C# Implementation of Split and Merge Algorithm for Image Segmentation

Velin Iordanov[1] and Ivo Draganov [2]

Abstract – **In this paper, we propose a C# implementation of the split and merge algorithm with a main applicability directed towards image segmentation. The algorithm process color and grayscale images by splitting them following quad-tree decomposition over intensity homogeneity criterion. Then adjacent blocks, including such from different tree levels, are merged together to form candidate regions for segmentation. Area filtration of the regions is the final processing step prior to generating the output map of the application. Testing is done with images containing printed text over complex background and promising results are registered.**

Keywords – **Image Segmentation, Split and Merge, C#, Area Filtration, Text Extraction.**

I. INTRODUCTION

Object detection in digital imagery and its recognition is a research area in which the main goal is to create a program for automatic segmentation of content visually embedded on a complex background.

Methods for object detection, in particular printed text, on a complex background form three groups: bottom-up [7, 12, 14], heuristic top-down [13, 2, 14] and top-down with machine learning [6, 8].

In "bottom-up" methods, the location of the text is not actually determined, it directly segments the image of areas, and those collapsed as symbolic are merged into words. Lienhart [7] segments partitioned images with a split-merge algorithm [5] and a growing region algorithm [9], while Bunke [12] groups the pixels of the text using a color clustering algorithm .These methods are sensitive to the size of the symbols, the available noise and the background type.

Through heuristic "top-down" methods, text blocks are first identified in the image processed by heuristic filters, then segmented into text and background areas. Difficulties arise here, both in detection and in segmentation. In [14] it is suggested that the text is contained in areas with a highly variable horizontal structure and additional spatial properties to be established by analyzing the related components. Smith et al. [11] localize the text by first locating the vertical edges with a preformed pattern, and then grouping them into text areas through a blurring process. The latter two approaches are quick, but many false searches occur, because often parts of the background also have a highly contrasting horizontal structure. Wu et al. [13] describe a text localization method based on texture segmentation. The method has a high computational complexity and is sensitive to background noise. In a more recent work Garcia [2] suggests a localization of text called a change in edge orientation based on the fact that text strings contain edges of different orientation. The disadvantage of the method is the non-use of multiple parallel edges, also characteristic of text strings. Apart from the peculiarities of the individual symbols, Sobettka et al. [12] offer detection based on a string when locating it.

Text detection problems - top-down heuristic methods are empirical and rely on predefined (operator) visual signs that often prove to be inappropriate on a complex background; multiple false searches (80-500%) are generated; learning methods somewhat solve this problem, but the following issues remain problematic: how to avoid the considerable time of classification applied to the entire image and how to reduce the variation in character size and change in brightness in terms of more stable features obtained in the pre-training phase? The process of normalization of the signs here occupies a central place as an object of intensive research.

In this study we propose a new direct approach of detecting isolated printed characters in images based on their size falling in preliminary defined limits over horizontals and verticals. The segmentation is based on contrast discrimination in small area employing the split-and-merge algorithm which attempts to overcome the major limitations of the aforementioned realizations. It is considered fast and reliable tool given a priori knowledge of the font sizes within processed images.

In Section II of the paper full description of the implemented algorithm is presented. It is done in C# following a flexible object-oriented module, also described as a structure in the same section. In Section III some experimental results are presented and then discussed in Section IV followed by a conclusion in Section V.

II. IMPLEMENTATION DESCRIPTION

A. Algorithm steps

Grayscale images are directly passed to the input of the system. If the image is in color then we covert it grayscale according to:

$$m[i][j] = 0.3 * r[i][j] + 0.59 * g[i][j] + 0.11 * b[i][j], \quad (1)$$

where $i=[0,M-1]$, $j=[0,N-1]$ and the resulting array should be of type double for preserving precision. After all calculations we go back to unsigned char representation within the range from 0 to 255.

