SCENE TEXT SEGMENTATION
BASED ON THRESHOLDING
Final Thesis Research

Alejandro Perez Sanmartín

Mentor: Matko Šarić
18-07-2015
Abstract

This research deals with the problem of text segmentation in scene images. Introduction deals with the information contained in an image and the different properties that will be useful for image segmentation. After that, the process of extraction of textual information is explained step by step. Furthermore, the problem of scene text segmentation is described more precisely and an overview of more popular existing methods is given.

Text segmentation method is created and implemented using C++ programming language with OpenCV library. Finally, algorithm is evaluated with images from ICDAR 2013 test dataset.
# Index

Abstract......................................................................................................................... 1
Index............................................................................................................................... 3
Glossary.......................................................................................................................... 4

1. Preface ...................................................................................................................... 5
   1.1. Introduction............................................................................................................. 5
   1.2. Motivation.............................................................................................................. 5
   1.3. Objective............................................................................................................... 5

2. Report ....................................................................................................................... 6
   2.1. Content of an image ............................................................................................... 6
   2.2. Properties of text in images .................................................................................. 6
   2.3. Process of extraction of text in image ...................................................................... 8
   2.4. Examples of general methods for segmentation of text ........................................ 13
   2.5. Implement text segmentation method using C++ and OpenCV library .............. 16
   2.6. Evaluation of results ............................................................................................. 25

3. Future Work ............................................................................................................. 29

4. Conclusions ............................................................................................................. 30

5. Acknowledgments .................................................................................................... 31

6. References ............................................................................................................... 32

Annex 1: Algorithm with Adaptative threshold.......................................................... 34
Annex 2: Algorithm with Otsu threshold ...................................................................... 37
Annex 3: Algorithm Check results ............................................................................... 39
Glossary

- TFG: Treball Final de Grau (Bacchelor’s final thesis).
- OCR: Optical character recognition.
- RGB: color model meaning Red, Green and Blue.
- HIS: color representation coordinate system meaning Hue, Intensity and Saturation.
- TIE: Text image extraction.
- MSER: maximally stable extremal regions.
- IHLS: color space meaning Intensity, Hue, Luminance and Saturation.
- OpenCV: Open Source Computer Vision.
- Jpeg: format of image meaning Joint Photographic Experts Group.
- MPEG: format of image meaning Moving Picture Experts Group.
- 2D: Two dimensions.
- 3D: Three dimensions.
1. Preface

1.1. Introduction

Images and video contain a lot of information that could be very useful for different applications. Image processing is used in diverse fields as medical tests, fingerprint reader, face recognition, text recognition, etc. For this reason, it is very interesting to study the process of obtaining this information. In this research, focus will be on one specific case: text segmentation. Firstly, it will be an introduction to information that an image contain. After that, different steps for extraction information will be explained. Further, an overview of existing methods and an explanation of the most popular ones will be presented. Finally, an algorithm is implemented with C++ language programming in Visual Studio and results are evaluated with images from ICDAR 2013 dataset.

1.2. Motivation

The personal motivation of this research is to complete my studies in Industrial Technologies researching in some specific field on Engineering, in this case in image processing. My purpose at the moment to accept this challenge is to learn and to deal with a specific problem that an engineer has to solve.

Moreover, creation of my own algorithm is an extra motivation. Nowadays, all technology is based in computer science and it has a lot of importance in a lot of companies. With this research, I have the possibility to improve my skills in C/C++ programming for algorithm implementation and in Matlab for results evaluation. It is a good opportunity that I couldn’t let pass thinking of my future and my entrance in the world of employment.

1.3 Objective

The principal objective of this research is to find a solution of a real engineering problem, in this case, text extraction and segmentation in scene images is. For it, a general algorithm is created that should work in different types of images.
2. Report

2.1 Content of an image

It is very important to study the content of an image in order to find the best form for image processing. The content of an image could be classified in perceptual content and semantic content. Perceptual content refers to attributes that we can feel with senses, such as color, intensity, shape, texture and their changes. Semantic content means objects and events like faces, text, vehicles and human action. Textual information often provides important clues for understanding the high-level semantics of multimedia content. Moreover, it is easy to extract text comparing to other semantic content, it has a lot of interesting information about the image and is used in so many applications. For these reasons, this research is focused on text extraction.

