgaon, Maharashtra 425001, India
srkolhe2000@gmail.com

Segmentation of text documents into lines, words, and characters has been an active research problem from many years. The accuracy of segmentation largely controls the success of subsequent character recognition phase. This task becomes challenging in case of cursive handwritten text documents rather than printed ones. MODI Script, one such script which is written in cursive manner and without lifting hand. This paper addresses line segmentation of Handwritten MODI Script documents. The algorithm described in this paper is broadly divided into two modules. The first module extends the result obtained from Horizontal Projection Profile Method and selects valley using a criterion based on minimizing the line segmentation error. The second module then eliminates the error from first module. The methods employed in this work are simple and computationally inexpensive. The results obtained by using this algorithm performs line segmentation accurately even for the document containing overlapping and/or touching lines.

*Keywords*: MODI Script; Segmentation; Valley Selection Criterion; Connected Component.

MSC Primary 68U15; Secondary 68-04;

## 1. Introduction

A script, which is generally neglected to be mentioned in the discussion on the Indian scripts, is MODI. MODI is the name of one of the scripts used to write the Marathi language, which is the primary language spoken in the state of Maharashtra in western India. The use of MODI in official Marathi documents and administration was common in Maharashtra till the end of 19th century. The British Government of Bombay Presidency in the beginning of 20th century for the sake of convenience and uniformity with the other areas of the presidency decided that the Devanagari (Baalbodh as it is called in Maharashtra) should be used as a primary writing system in administration.

MODI was widely used even in 1940s by the people of older generation for personal and financial documentation [Ramraje, (2013)]. MODI was an official script to write Marathi until the 20th century when the Baalbodh style of the Devanagari script was promoted as the standard writing system for Marathi [Wikipedia, (2015)]. There are around four crore Historical MODI documents at Peshwe Daftar, Pune itself. They are from the Era of Mughals, Shivaji Maharaj, Peshwe, and British. These days only a handful of people can read, interpret, and understand MODI Script. In order to get a true insight into the History, one must first be able to read/interpret such documents. It will take years for Human Transliteration of these documents. Several old land deeds and property documents are in the MODI script, and they need to decipher it to resolve issues in court. Transliteration of birth and/or death certificates written in MODI script into 'Devanagari' script will be immensely helpful.

Although the recognition of handwritten documents in Devanagari script is itself quite challenging, the recognition of handwritten documents in MODI script is even more challenging for the various (not all) reasons, viz., After detailed study, 1) It was found that more than 50 commonly used words have short forms, 2) All the words appearing in any text are written consecutively, i.e. there is no space between words, 3) Same representation is used for multiple characters, 4) One character is written in more than one ways, 5) Representation for letter modifier differs from letter to letter, i.e. Not all letters have consistent representation for letter modifiers, 6) Although a word contains an 'Anuswar', most of the times it is not written, and many others.

For the accurate recognition of these characters, segmentation will play a key role. Horizontal Projection Profile [HPP] yields proper segmentation of lines provided that each adjacent lines are separated from one another by at least one zero-row pixel. Due to the presence of several letter modifiers like 'maatra', 'velanti', 'ukar', 'anuswaar', and also because of varied writing styles of different people viz., skewed writing, indifferent heights of characters etc., it is observed that most of the adjacent lines were either touching each other or overlapping with one another or both, so mere HPP method alone does not suffice. The work presented in this paper uses the result obtained from HPP method as input and performs the segmentation of overlapping and/or touching lines in the handwritten text document by identifying and selecting valleys which minimizes the segmentation error as shown in figure 4 and 6. The structure of the paper is as follows: In Section 2, summary of the previous works on text segmentation, especially, in handwritten cursive script recognition is discussed. Preprocessing steps are described in Section 3. Section 4 elaborates the actual work carried out for this paper. Then, in section 5, experimental results are explained. Conclusion and future prospects are discussed in section 6. Finally the paper ends with acknowledgments and references in section 7 and 8 respectively.

