International Journal of Computer Theory and Engineering, Vol. 2, No. 3, June, 2010
1793-8201

Detection and segmentation of Handwritten Text in Gurmukhi Script using Flexible Windowing

Rajiv Kumar and Amardeep Singh

***Abstract* – To make use of non editable scanned image of thedocument, one has to pass through the recognition process. The recognition process consists of sub processes like pre processing, segmentation and then recognition. Segmentation process is the most significant process because if the segmentation is incorrect then we can not have the correct result, it is just like garbage in and garbage out. On the same time it is one of the complex processes too. It is more difficult if the document is handwritten because in that case only few points are there which can be used to make segmentation. In this paper, we tried to formulate a procedure which is used to segment the scanned document image into lines then into words and finally to characters. For that purpose we used the concept of flexible window, that is, the window whose size can be adjusted according to needs. One module is designed to find the window. Same module is used to get the different types of outputs (lines, words, and characters) with a little bit adjustment to parameters passing as well as to the procedure itself. The concept was applied to different documents and we got good reasonable results.**

***Index Terms* — OCR, Segmentation, Handwritten, Flexible, Window.**

I. Introduction

The objective of automatic document processing is to recognise text, graphics and pictures in digital images and extract the intended information, as would a human. Textual and graphical are two categories of document processing dealing, respectively, with the text and the graphics components of a document image. Document processing leads to the theory of optical character recognition (OCR). But before recognising the character it is required to segment area which would have that character. So to have such area, a perfect segmentation of characters is required before individual characters are recognized. Therefore segmentation techniques are to apply to word images before actually putting those images to reorganization process. The simplest way to segment the characters is to use inter – character gap as a segmentation point. However, this technique results in partial failure if the text to be segmented contains touching characters. If the document is made up of handwritten characters then the situation becomes complex. Our work is related with the segmentation of handwritten text written in Gurmukhi script which is one of the popular scripts used to write Punjabi , a popular spoken language of northern India.

Gurmukhi script alphabet consists of 41 consonants and 12 vowels. Besides these, some characters in the form of half characters are present in the feet of characters. Writing style is from left to right. In Gurmukhi, there is no concept of upper or lowercase characters. A line of Gurmukhi script can be partitioned into three horizontal zones namely, upper zone, middle zone and lower zone. Consonants are generally present in the middle zone. These zones are shown in figure1. The upper and lower zones may contain parts of vowel modifiers and diacritical markers.

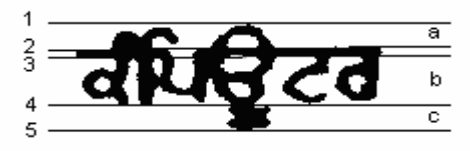

Figure 1 : a) Upper zone from line number 1 to 2,
b) Middle Zone from line number 3 to 4,
c) lower zone from line number 4 to 5

II. Pre Processing

The image file is in grey scale. But we require two types of information – either zero or one. For that purpose, we calculated the average of intensities of all the pixels present in the document image file. Then the intensity of each pixel is set as per the following rule:

if (pixel intensity < Average intensity) then
pixel intensity = 0
else pixel intensity = 1

The quality of scanned image depends upon the scanner type too and it plays an important role in segmentation. We are using higher end scanner for the scanning purposes. But even if some impurities are introduced due to used paper quality or due to scanner quality then these are taken care by using anti – windowing concept. As per this concept, first the area is searched with a window of width dw. If there is only few pixel and nothing is around the window then those intensities of present pixel values are set to zero. The words and characters are handwritten. Between any two lines and any two words there is a definite gap of minimum width. A line is supposed to have different words and the words are made up of one or more characters. The lines consisting of words are generally straight in nature. If there is any skew then present work may not work properly.

III. Flexible Window

Flexible window is the window which can be adjusted as per the change in the requirements. Here is the concept to get a window which is flexible in size:

From the image's starting point (generally it is 0, 0), get first pixel position present in a row and move towards right side to get right most pixel position i.e. get first and last

pixel positions of a row. Suppose there are n rows of pixel say r1, r2, ……., rn in a line of a document. There would be n pixel position for left most side and n pixel positions for right most side too. These are Lx1, Lx2, ……, Lxn for left side and Rx1, Rx2, ……, Rxn on right side.