2.2 Properties of text in images

Text in images has 5 main characteristics:

1. **Geometry**: geometry of text could change a lot in an image. We can differentiate 3 properties:
   - **Size**: every text on every image has different size, but it is common that one text in one image has regularity in his size.
   - **Alignment**: it is usual that text appears horizontally (like in subtitles) but we can find it in different positions like vertical, diagonal or curve. Moreover, it could appear with perspective distortions if text is not in the image plane (3D).
   - **Inter-character distance**: characters in a line of text usually have uniform distance between them.

For example, in Image 1, there is text with 2 different sizes: “Kraft” is the smaller one and “PHILADELPHIA” the bigger one. Each one has uniform inter-character distance and text appears horizontally.
2. **Color**: text usually has the same or similar colors in single images. It is very important for text detection and its differentiation from the background. When text has only one color, it is called monochrome. Sometimes in videos or some documents, text has more than 2 colors and it is called polychrome. Image 2 is an example for polychrome document. It has text of different colors and sizes.

![Image 2: example polychrome](image2.jpg)

3. **Motion**: The same characters usually exist in consecutive frames in a video with or without movement. If it has not movement, it is called static. There are different variations of movement: linear movement, 2D rigid constrained movement, 3D rigid constrained movement and free movement.

4. **Edge**: is it easy to recognize the strong edges at the boundaries of text and differentiate it from the background. Next image is a good example of strong edges:

![Image 3: example of strong edges](image3.jpg)

5. **Compression**: Many digital images are recorded, transferred, and processed into a compressed format. Thus, a faster text image extraction system can be achieved if one can extract text without decompression. There are two types: uncompressed image JPEG and MPEG-compressed image.
2.3 Process of extraction of text in images

There are many special cases of techniques for text extraction in image based on detection of the features mentioned above and variations or differences with background. All these processes have the following in common:

- Text detection: determination of text presence in a given frame.
- Text localization: determination of text location and generate bounding boxes.
- Text tracking: reduce processing time.
- Text extraction and enhancement: text components are segmented from the background.
- Recognition. (OCR).

In diagram 1, the process of text extraction is shown:

![Diagram 1: Steps in a process of extraction of text]

2.3.1 Text detection

This is the first step, where it is determined if there is text or not in the image. There are some researches that try to solve the problem of detection of text. H.K. Kim [1] select a frame from shots detected by a scene-change detection method as a candidate containing text. M.A. Smith, and T. Kanade [2] defined a scene change based on the difference between two consecutive frames and then used this scene-change information for text detection. U. Gargi [3] performed text detection using the assumption that the number of intra coded blocks in P- and B- frames of an MPEG compressed video increases, when a text caption appears. Lim et al. Made [4] make a simple assumption that text usually has higher intensity than the background.
2.3.2 Text localization

In this step, the purpose is to detect the location of the text and generate boundary boxes. There are two main types of text localization methods: region-based methods and texture-based methods.

- **Region-based:** use the properties of the color or gray scale in a text region or their differences with the corresponding properties of the background. There are two types:
  - Connected component-based: these methods first discard the majority of background pixels using low-level filters, and then construct component candidates from remaining pixels using a set of heuristic properties, for example, consistency of stroke width and color homogeneity. Connected component analysis is further applied for filtering out outliers.
  - Edge-based methods: focus on the high contrast between the text and the background.

- **Texture-based:** Texture-based methods use the observation that text in images have distinct textural properties that distinguish them from the background. These methods scan the image at different scales using sliding windows, and classify text and non-text regions based on extracted window descriptors.

In the next image, there is an example for text localization. The boundary boxes are selecting the part of the image where text is.
2.3.3 Text extraction and enhancement

Text extraction is the stage where the text components are segmented from the background. Segmentation of images means that image is divided in different regions (edges) that have some homogeneous characteristic. The simplest property that pixels in a region can share is intensity. So, a natural way to segment such regions is through thresholding, obtaining the separation of light and dark regions. Threshold creates binary images from grey-level ones by turning all pixels below some threshold to zero and all pixels about that threshold to one.