## 2. Previous Works

This section highlights the work done by various researchers on MODI Script.

[Besekar and Ramteke (2013)] explain the theoretical analysis of MODI script according to recognition point of view. It explains the similarity between Devanagari and MODI Script. Structural features of individual characters is also discussed. Various difficulties in extracting structural features of the characters is highlighted too.

[Rathi *et al.* (2015)] propose a system for recognition of handwritten MODI script characters. Median filter is used for noise removal. Stentiford algorithm is used for thinning operation. The algorithm used for thresholding is not mentioned.
Structural similarities of standard characters and handwritten characters in MODI script are verified in [Ramteke and Katkar, (2013)] using measured structure similarity approach. Different classification, pre-processing, and segmentation methods and pattern recognition techniques like Kohonen Neural Network, and backpropagation neural network have been discussed. An attempt is made to apply measured structure similarity approach to off-line recognition of handwritten MODI characters.
[Besekar, (2011a)] proposes the recognition of a set of handwritten digits using mathematical morphology. A decision tree has been developed to aid in the classification process. At each node, a morphological operation is applied on the incoming digit, and the results dictate what the next step in the sequence should be.
A modified algorithm for the recognition of binary image (MODI Numerals) is presented in [Besekar and Ramteke, (2011b)]. Gray-scale images are considered. Initially, the image is normalized and its sparse matrix is also normalized. The algorithm is applicable after image binarization process and then it derives chain code for the image.
[Kulkarni *et al.* (2014)] presents efficiency of Zernike moments over Hu's 7 moment with zoning for automatic recognition of offline handwritten 'MODI' characters. 100 repetitions of each character were provided for training. Training samples used were 3220 which represents group of 70 samples for each 46 characters and testing sample contained 1380 which represents group of 30 samples for each 46 characters.
[Anam and Gupta, (2015)] have developed MODI Script Character Recognizer System (MSCR) using Otsu's Binarization algorithm and Kohonen neural network method by considering 22 different characters of MODI script including vowels and consonants to train the system. The sample data set (training images) of the characters was maintained by taking handwritten samples from different people.
[Kulkarni *et al.* (2015b)] presents efficiency of *Zernike* moments over Hu's moment for recognition of handwritten 'MODI' numerals and highlights that the Zernike moments are found to be more reliable and accurate features for recognition of handwritten MODI numerals as compared to Hu's seven invariant moments.
[Kulkarni *et al.* (2015a)] investigates recent advances in Handwritten Optical Character Recognition system for MODI script.
[Kumar *et al.* (2014)] applied water reservoir based technique and vertical projection profile method for identification and segmentation of touching characters in handwritten Gurumukhi words. Touching characters are segmented based on reservoir base area points.

## 3. Preprocessing Steps

The outline of the preprocessing phase is as follows:
**Step 1:** Sample Images are acquired/scanned at a spatial resolution of 300 dpi as shown in Figure 1.
**Step 2:** Each sample image is converted from RGB to gray-scale using rgb2gray function as shown in Figure 2.
**Step 3:** These gray-scale images are resized to square matrix of size 1024x1024 pixels.
**Step 4:** Bleed-through effect in these highly degraded document samples is largely removed by preserving most of the text image details by analyzing histogram peaks and

valleys obtained using [De Silva, *et al.* (2010)], thereby, selecting an appropriate threshold value for image binarization.
**Step 5:** The sample images are binarized using the threshold obtained in the previous step.
**Step 6:** For line segmentation, the input image obtained from previous step is padded with 15x15 window (shown in Figure 3) as it was observed that some characters were touching the border as shown in Figure 2.
**Step 7: S**um of all the nonzero values from each row of f(x,y) is taken and the result is stored in a column vector named 'hp'.

## 4. The Proposed Method

Major headings should be typeset in boldface with the first letter of important words capitalized.