Find MIN X = gub(Lx1, Lx2, ……, Lxn) // gub – greatest upper bound

MAX X = llb(Rx1, Rx2, ……, Rxn) // llb – least lower bound

To have the horizontal boundary of the window, we are using the concept of no pixel zone i.e. between two no pixel zones there is a line. Therefore take a upward movement. Suppose there are n columns of pixel, say c1, c2, ……., cn, in a line of a document. There would be n pixel positions for bottom most side and n pixel positions for uppermost side near the no pixel zone too. Suppose Ly1, Ly2, ……, Lyn are the n pixel positions for bottom side and Ry1, Ry2, ……, Ryn on upper side.

Find MIN Y = gub(Ly1, Ly2, ……, Lyn)

MAX Y= llb(Ry1, Ry2, ……, Ryn)

A window with coordinate (MINX, MINY), (MINX, MAXY), (MAXX, MAXY) and (MAXX, MINY) is formulated. These window coordinates are written in a file. So in this way, all the lines are identified and their positions as window coordinates are written in a file.

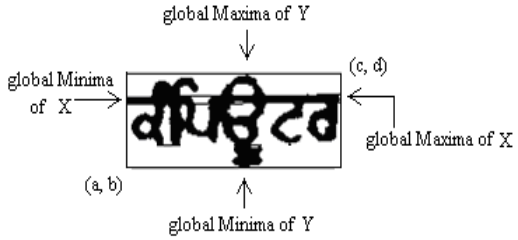

Figure 2 : Window with coordinates (a, b) and (c, d)

As we know that a line in the document consists of one or more words. Then next step is to find the positions of words present in a particular line. For that, get the coordinate of the window coordinates of a line. Start traversal from the left most – bottom most corner of the window. Get the first dot position say it is (a, b). Start upward movement; get the position of last dot of this column before the no pixel zone. Scan the second column; get the pixel position of this column. Somewhere there would be one or two columns which are having no dots. Note down the position of such columns. Now we have found a cluster of pixels. The position of this cluster is noted down in a file. Repeat the process for each cluster present in a line. Repeat the process for each line.

If the size of window found is less than w (negligible) then return 0, means nothing is present there.

Once the lines are identified, now this is the turn to find the word positions present in a line. Start from the starting position of the line. Between two words there is no pixel zone. So the concept, how to get window for line, discussed above can be used to find the words but here would be a difference in use. Now X would be Y and Y would be X. Earlier in case of line, it was XY plane now it would be YX plane i.e. the same module is used but the approach towards parameters is +90 degree. We can say that by changing the window size we can have words present in the line. The words positions are written down in the file. The file is having line information as well as words information corresponding to each line. We are moving from macro to micro level.

Structure used to represent a Point

```
POINT
{
        int X
        int Y
}
```

The concept is put in the form of pseudo code, to find the window, is given below.

To get the Vertical Boundary as well as Horizontal Boundary position of the window, two modules are written. These modules are packaged in another module named BOUNDARY.

```
BOUNDARY(SLOC, MAX, MIN, OFILE)
{
Call Vertical Boundary (SLOC as POINT, d as integer, X as integer, MAXY as integer, MINY as integer)
Call Horizontal Boundary (SLOC as POINT, MAXY as integer, MAXX as integer, MINX as integer)
Write in OFILE, MINX,MINY, MINX, MAXY, MAXX, MAXY, MAXX, MINY
}
```