Text extraction methods are classified as thresholding-based and grouping-based methods.

2.3.3.1 Thresholding-based


Otsu’s threshold method involves iterating through all the possible threshold values and calculating a measure of spread for the pixel levels that make them fall in foreground or background. The aim is to find the threshold value where the sum of foreground and background spreads is at its minimum.

Niblack’s [7] algorithm is a local threshold method based on the calculation of the local mean and local standard deviation. The idea of Niblack’s method is to vary the threshold over the image, computed in a small neighborhood of each pixel. The threshold is decided by the formula:

\[
T_{\text{Niblack}}(x, y) = m(x, y) + k \cdot s(x, y)
\]

Equation 1: Niblack’s thresholding

Where \(m(x, y)\) is the local mean value and \(s(x, y)\) is the value of standard deviation. The size of the neighborhood should be small enough to preserve local details, but at the same time large enough to suppress noise. The value of \(k\) is used to adjust how much of the total print object boundary is taken as a part of the given object.

Sauvola’s algorithm [8] claims to improve Niblack’s method by computing the threshold using the dynamic range of image gray-value standard deviation, \(R\):

\[
T_{\text{Sauvola}} = m(x,y) \cdot (1 - k \cdot (1 - s(x,y)/R))
\]

Equation 2: Sauvola’s thresholding
where R is the maximum value of the standard deviation (R = 128 for a greyscale document), and k is a parameter which takes positive values in the range [0.2, 0.5]. The local mean m(x,y) and standard deviation s(x,y) adapt the value of the threshold according to the contrast in the local neighborhood of the pixel.

This method outperforms Niblack’s algorithm in images where the text pixels have near 0 gray-value and the background pixels have near 255 gray-values. However in images where the gray values of text and non-text pixels are close to each other, the results degrade significantly.

In general, methods that use a global threshold typically work well when the text occupies a large part of the picture and is well contrasted from the background. On the other hand, local binarization techniques can handle better with variations of illumination and text color.

### 2.3.3.2 Grouping-based

Grouping-based includes region-based, learning-based and clustering-based. In grouping-based methods, groups of text pixels are formed according to certain criteria.

#### a) Region-based

Region growing and split-and-merge algorithms, representing bottom-up and top down strategies, belong to region based approaches. For example, Karatzas and Antonacopoulos[9] segment text in web images using split-and-merge technique for chromatic and achromatic image regions. These regions are recursively split using intensity histogram for achromatic pixels and hue histogram for chromatic pixels. R. Lienhart, A. Wernicke [10], a 4-neighborhood region growing algorithm with Euclidean distance in RGB color space is used for background separation. Dujmic et al. [11] proposed scene text extraction method employing region growing algorithm in HSI color space with modified cylindrical distance.

#### b) Learning-based

Learning-based methods employ classifiers like multi-layer perceptrons and support vector machines. This approach is more often used in text localization where classifier estimates text probability based on region feature vector [12].
c) Clustering-based

Clustering-based methods rely on hypothesis that text and background pixels tend to form groups in an appropriate color space. The most popular technique is k-means [13] although other clustering M. Šarić, H. Dujmić, M. Russo approaches like GMM and spectral clustering also attract researcher’s attention. Garcia and Apostolidis [14] segment text using 4-means algorithm in HSV color space. Mancas-Thillou and Gosselin [15] proposed text extraction method that uses clustering in RGB color space with two metrics.

Extracted text image has to be converted to a binary image and enhanced before it is fed into a OCR engine. Enhancement of the extracted text components is usually required because the text region usually has low-resolution and noise. Text enhancement techniques can be divided into two categories: single frame-based or multiple frame-based.