### 4.1. *Problem Statement*

Let u = u(x) be an image resulted from scanning of a MODI Script document, where x = $[i, j]^T \in R^2$. The objective is to obtain individual lines accurately segmented from the document with minimal noise and interference of letter modifiers.

### 4.2. *Notations*

The algorithm uses several terms based on the analysis done on figure 4. For better understanding of the work, their notations and a brief discussion is given below:

- *Increasing Trend:* An increasing trend simply tells that in a given series of values, most of the times, next value will be larger than the previous value.
- *Decreasing Trend: A* decreasing trend tells that in a given series of values, most of the times, next value will be smaller than the previous value.
- *Segmentation Threshold 1:* A counter variable used during increasing trend. This counter variable is reset to zero whenever new maxima is located, whereas, it is incremented by one otherwise. When this counter variable reaches to $t_0$ (value obtained after experimental analysis, which is set as default), it is concluded that a current maxima is the true maxima.
- *Segmentation Threshold 2:* A counter variable used during decreasing trend. This counter variable is reset to zero whenever new minima is located, whereas, it is incremented by one otherwise. When this counter variable reaches to $t_0$ (value obtained after experimental analysis, which is set as default), it is concluded that a current minima is the true minima.

### 4.3. *Mathematical Formulation*

Let u = u(x) be an image resulted from the result of preprocessing steps, where x = [i, j]$^T \in \boldsymbol{R^2}$ with 'r' rows and 'c' columns. Pixel intensity of each row i, where $\mathbf{1} \leq \boldsymbol{i} \leq \boldsymbol{r}$ is formulated as a linear function of u defined as follows:

$$F(i) = \sum_{j=1}^{c} \boldsymbol{u}(\boldsymbol{x}) = \sum_{j=1}^{c} \boldsymbol{u}([\boldsymbol{i}, \boldsymbol{j}]^T) \tag{1}$$

Now, the Difference Operator ∇ of function F is defined as:

$$\nabla F(i) = F(i+1) - F(i) \tag{2}$$

The mathematical formulation to identify increasing trend in the current line is given as follows:

A new function called 'G' is defined to identify true maxima, and two more functions called 'T' and "I" are also defined to identify current true maxima, and position of that true maxima respectively.

Consider

$$\begin{gathered} G(1) := 0; \\ T(1) := 0; \\ I(1) := 0; \\ \left.\begin{gathered} G(j+1) := F(i+1) \\ T(j+1) := 0 \\ I(j+1) := i+1 \end{gathered}\right\} if\ \nabla F(i) > 0 \\ \left.\begin{gathered} G(j+1) := G(j) \\ T(j+1) := T(j)+1 \\ I(j+1) := I(j) \end{gathered}\right\} otherwise \end{gathered} \tag{3}$$

When T(j+1) = Thresholding value $t_0$, Then value of G(j+1), denoted as M will give the true maxima (True Peak) in the current line, and I(j+1), denoted by P, will give its position.

Similarly, the mathematical formulation to identify decreasing trend in the current line is given as follows:

A new function called 'G' is defined to identify true minima, and two more functions called 'T' and "I" are also defined to identify current true minima, and position of that true minima respectively. Consider

$$\begin{gathered} G(1) := M; \\ T(1) := 0; \\ I(1) := P; \\ \left.\begin{gathered} G(j+1) := F(i+1) \\ T(j+1) := 0 \\ I(j+1) := i+1 \end{gathered}\right\} if\ \nabla F(i) < 0 \end{gathered}$$

$$\left.\begin{matrix} G(j+1) := G(j) \\ T(j+1) := T(j)+1 \\ I(j+1) := I(j) \end{matrix}\right\} otherwise \tag{4}$$