[1]Velin Iordanov is with the Faculty of Telecommunications at Technical University of Sofia, 8 Kl. Ohridski Blvd, Sofia 1000, Bulgaria, E-mail: velin.iordanov@gmail.com

[2]Ivo Draganov is with the Faculty of Telecommunications at Technical University of Sofia, 8 Kl. Ohridski Blvd, Sofia 1000, Bulgaria, E-mail: idraganov@tu-sofia.bg

The obtained array *m[M][N]* is divided into quarters (Fig. 1) - if any of the dimensions *M* and/or *N* does not allow for an exact division (odd number), then the first halves (left and/or right) are taken as a half of the whole increased with 1 (it will always be even) and the rest with 1 less (e.g. if *M*=7, the quartile lengths to the left of the whole array are assumed to be 4 (half the nearest even number) and the lengths of the quarters to the right are taken 3).

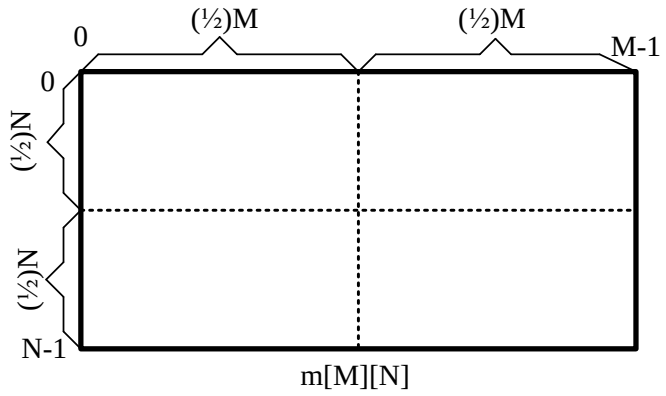

Fig. 1. Two-dimensional array split in quarters

For each quarter the element with maximum and minimum value is found and so is the difference in absolute value (positive number) and it is compared with predefined threshold - thresh1. If thresh1 is greater, all the values of the quarter elements are replaced with the average of their old values (again rounded to an integer, but the averaging itself is not an integer but a double). If thresh1 is less, the current quarter is divided again into quarters (if again one of its two or both sizes is an odd number - the approach from above is applied) and recheck for each new one if the absolute value of the difference between the maximum and minimum is greater than or less than thresh1 and the above steps apply. The splitting of quarters continues until a single element (1 value of the entire matrix) is reached if necessary.

Here we have rectangular blocks of different sizes (generally from *(M/2)x(N/2)* to 1x1 elements) each containing elements of the same values (the average of their original values). For each of these blocks, check whether their width or height (both sizes) exceeds thresh2 - if only one of these sizes exceeds it - all values of the given block are equalized to zero.

The array with dimensions *MxN* elements already has blocks containing zeros and positive values (the same within each block separately). For each block consisting of non-zero elements, all adjacent blocks are checked, which are also non-zero. The check is done by value - if the difference per module between the value of the current block (it is one for the whole) and the currently comparing neighboring (also one) is less than thresh3 - the two blocks are assumed to be "connected" and both are replaced with the weighted average of their two previous values - the number of pixels in one area of its brightness, collected by the number of pixels of the other in its brightness and divided by the total number of pixels for the two areas.

So far we have obtained a two-dimensional array *MxN* elements in which we have a number of areas (macro blocks) whose values are constant (equipotential domains). For each area obtained, the width and height (in number of elements) of the smallest rectangle surrounding it (Fig 2). If either the width or height of an area is less than thresh4, the area values are equal to zero.

Now we have a two-dimensional array MxN element containing equipotential domains, each with a value between 0 and 255 (including boundaries) - it is necessary one to be able to "return" it as a source parameter (e.g. by a pointer to the memory area) and we can also save it from this function in a binary file on the disk. All other intermediate areas (excluding the last received array) from the RAM for intermediate operations should be cleared.

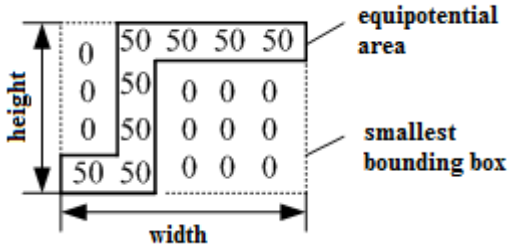

Fig. 2. Equipotential area dimensions estimation

B.Source design

The diagram of the classes SplitAndMergeSegmenter and ImageSegment are given in Fig. 3. The splitting is implemented recursively within the ImageSegment class where all obtained segments with its internal values and position within the matrix of the whole image. Division is applied until block size reaches 1x1 pixels.