2.3.4 Recognition (OCR)

Optical character recognition (OCR) is the mechanical or electronic conversion of images of typewritten or printed text into machine-encoded text. OCR systems have been available for a number of years and the current commercial systems can produce an extremely high recognition rate for machine-printed documents on a simple background. However, it is not easy to use commercial OCR software for recognizing text extracted from images or video frames to handle large amount of noise and distortion in TIE applications.
2.4 Examples of general methods for segmentation of text

There are a lot of different general methods proposed for segment the text from the background. Here, it is explained some popular methods:

a) **Multi-level Maximally Stable Extremal Regions (MSERs)**

Multi-level MSER [19] technology improve MSER [16-18] that has been adopted in many scene text detection and recognition systems. It identifies the best-quality text candidates from a set of stable regions that are extracted from different color channel images.

In order to identify the best-quality text candidates, a segmentation score is defined which exploits four measures to evaluate the text probability of each stable region including:

1) **Stroke width [20]**: text usually has nearly constant stroke width while background regions usually have a varying stroke width. Stroke width could be defined as the number of pixels. Thus it is an effective measure to eliminate the noisy backgrounds.

2) **Boundary curvature [21]**: measures the smoothness of the stable region boundary.

3) **Character confidence**: measures the likelihood of a stable region being text based on a pre-trained support vector classifier. It could be used to choose better text candidates and eliminate false alarms.

4) **Color constancy**: measures the global color consistency of each selected text candidate.

Finally, the MSERs with the best segmentation score from each channel are combined to form the final segmentation.

b) **Image binarization for end-to-end text understanding in natural images [22]**

This method proposes a new binarization algorithm. It is shown with an example. Firstly, we have an input image:
The method consists of the following steps:

1) **Local binarization**: producing seed pixels using Niblack binarization. In particular, we perform local binarization with a rather small window size, since using large window size inside Niblack usually causes small letters to merge and we want to avoid this effect. In the next image, we can appreciate local binarization for dark text on light background. The candidate text regions are shown in blue.

![Image 6: Local binarization of Image 5](image)

2) **Seed pixel strength estimation**: the normalized value of Laplacian of image intensity is computed at each pixel. The result of the Laplacian operator \([23-24]\) tends to have large absolute values near edges, where the local binarization with small window provides correct labels. Within the interior part of the letters the values of the Laplacian are usually close to zero. In image 7, it is appreciated the seeds resulting from incorporating local binarization and the Laplacian of the image intensity.

![Image 7: Local binarization and Laplacian of image 5](image)

3) **Global binarization**: global optimization which accounts for pixel similarity for correcting errors of initial labeling. The image 8 shows the binarization after global optimization for dark text on light background. The candidate text regions are shown in blue.

![Image 8: Binarization after global optimization](image)
c) **Scene Text Extraction in IHLS Color Space Using Support Vector Machine**

This method uses chromatic and lightness component for generation of extraction hypotheses. Moreover, it incorporates SVM (support vector machine) based text detection stage as tool for hypotheses verification.

The choice of chromatic and lightness components is based on their complementarity with respect to image degradations like shadows and highlights. Another novelty is the usage of IHLS color space for text extraction task which is motivated by saturation definition that eliminates instability of this component at low lightness values.

The steps of proposed extraction method are shown in diagram 2:

1. **Input image**

2. **Generation of text extraction hypothesis**

3. **Hypothesis verification using SVM**

4. **OCR on chosen hypothesis**

![Diagram 2: Steps extraction method](image)

The first step is generation of two text extraction hypotheses: one based on the chromatic distance, that is, chromatic component, and the second based on the lightness component. In this step, Otsu threshold is performed on both components in order to generate two binary images that represent candidates for final extraction results. After this step the hypothesis verification is performed using SVM classifier. For each one, it is estimated the degree of "character–similarity". Segmentation with higher average "character–similarity" value is chosen and sent to recognition stage done by OCR software.
2.5 Implementation of text segmentation method using C++ and OpenCV library

The program chosen for the implementation of text segmentation is Visual Studio 2013 and programming language is C++. Algorithm uses OpenCV library that is available for free access in www.opencv.org.