When T(j+1) = Thresholding value $t_0$, Then value of G(j+1) will give the true minima (True Valley) in the current line, and I(j+1) will give its position.
Let $L = \{[r,s]^T \in R^2 : I(j) \leq r \leq I(j+1), 1 \leq s \leq c\}$ be the current line used for further postprocessing.
Now, a new function called 'H' is defined as

$$H_i(j) = u(i,j) \tag{5}$$

Let $\chi$ be a characteristic function of positive integers. Then the following function called as $\phi$ is defined which gives the column indices in a row 'i' whose pixel intensity is greater than zero:

$$\phi_i(j) = j\chi o H_i \tag{6}$$

If $\phi_i(j) \neq 0$ then two new functions $F_j$ and $\psi_j$ are defined respectively as follows:
$F_j(i) = u(i,j-1) + u(i,j) + u(i,j+1)$ gives sum of intensities of the pixels in $i^{th}$ row, $j^{th}$ column, and left and right neighbors of j. This function identifies the possibility of overlapping text of previous/next line into the current line.
$\psi_j(i) = 1 - \chi o F_j(i)$ is identically zero to the extent of the overlapping text of previous/next line into the current line.
$If\ \exists\ I(j+1) \geq i \geq I(j+1) - \frac{I(j+1)-I(j)}{2}\ such\ that\quad \psi_j(i) \neq 0,$ then the connected component of overlapping text of next line into the current line is located and shall be removed, otherwise, it is considered as part of the current line only and hence shall not be removed.
$If\ \exists\ I(j) \leq i \leq I(j) + \frac{I(j+1)-I(j)}{2}\ such\ that\quad \psi_j(i) \neq 0,$ then the connected component of overlapping text of previous line into the current line is located and shall be removed, otherwise, it is considered as part of the current line only and hence shall not be removed.

The function $f(i,y) = y\chi\big(u(i,y)\big)$ gives the column indices of nonzero pixels in the $i^{th}$ row, and the function $g(i,y) = f(i-1,y-1) + f(i-1,y) + f(i-1,y+1)$ gives the column indices of nonzero pixels which are adjacent to pixel (i, y). Applying this function $g$ iteratively until $g^k = g\ o\ g\ o\ \ldots o\ g\ (i,y)\quad k\ times$ becomes eventually zero and k does not exceed r/3, where r is the number of rows in the current line. The connected component which is either touching the upper border of the current line or very near to it but not exceeding beyond the $\frac{1}{3}$ of r is considered as a part of the lower zone of previous line. Thus, this function g identifies the upper zone of current line which is actually the part of the lower zone of the previous line.

The function $f(i,y) = y\chi\big(u(i,y)\big)$ gives the column indices of nonzero pixels in the $i^{th}$ row, and the function $g(i,y) = f(i+1,y-1) + f(i+1,y) + f(i+1,y+1)$ gives the column indices of nonzero pixels which are adjacent to pixel (i, y). Applying this function $g$ iteratively until $g^k = g\ o\ g\ o\ \ldots o\ g\ (i,y)\quad k\ times$ becomes eventually

zero and k does not exceed r/3, where r is the number of rows in the current line. The connected component which is either touching the lower border of the current line or very near to it but not exceeding beyond the $\frac{1}{3}$ of r is considered as a part of the upper zone of the next line. Thus, this function g identifies the lower zone of current line which is actually the part of the upper zone of the next line.

Let $C_1, C_2, \dots, C_n$ be the connected components of current line with $|C1| \geq |C2| \geq \cdots \geq |Cn|$

$where\ |Ci| = \sum_{(i,j)\in C_i} \chi\big(u(i,j)\big)$ is the number of pixels in the connected component $C_i$. and $\chi$ is a characteristic function of positive integers. Clearly, by assumption $C_1$ is maximal component.
Let t denote median of the row indices of rows in $C_1$. Let $\chi_t and\ \chi_{c_i}$ be characteristic functions of the row with index 't' and component $c_i$ respectively. Therefore, function $f = \chi_{c_1} + \chi_t(\sum_{i=2}^{n} \chi_{c_i})$ is characteristic function of true text in the current line, and the function $g(i,j) = u(i,j)f(i,j)$ gives the true text in the current line, thereby, eliminating remaining noise from the line.