Input parameter

SLOC is of POINT type and it would be MINX and MINY

Output parameter

MAX Y is of Integer type

MIN Y is of Integer type

d can be a predefined value

or can be calculated as d = First no Pixel Zone Area – Second no pixel zone area

X Horizontal width of the image // can be obtained from the file information.

```
Vertical Boundary(SLOC, d, X, MAXY, MINY)
 { MAXY=SLOC.Y
     MINY = –1
     For I = SLOC.X      to      X
        {Y=SLOC.Y
         while((MAXY – Y) < d)
          {  if(Pixel Present)
               {Current _Col_Pixel = Pixel Position
                If (MINY== –1) MINY = Current_Col_Pixel.Y
                If((Current_Col_PixelY – MINY)<0)
                          MINY= Current_Col_Pixel.Y
If ( (Current_Col_Pixel.Y - MAXY)> 0 AND (Current_Col_PixelY–MAXY)< d)
                MAXY = Current_Col_Pixel.Y
                }
               y = y + 1
          }
        }
}
```

```
Horizontal Boundary (SLOC, MAXY, MAXX, MINX)
{MINX= – 1
 For I = SLOC . Y to MAXY
    { X = SLOC.X
     For J = SLOC.X to X
     {
       If(Pixel Present)
       { Current_Row_Pixel = Pixel Position
          If (MINX = = – 1) MINX = Current_Row_Pixel.X
                  Else If (( MINX – Current_Row_Pixel.X)>0)
                           MINX = Current_Row_Pixel.X
                  If ((MAXX – MINX) > 0)
                           MAXX = Current_Row_Pixel.X
       }
     }

     }
}
```

IV. Implementation And Result Discussion

The above idea is implemented using Visual Basic 6.0. The authors are good in Programming with Visual Basic 6.0. There were two choices either to learn a new language or to explore the already known language. We opted the second option. Initially it was very difficult to find a way to process a BMP file in VB 6. But later with consistent efforts we were able to find out the way. Here we are discussing some basic information regarding the BMP file. The file header as well as Info header, used to read the BMP file, is given below:

```
Private Type BITMAPFILEHEADER
        bfType          As Integer
        bfSize          As Long
        bfReserved1     As Integer
        bfReserved2     As Integer
        bfOhFileBits    As Long
End Type
```

As we know, generally, two bytes are used for Integer and four bytes are used for Long, therefore the first 14 bytes of information from the BMP file is extracted with the help of this User-Defined Data Type. bfType will always return 19778 which represents two character string, "BM" for bitmap. This is a standard practice that all bitmaps start with these two characters. The entire file's size in bytes is described by bfSize, and bfReserved1 and bfReserved2 are reserved spaces and these should be set to zero. The byte offset, from the beginning of the file at which the bitmap data starts, is given by the value of bfOffBits. So if the value bfOffBits is 1078 (as it should for most 8bit bitmaps) then the header and color table will be over by the 1078th byte, and after that the picture data is going to start.

```
Private Type BITMAPINFOHEADER
        biSize          As Long
        biWidth         As Long
        biHeight        As Long
        biplanes        As Integer
        biBitCount      As Integer
        biCompression   As Long
        biSizeImage     As Long
        biXPelsPerMeter As Long
        biYPelsPerMeter As Long
        biClrUsed       As Long
        biClrImportant  As Long
End Type '40 bytes
```

The size of the **BITMAPINFOHEADER** structure is in **biSize** and it is given in bytes (generally, equals 40). The width and height of the bitmap in pixels is described by **biWidth** and **biHeight**. **biPlanes** describes the number of planes contained in the bitmap, this is not normally used, and is set to one.

biBitCount describes the "bit-depth" of this bitmap. It can have any of four values: 1, 4, 8, and 24. If the value of bit depth taken as 1 then it indicates that the bitmap will have only two colors (monochrome), a value of bit depth as 4 will allow 16 colors, 256 colors when the value is as 8, and if it is equal to 24 then it means that 16.8 million colors.

biXPelsPerMeter and **biYPelsPerMeter** describe the resolution of the bitmap in pixels/ meter. If one wants to use standard resolution, then just set these to zero. **biClrUsed** indicates how many of the colors from the color table are actually used in the bitmap, if it is set to zero then you are allowed to use all colors. **biClrImportant** tells about that which colors are most important, this information can be used by the programmer. If this is set to zero then standard operation are available. After reading the file header as well info header, then using the normal VB code, BMP file can be processed. After successfully implementing the concept, we got result which are presented in the following tables:

TABLE 1: ACCURACY FOR LINE SEGMENTATION

Document	No of Lines	Correctly Detected	Inaccurate segmentation	Accuracy
Doc1	15	13	2	86%
Doc2	18	16	2	88.88%
Doc3	20	17	3	85%
Doc4	33	30	3	90.9%

TABLE 2: ACCURACY FOR WORD SEGMENTATION

Document	No of Lines	Correctly Detected	Inaccurate segmentation	Accuracy
Doc1	68	62	6	91.17%
Doc2	76	68	8	89.47%
Doc3	95	83	16	87.36%
Doc4	120	101	19	84.16%

TABLE 3: ACCURACY FOR CHARACTER SEGMENTATION

Document	No of character	Correctly Detected	Inaccurate segmentation	Accuracy
Doc1	99	89	10	89.89%
Doc2	158	140	18	88.60%
Doc3	237	188	49	79.32%
Doc4	297	242	55	81.48%

V. Conclusion

Once the idea is implemented and is applied on number of scanned documents, the results were encouraging. The lines were detected with a great accuracy. Few lines were there which were detected inaccurately. The inaccuracy was because of lower zone. The lines which were having some characters in the lower zone were interpreted wrongly i.e. the characters present in the lower zone were treated as a separate line. Similarly, for the words, the segmentation was good. There was some wrong segmentation for characters. This was due to the shape of some of Gurumukhi characters.

There are certain characters which are combined in nature. As a single unit they are segmented well but when it is expected to segment these in two characters then module does not give the same. But this experience gives a great satisfaction to the authors that same module is used for different type of outputs and it works well. Now we are trying to upgrade the code so that we can have more improved results.

REFERENCES

[1] Y. Lu. "Machine Printed Character Segmentation – an Overview". Pattern Recognition, vol. 29(1): 67-80, 1995
[2] Rajiv K. Sharma and Amardeep Singh, "Segmentation of Handwritten Text in Gurmukhi Script", International Journal of Computer Science and Security, volume (2) issue (3), 2008
[3] M. K. Jindal, G. S. Lehal, and R. K. Sharma. "Segmentation Problems and Solutions in Printed Degraded Gurmukhi Script". IJSP, Vol 2(4),2005
[4] G. S .Lehal and Chandan Singh. "Text segmentation of machine printed Gurmukhi script". Document Recognition and Retrieval VIII, Proceedings SPIE, USA, vol. 4307: 223-231, 2001
[5] Veena Bansal and R.M.K. Sinha. "Segmentation of touching and Fused Devanagari characters, ". Pattern recognition, vol. 35: 875-893, 2002.
[6] R. G. Casy and E. Lecolinet. "A survey of methods and strategies in character segmentation". IEEE PAMI, Vol. 18:690 – 706,1996.
[7] U. Pal and Sagarika Datta. "Segmentation of Bangla Unconstrained Handwritten Text". Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR), 2003.
[8] U. Pal, S. Sinha and B. B. Chaudhuri. "Multi-Script Line identification from Indian Documents", Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR) 2003.
[9] Rajean Plamondon, Sargur N. Srihari. "On – Line and Off – Line Handwritting Recognition: A Comprehensive Survey", IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol 22(1). Janurary, 2000.
[10] Giovanni Seni and Edward Cohen. " External word segmentation of off – line handwritten text lines". Pattern Recognition, Vol. 27(1): 41-52, 1994.
[11] http://www.vb6.us/tutorials/handling-bitmaps-tutorial

Rajiv K. Sharma obtained his M.Tech (CSE) and is currently doing Ph.D. in Faculty of Engg. from the Punjabi University, Patiala, Punjab, India. At present, he is the Assistant Professor in the School of Mathematics and Computer Applications, Thapar University, Patiala. He has published several articles in journals and conferences.

Amardeep Singh obtained his M.Tech (CSE) from the Punjabi University, Patiala, Punjab and Ph. D. from Thapar University, Patiala. He is Reader, in Department of Computer Engineering, University College of Engineering, Punjabi University, Patiala. He has published several articles in journals and conferences.