The method Merge use a list of all registered segments. According to the algorithm description from Section II.A weighted average intensity is associated to the resulting bigger regions after merging.

The essential parts from the definitions of the classes are listed below:

```
public class SplitAndMergeSegmenter
  {
    private const double RedMultiplier = 0.3;
    private const double GreenMultiplier = 0.59;
    private const double BlueMultiplier = 0.11;
    private readonly ICollection<ImageSegment>
segments;
    private readonly Bitmap image;
    private readonly int thresh1;
    private readonly int thresh2;
    private readonly int thresh3;
    private readonly int thresh4;
    public SplitAndMergeSegmenter(Bitmap image, int
thresh1, int thresh2, int thresh3, int thresh4)
    {
      this.image = image;
      this.segments = new List<ImageSegment>();
      this.thresh1 = thresh1;
```

```
            this.thresh2 = thresh2;
            this.thresh3 = thresh3;
            this.thresh4 = thresh4;
        }
    private void Merge(IEnumerable<ImageSegment> imageSegments)
        {
            foreach (var segment in imageSegments)
            {
                var block = imageSegments.Where(x => CanMerge(x, segment));
                if (block == null)
                {
                    continue;
                }
                foreach (var item in block)
                {
                    CombineSegments(segment, item);
                }
            }
    …
    }

    public class ImageSegment
      {
        public ImageSegment(int[,] pixels, int topLeftX, int topLeftY, int bottomLeftX, int bottomLeftY, int topRightX, int topRightY, int bottomRightX, int bottomRightY)
        {
            this.Pixels = pixels;
            this.topLeftIndexX = topLeftX;
            this.topLeftIndexY = topLeftY;
            this.topRightIndexX = topRightX;
            this.topRightIndexY = topRightY;
            this.bottomLeftIndexX = bottomLeftX;
            this.bottomLeftIndexY = bottomLeftY;
            this.bottomRightIndexX = bottomRightX;
            this.bottomRightIndexY = bottomRightY;
            this.segments = new List<ImageSegment>();
        }
        public int[,] Pixels { get; set; }
        public int topLeftIndexX { get; set; }
        public int bottomLeftIndexX { get; set; }
        public int topRightIndexX { get; set; }
        public int bottomRightIndexX { get; set; }
        public int topLeftIndexY { get; set; }
        public int bottomLeftIndexY { get; set; }
        public int topRightIndexY { get; set; }
        public int bottomRightIndexY { get; set; }
        public bool isZeroed { get; set; }
        public bool isCombined { get; set; }
    public ICollection<ImageSegment> segments { get; set;}
        public void SetAvarageValue(int number)
        { for (int i = 0; i < this.Pixels.GetLength(0); i++)
            { for (int j = 0; j < this.Pixels.GetLength(1); j++)
                { this.Pixels[i, j] = number; } } } }
```