### 4.4. *Proposed Algorithm (Module 1)*

The algorithm SegmentLines used to segment lines is described as follows:

*Algorithm SegmentLines()*

*Input:* f(x,y) and result obtained from HPP method

//Input image padded with 15x15 window, 'hp' array, and $t_0$

// which is default to 50

*Output:* x[m] and y[n] // Two vectors, say x, and y, of size m and n respectively.
// First vector x representing all the maxima shown in the figure 4.
// Second vector y representing all minima obtained from the valleys shown in figure 4.
{ // begin
n := 1;
y[n] := Return_Index_of_First_NonZero_Value_In(hp);
// this represents start of the first line
y[n] := y[n] -1;
n := n + 1;
// make y[1] point to the last false minima before start of the first line
curmax := 0;
flag := 0;
// flag value zero indicates increasing trend whereas,
// flag value 1 denotes decreasing trend
i := 1;
m := 1;
repeat until end-of array(hp)
{

```
if flag = 0
{
        while curmax is < hp[i]
        {
                curmax := hp[i];
                j := i;
// index of false maxima (current peak)  along the current line is recorded

                counter1 :=0; // Segmentation Threshold 1
                i := i + 1;
        }
        if curmax is > hp[i]
                counter1:= counter1 + 1;
        if counter1 = t0
        {
// it implies that a true maxima along the current line has been reached
                x[m] := j;
                m: = m + 1;
                flag: = 1;
                curmin := curmax;
                curmax := 0;
                counter1 := 0;
                i := j;
                Go back to repeat-until Loop
        }
}
else if flag = 1
{
        while curmin is > hp[i]
        {
                curmin := hp[i];
                l := i;
// index of false manima (current valley)  along the current
// line is recorded
                counter2 :=0; // Segmentation Threshold 2
                i := i + 1;
        }.
        if curmin is < hp[i]
                counter2:= counter2 + 1;
        if counter2 = t0
        {
// it implies that a true minima along the current line has been reached
                y[n] := l;
                n: = n + 1;
                flag: = 0;
                curmax := curmin;
                counter2 := 0;
                i := l;
```

```
                    Go back to repeat-until Loop
            }
}
i := i + 1;
}
return (x,m,y,n);
}
```

Finally, individual lines are segmented using the indices obtained in second output array y as shown in figure 5.

### 4.5. *Proposed Algorithm (Module 2)*

The algorithms which removes lower zone of previous line and upper zone of next line from the current line are described as follows:

*Algorithm RemoveCCofPrev*

*Input:* Current Line (curline), Spatial Coordinates of Upper Border Non-zero pixel (r and c), Number of rows in current line (sr), extent (stop) which is initialized to 1.

*Output:* Current Line without lower zone of previous line

```
{ // begin
        if any of the values of r and/or c are zero or if r is equal to sr or if intensity value of curline(r,c) is zero, then
        return SUCCESS // changes are saved
        else
        {
        curline(r,c) := 0;
        recursively call the same function for the next row, r+1 for the values of c-1, c, and c+1 respectively.
        stop := stop + 1;
        if stop >= sr/3
                return FAILURE // changes are not saved
}
```

*Algorithm RemoveCCofNext*

*Input:* Current Line (curline), Spatial Coordinates of Lower Border Non-zero pixel (r and c), Number of rows in current line (sr), extent (stop) which is initialized to 1.

*Output:* Current Line without upper zone of next line

```
{ // begin
        if any of the values of r or c is zero or if intensity value of curline(r,c) is zero, then return SUCCESS // changes are saved
        else
        {
        curline(r,c) := 0;
        recursively call the same function for the previous row, r-1 for the values of c-1, c, and c+1 respectively.
        stop := stop + 1;
        if stop >= sr/3
                return FAILURE // changes are not saved
}
```

The result is shown in Line Number 5 (before and after removal of Connected Component) in Figure 6.