Algorithm implementation is written in Annex 1. In this chapter, it is explained step by step. It follows the next scheme:

1. Read the image
2. Transform image to grayscale.
3. Blur image before preprocessing.
4. Threshold.
5. Filter after threshold.
6. Find contours.
7. Filter contours for find letters.
8. Draw letters.

First of all, vectors are used for read and save on folders all images of icdar 2013 dataset. If any problem occurs opening folders and reading the images, program will notice it. The main function of the algorithm is process_image, where input images are processed.

Firstly, image is read with OpenCV function imread and at the same time is transformed to grayscale. After that, it is converted to CV_8U. This step is necessary for correct image processing.

Now image is filtered with a Gaussian blur (also known as Gaussian smoothing) to reduce image noise. The Gaussian blur is a type of image-blurring filter that uses a Gaussian function for calculating the transformation to apply to each pixel in the image. The equation of a Gaussian function in two dimensions is:

$$ G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} $$

Equation 3: Gaussian blur
Where:

- \( x \): is the distance from the origin in the horizontal axis.
- \( y \): is the distance from the origin in the vertical axis.
- \( \sigma \): is the standard deviation of the Gaussian distribution.

This formula produces a surface whose contours are concentric circles with a Gaussian distribution from the center point. Values from this distribution are used to build a convolution matrix which is applied to the original image. Each pixel's new value is set to a weighted average of that pixel's neighborhood. The original pixel's value receives the heaviest weight and neighboring pixels receive smaller weights as their distance to the original pixel increases. The result is a blur that preserves boundaries and edges better than others.

After image blurring, it is applied one morphological transformation for continue filtering the image: **Opening**. Morphological operations apply a structuring element to an input image and generate an output image. In this case, Opening consists in the erosion of the image followed by a process of dilation. It is useful for removing small objects. The code is this:

```cpp
cv::Mat const structure_elem = cv::getStructuringElement(
    cv::MORPH_RECT, cv::Size(5, 5));
cv::Mat open_result;
cv::morphologyEx(blurred, open_result, cv::MORPH_OPEN, structure_elem);
```

As it is shown in image, it is used rectangles as structure of element with window Size (5,5). Next image, shows the effects of this function for small noise removal.
In this example, it is appreciate a great difference between using this function or not for different values of threshold. This function greatly improves result. Nevertheless, sometimes it creates some problems when the borders of letters are not well defined and it deletes some pixels of the text as it is shown in next example:
Next step is image thresholding. There are different functions and algorithms that make this threshold. In this research, 2 of them are used:

a) **Otsu-threshold**

Nobuyuki Otsu performed this clustering-based method for image threshold. As it is explained before, the algorithm assumes that the image contains two classes of pixels following bi-modal histogram: foreground pixels and background pixels. In OpenCV, function automatically calculates a threshold value from image histogram for a bimodal image. In next images, it is shown the same examples as before with Otsu segmentation:

![Image 17: Input ICDAR 2013 image 113](image17.png) ![Image 18: Otsu threshold image 113](image18.png)

Nevertheless, in some images does not detect the letters as it is shown in image 20:

![Image 19: Input ICDAR 2013 image 106](image19.png) ![Image 20: Otsu threshold image 106](image20.png)
b) **AdaptiveThreshold**

Function **adaptiveThreshold** transforms a gray scale image to a binary image according to the next formula:

\[
dst(x, y) = \begin{cases} 
\text{maxValue} & \text{if } \text{scr}(x, y) > T(x, y) \\
0 & \text{otherwise}
\end{cases}
\]

\text{Equation 3: Adaptive thresholding}

Where \(T(x, y)\) is a mean of the blockSize x blockSize neighborhood of \((x, y)\) minus \(C\).

\[
T(x, y) = m(x, y) - C
\]

\text{Equation 4: Adaptive thresholding}

\(\text{maxValue}\) is implemented as 255. It means that pixels are segmented as 0 or 255 depend if intensity is more or less than \(T(x, y)\). In image 22, it is shown segmentation result for the same example with blockSize = 51 and \(C = 20\).

\[
\text{Equation 4: Adaptive thresholding}
\]

It is shown that the letters are segmented correctly, but there is a lot of noise. This problem should be solved with filters in next steps. In next example, Adaptive segment better the letters than Otsu:
The biggest problem founded with Adaptive threshold is to choose the values of blockSize and $C$ to properly segment the images. Next table shows the segmentation of the next image using different values for blockSize and constant $C$.