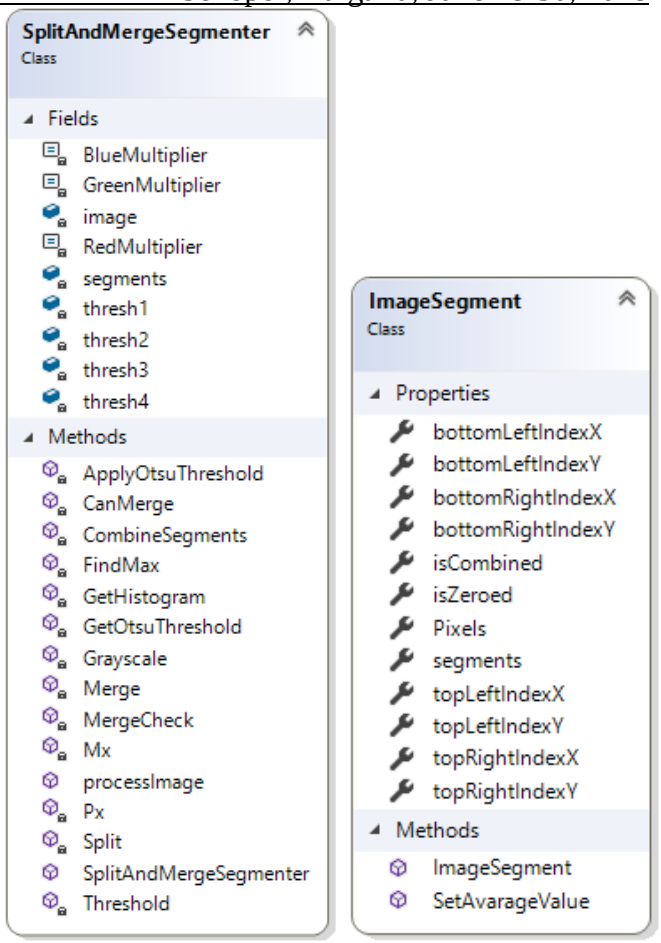

Fig. 3. Diagrams of the main application classes

III. Experimental Results

The experimental test set includes 100 color images in RGB format at 24 bpp with 320x240 pixels in dimension. They are processes on PC compatible machine with Intel Core 2 Quad Q8300 CPU running at 2.5 GHz with 4 GB of RAM under Windows 10 OS. Empirically obtained thresholds based on font sizes present in the test images are set to be: thresh1 = 100, thresh2 = 5, thresh3 = 70, thresh4 = 1. Segmentation accuracy and execution time of the current implementation are presented in Table I compared to those of the implementation of Wu et al. [13].

Table I
Segmentation Accuracy and Time

Para-meters / Method	Correctly segmented symbols, %	Fragmented symbols, %	Missed symbols, %	Average execution time, sec
[13]	79.6	7.6	12.8	-
Proposed	81.3	10.5	18.2	0.5

A single image with its resulting segmented appearance can be observed in Fig. 4. Due to the low resolution with high compression ratio in MPEG-1 format (being an excerpt from SDTV footage with subsequent downsampling) considerable amount of false detections are attached to the symbols.

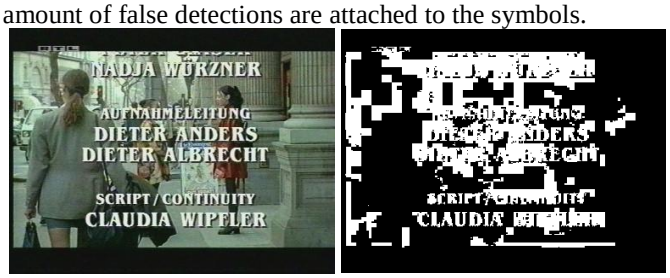

Fig. 4. Low resolution TV footage with lots of false detections

Some sample segmented images outside the main test set are given in Fig. 5 and Fig. 6 where different type of errors in segmentation appear. All the characters from the image in Fig. 5 are well detected but their edges are jagged because of the small scale to which splitting blocks emerge at the last segmentation step.

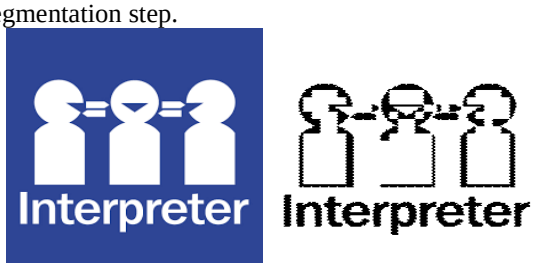

Fig. 5. Test results over artificially generated image

In the image from Fig. 6 there is large number of missing letters from the lower part of the text field. The reason for that is the small font size. Here, additional deteriorations exist as a sequence of the more complex forms of the characters.