The algorithms which identifies part of the upper zone of current line segmented into the previous line as well as lower zone of the current line segmented into the next line and places them into the current line segment are described as follows:

*Input:* previous line or next line

*Output:* A matrix containing upper zone of current line segmented into the previous line or lower zone of the current line segmented into the next line.

{

Scan the line column by column until all the columns are exhausted

For every possible legal value of column, travel row-by-row until either a non-zero valued intensity is reached (marking the start of the connected component) or $\frac{1}{3}$ rd of the height of the number of rows in the lines is reached. Each time, record the largest value of row. This final largest value of row determines the number of rows of the output matrix.

Append this output matrix into the current line at an appropriate end.

}

The result obtained is shown in Line Number 4 (before and after addition of connected component) in Figure 6.

Finally, only the true text in the current line is retained by selecting the largest connected component and eliminating all other connected components, which are either data of adjacent lines or leftover noise as shown in Line Number 7 (before and after removal of unwanted text and leftover noise) in Figure 6.

## 5. Experimental Results and Discussion

Original noisy image is shown in figure 1. The bleed-through effect is clearly visible. Some lines are overlapping with one another. Figure 2 depicts gray-scale image. Figure 3 contains binarization result obtained using [De Silva, *et al.* (2010)]. Figure 4 shows the graphical plot of array 'hp' which is used for the valley selection to minimize the segmentation error. Each x co-ordinate represents row number of the document image, whereas, every y co-ordinate gives number of nonzero pixel intensity values along that row. Figure 5 shows the result obtained using module 1 of the algorithm presented in this paper. It shows how lower zone of previous line segmented into the current line is removed from it. Figure 6 shows the result obtained using module 2 of the algorithm presented in this paper. It adds the eliminated portion into the previous line. False versus True Minima is depicted in Figure 7.

The module 1 of the algorithm is based on two interesting observations obtained from figure 4. The first one is that the number of peaks clearly tells number of lines present in the image. The second important observation is that number of pixels are usually increasing (but not strictly increasing, making segmentation task more challenging) until a peak is reached while passing through a line, similarly, the number of pixels are decreasing (but not strictly decreasing) after the peak is crossed till the start of the new line.

The module 1 of the algorithm finds out such trends and uses it to perform line segmentation effectively. The module 2 of the algorithm correctly places the relevant text into the appropriate lines and also removes unwanted text of adjacent lines and the leftover noise as well.

Figure 1: Original RGB Image

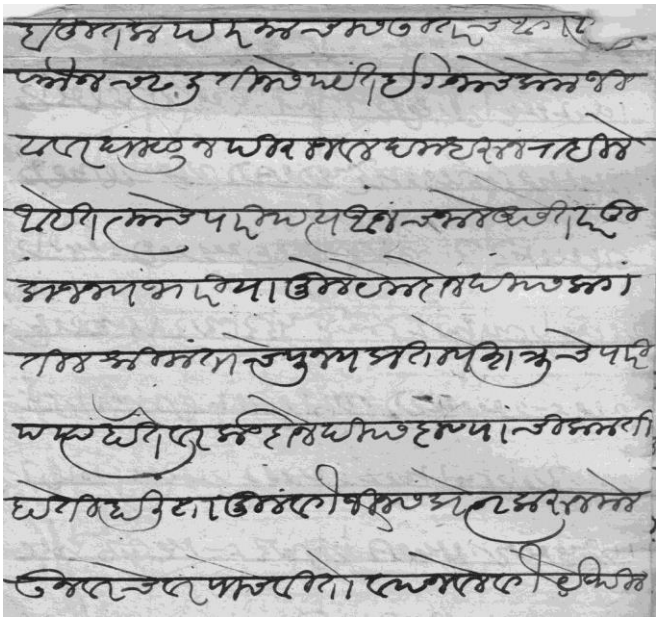

Figure 2: Gray-Scale Image

Figure 3: Binarized and padded image using [De Silva, *et al.* (2010)]