<table>
<thead>
<tr>
<th>BlockSize</th>
<th>C=20</th>
<th>C=30</th>
<th>C=40</th>
<th>C=50</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td><img src="image1.png" alt="Segmentation1" /></td>
<td><img src="image2.png" alt="Segmentation2" /></td>
<td><img src="image3.png" alt="Segmentation3" /></td>
<td><img src="image4.png" alt="Segmentation4" /></td>
</tr>
<tr>
<td>21</td>
<td><img src="image1.png" alt="Segmentation1" /></td>
<td><img src="image2.png" alt="Segmentation2" /></td>
<td><img src="image3.png" alt="Segmentation3" /></td>
<td><img src="image4.png" alt="Segmentation4" /></td>
</tr>
<tr>
<td>31</td>
<td><img src="image1.png" alt="Segmentation1" /></td>
<td><img src="image2.png" alt="Segmentation2" /></td>
<td><img src="image3.png" alt="Segmentation3" /></td>
<td><img src="image4.png" alt="Segmentation4" /></td>
</tr>
<tr>
<td>41</td>
<td><img src="image1.png" alt="Segmentation1" /></td>
<td><img src="image2.png" alt="Segmentation2" /></td>
<td><img src="image3.png" alt="Segmentation3" /></td>
<td><img src="image4.png" alt="Segmentation4" /></td>
</tr>
<tr>
<td>51</td>
<td><img src="image1.png" alt="Segmentation1" /></td>
<td><img src="image2.png" alt="Segmentation2" /></td>
<td><img src="image3.png" alt="Segmentation3" /></td>
<td><img src="image4.png" alt="Segmentation4" /></td>
</tr>
</tbody>
</table>

*Table 2: Adaptive threshold image 163 for different values of BlockSize and C*
In order to make a general program for all images, the values of blockSize = 51 and C = 20 are selected. They are chosen because more pixels are detected and drawn and we are sure that all letters are segmented correctly. The disadvantage as it is commented before, is all the noise that appears and we will need to remove.

After checking the different methods of threshold, algorithm will be implemented using Otsu-thresholding and AdaptiveThreshold in order to compare them.

After threshold processing, it is applied another filter based in structure element and the opposite of opening: **Closing**. It is obtained by the dilation of an image followed by an erosion and is useful for remove small holes. Code is shown in next image, and as with opening, it is used rectangles as structuring elements with Size (5,5).

```cpp
cv::Mat const structure_elem2 = cv::getStructuringElement(
    cv::MORPH_RECT, cv::Size(5, 5));

cv::Mat close_result;

cv::morphologyEx(image_threshold, close_result,
    cv::MORPH_CLOSE, structure_elem2);
```

**FindContours** detect the contours of a binary image. Code is:

```cpp
vector<vector<Point>> contours;
vector<Vec4i> hierarchy;

cv::findContours(close_result, contours, hierarchy, CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE, Point(0, 0));
```

CV_RETR_CCOMP is the type of retrieval mode used. It retrieves all of the contours and organizes them into a two-level hierarchy. At the top level, there are external boundaries of the components. At the second level, there are boundaries of the holes.

Now function **boundingRect** is used to calculate the minimal up-right bounding rectangle for each contour. This function gives the possibility to know the height and width of each rectangle in each contour and with this measure, apply the next equations and properties to try to draw only the letters. Properties selected are maximum and minimum Area, aspect ratio, occupy rate and compactness.

Area of each contour is calculated with OpenCV function **contourArea**. Minimum area and maximum area are defined in order to delete noise that appears.

Values selected are:
- Minimum Area: 75
- Maximum Area: 25000

Occupy rate is the relation between the Area of the contour and its height and width. It is calculated with next formula:

\[
\text{Occupy rate} = \frac{\text{area}}{\text{height} \times \text{width}}
\]

Equation 5: Occupy rate

Height and width are calculated with `boundingRect` for each contour and Area is calculated with function `contourArea`.