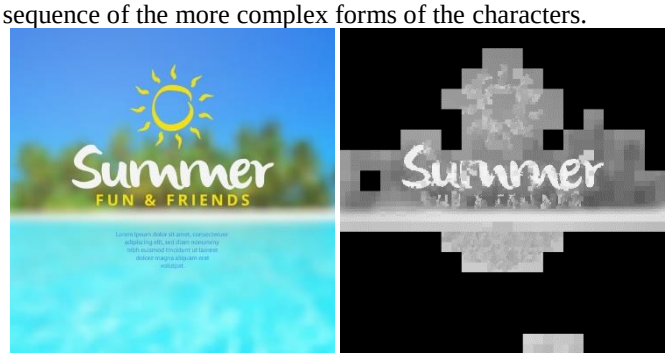

Fig. 6. Processing results for text with various font types and sizes

IV. DISCUSSION

The main influencing factors are textures from the complex backgrounds in the scenes. This hampers the process of accurate merging for the blocks that belong to the print symbols. Another factor is the thickness of the symbols, which, if below a certain threshold, impedes the removal of part of the blocks after segmentation. When area filtration is carried out, sometimes it leads to erosion of the symbols. Further, accurate segmentation is limited by overlapping translucent text on a complex background. There solid color fill of the individual symbols acquire color component variations, which also leads to disruptions in separate parts.

V. CONCLUSION

The proposed implementation of the split and merge algorithm in C# offers performance which satisfies potential users from practical point of view using contemporary Desktop computers. Its object-oriented class definitions may be easily extended. It is a practical ground for future modifications of the algorithm in order to cope with fonts more complicated in form and imposed in smaller sizes. Object textures from the background resembling present characters is the other direction to which further enhancements should be sought.

REFERENCES

[1] D. Chen, K. Shearer, H. Bourlard, “Text Enhancement with Asymmetric Filter for Video OCR”, In Proc. of the 11th Int. Conf. on Image Analysis and Processing, pp. 192–198, Sept. 2001.

[2] C. Garcia, X. Apostolidis, “Text Detection and Segmentation in Complex Color Images”, In Proc. of the Int. Conf. on Acoustics, Speech and Signal Processing, pp. 2326–2329, 2000.

[3] O. Hori, “A Video Text Extraction Method for Character Recogntion”, In Proc. of the Int. Conf. on Document Analysis and Recognition, pp. 25–28, Sept. 1999.

[4] H. Kamada, K. Fujimoto, “High-speed, High-accuracy Binrization Method for Recognizing Text in Images of Low Spatial Resolutions”, In Proc. of the Int. Conf. on Document Analysis and Recognition, pp. 139–142, Sept. 1999.

[5] R. Laprade, M. Doherty, “Split-and-merge Segmentation using an F Test Criterion”, SPIE image understanding and man-machine interface, pp. 74–79, 1987.

[6] H. Li, D. Doermann, “Text Enhancement in Digital Video using Multiple Frame Integration”, In Proc. of the ACM Multimedia, vol. 1, pp. 385–395, Orlando, Florida, USA, 1999.

[7] R. Lienhart, “Automatic Text Recognition in Digital Videos”, In Proc. SPIE, Image and Video Processing IV, pp. 2666–2675, Jan. 1996.

[8] R. Lienhart, A.Wernicke, “Localizing and Segmenting Text in Images and Videos”, IEEE Trans. on Circuits and Systems for Video Technology, vol. 12, no. 4, pp. 256–268, 2002.

[9] T. Pavlidis, Y.T. Liow, “Integrating Region Growing and Edge Detection”, IEEE Trans. on Pattern Analysis and Machine Intelligence, pp. 225–233, 1990.

[10] T. Sato, T. Kanade, E. K. Hughes, M. A. Smith, “Video OCR for Digital News Archives”, In Proc. of the IEEE Workshop on Content Based Access of Image and Video Databases, pp. 52–60, Bombay, Jan. 1998.

[11] M. A. Smith, T. Kanade, “Video Skimming for Quick Browsing based on Audio and Image Characterization”, Technical Report CMU-CS-95-186, Carnegie Mellon University, July 1995.

[12] K. Sobottka, H. Bunke, H. Kronenberg, “Identification of Text on Colored Book and Journal Covers”, In Proc. of the Int. Conf. on Document Analysis and Recognition, pp. 57–63, 1999.

[13] V.Wu, R. Manmatha, E. M. Riseman, “Finding text in images”, In Proc. of the ACM Int. Conf. Digital Libraries, pp. 23–26, 1997.

[14] Y. Zhong, K. Karu, A. K. Jain, “Locating Text in Complex Color Images”, Pattern Recognition, vol. 10, no. 28, pp. 1523–1536, 1995.