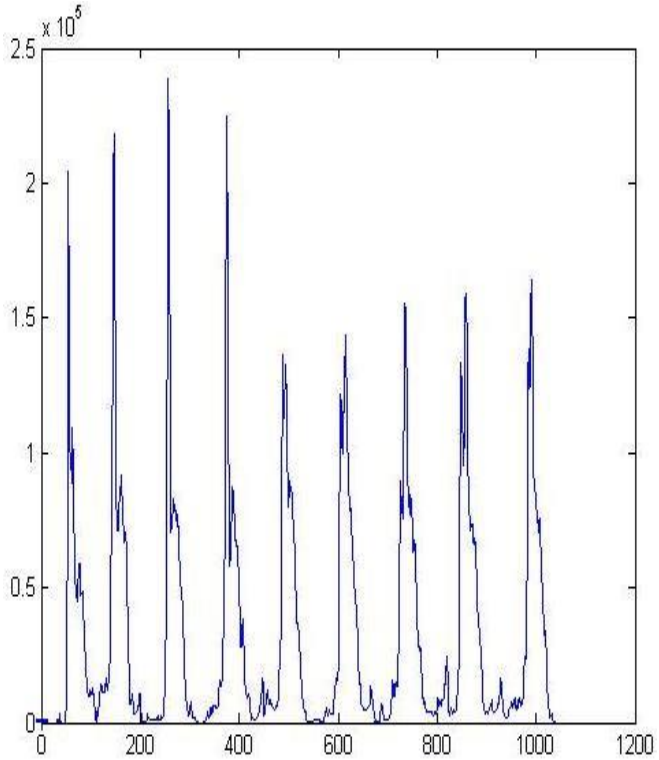


Figure 4: The array 'hp' shown graphically

Figure 5: Lines Segmentation Algorithm (Module 1) Result

Line Number 5 (Before removal of connected component)

Line Number 5 (After removal of connected component)

Line Number 4 (Before addition of connected component)

Line Number 4(After addition of connected component)

Line Number 7 (Before the removal of leftover noise)

Line Number 7 (After removal of leftover noise)

Figure 6: Line Segmentation Algorithm (Module 2) Result, Line Number 5, 4, and 7

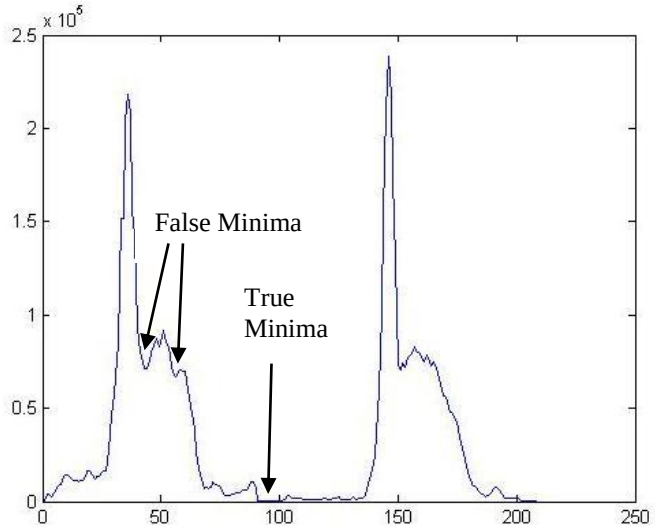


Figure 7: False vs. True Minima

## 6. Conclusion and Future Prospects

The methods employed in this work are simple and computationally inexpensive. The results obtained by using this algorithm are encouraging even though the document contains overlapping and/or touching lines as it is based on selecting valley which minimizes the segmentation error. Furthermore, it identifies part of the upper zone of current line segmented into the previous line as well as lower zone of the current line segmented into the next line and places them into the current line segment. Moreover, it also removes lower zone of previous line and upper zone of next line from the current line.

Future work involves extracting individual characters from a line. For accurate character segmentation, skew correction shall also be performed. Character Segmentation will become simpler if shirorekha is removed from the line as it will increase the probability of having isolation among characters.

## 7. Acknowledgments

The Authors would like to thank Vice-Chancellor of Deccan College Deemed University, Pune, Maharashtra for giving access to historical documents written in Modi script.
In particular, authors are deeply grateful to Dr. Girish Mandke, Curator, Maratha History Museum, Deccan College for providing the data set.
The authors are thankful to the University Grants Commission, New Delhi for supporting this research at School of Computer Sciences, North Maharashtra University, Jalgaon under the Special Assistance Programme (SAP) at the level of DRS-II.

## 8. References

Anam S.; Gupta S. (2015): An Approach for Recognizing Modi Lipi using Ostu's Binarization Algorithm and Kohenen Neural Network: International Journal of Computer Applications, Volume 111, no. 2, pp. 28-34.

Besekar, D. N. (2011a): Recognition of Numerals of MODI Script Using Morphological Approach: International Referred Research Journal, ISSN- 0974-2832, RNI-RAJBIL, 2009/29954, Volume III, Issue 27, pp. 63-66.

Besekar D.N.; Ramtecke R.J. (2011b): A Chain Code Approach For Recognising Modi Script Numerals: Indian Journal of Applied Research, Volume 1, Issue 3, ISSN - 2249-555X.

Besekar, D. N.; Ramteke R.J. (2013): Study for Theoretical Analysis of Handwritten MODI Script– A Recognition Perspective: International Journal of Computer Applications 64, no. 3, pp. 45-49.

De Silva D. V. S., *et al.* (2010): Adaptive sharpening of depth maps for 3D-TV. Electronics Letters 46, no. 23, pp. 1546-1548.

Kulkarni S. A. *et al.* (2014): Offline Handwritten MODI Character Recognition Using HU, Zernike Moments and Zoning: arXiv preprint arXiv:*1406.6140*.

Kulkarni S. A., *et al.*(2015a): Review on Recent Advances in Automatic Handwritten MODI Script Recognition: International Journal of Computer Applications (0975–8887) 115, no. 19.

Kulkarni S. A., *et al.*(2015b): Analysis of Orthogonal Moments for Recognition of Handwritten MODI Numerals: 2nd Natural language Processing and Data Mining (NLPDM), VNSGU, Surat, SNLPDM04, pp. 1-6.

Kumar M.; Jindal M.K.; Sharma R.K. (2014): Segmentation of isolated and touching characters in offline handwritten gurmukhi script recognition: International Journal of Information Technology and Computer Science (IJITCS) 6, no. 2, pp. 58-63.

Ramraje R.A. (2013): HISTORY OF MODI SCRIPT IN MAHARASHTRA: Global Online Electronic International Interdisciplinary Research Journal (GOEIIRJ); Volume II, Issue I, ISSN : 2278 – 5639, pp. 54-58.

Ramteke, A. S.; Katkar G.S. (2013): Recognition of Off-line Modi Script: A Structure Similarity Approach: International Journal of ICT and Management, ISSN No. 2026-6839, Volume I, Issue I, pp. 12-15.

Rathi S.R.; Jadhav R.H.; Ambildhok R.A. (2015): Recognition and Conversion of Handwritten MODI Characters: International Journal of Technical Research and Applications, e-ISSN: 2320-8163, Volume 3, Issue 1, pp. 128-131.

Wikipedia, (2015): Modi Alphabet, From Wikipedia, the free encyclopedia, https://en.wikipedia.org/wiki/Modi_alphabet