Aspect ratio is the relation between height and width. It is calculated with next formula:

\[
\text{Aspect ratio} = \frac{\max(\text{width, height})}{\min(\text{width, height})}
\]

Equation 6: Aspect ratio

Compactness is the relation between the Area and the perimeter:

\[
\text{Compactness} = \frac{\text{area}}{\text{perimeter} \times \text{perimeter}}
\]

Equation 7: Compactness

Perimeter is calculated with function OpenCV function `ArcLength` that returns a contour perimeter.

All this parameters are used for filtering our images. Biggest problem is to choose maximum and minimum values for each parameter because is an algorithm for all the images and each image is different and letters have different sizes and forms. The values of filters that are perfect for eliminate all noise in one image, at the same time are eliminating letters in other image. Chosen values that in general for all images better work are written in next code:

```python
if (contourArea(contours[idx]) > CONTOUR_AREA) & (contourArea(contours[idx]) <= CONTOUR_AREA_MAX))
{
    if ((occupyrate >= 0.03) & (occupyrate <= 0.95))
    {
        if (aspectratio <= 6)
        {
            if (compactness > 0.003 & compactness <= 0.95)
```
Finally, contours that pass all the filters are drawn by function `drawContours`:

```c
Scalar color(rand() & 255, rand() & 255, rand() & 255);
drawContours(drawing, contours, idx, color, CV_FILLED, 8, hierarchy);
```

Algorithm iterates in contours until idx is >= 0. Hierarchy is used to draw the contours because result is better because the contours of the holes are detected. It is possible to see the whole algorithm in Annex 1.

Once contours are drawn, a new problem is detected. In some images, text is segmented in the wrong way and the contours are drawn in the opposite form. Outside of letters is drawn, not the inside as we want. It is easy to understand with next example:

![Image 26: Input ICDAR image 181](image1)
![Image 27: Result segmentation image 181](image2)

For try to solve this problem, it has been implemented some code for try to detect if the image is darker or lighter because usually the letters are the opposite as the background. It is tried checking if the pixels are lighter or darker in the first rows and in the last ones and calculate the median between them. It didn’t work so well and finally it is decided to count all black and white pixels and if the difference between them is higher than some value, invert the image. It works for some of the images, but not for all of them and for evaluate the results we will test the algorithm twice, in normal way and inverting the image.
2.6 Evaluation of results

2.6.1 Examples of images where text is correctly segmented

In next images, it is shown some of the results where algorithm works in correct way segmenting all the text in the image:
3.6.2 Problems in images where text is not correctly segmented

In this chapter, some mistakes are commented.

- **Noise**: in some images algorithm doesn’t filter enough the noise of the background. One example of this is image 35:

![Image 34: Input ICDAR Image117](image1) ![Image 35: Result segmentation image 117](image2)

Text is well detected and segmented, but it is some noise that algorithm can not eliminate.

- **Letters wrong segmented**: in some images, some letters doesn’t appear. One example of this, is image 37, letters “fessi” are bad segmented and not appear in the result.

![Image 36: Input ICDAR Image117](image3) ![Image 37: Result segmentation image 117](image4)

Solving these problems for each picture is not difficult, just select better values for filter the letters. Nevertheless, for the algorithm work well for the 229 pictures tested is really hard task. In next chapter, some improvements are implemented in future work.
Last problem is complicated to solve for each image. There are some images that they have some letters darker and some letters lighter. Then when algorithm works, some of them are segmented in the correct way and some of them are drawn in the opposite way. In image 39, it is shown one example:

“Beefeater” is drawn in the correct way, but “The Albert” is in the opposite. This problem is difficult to solve, algorithm has to be changed with improvements in the moment of letter detection, without depending of the color of the letter.

2.6.3 Results

In Annex 3, it is shown functions used in Matlab for evaluate the results. Images obtained from the algorithm are compared with results provided for ICDAR 2013 data test. It is chosen two parameters for evaluate the results, precision and recall:

- **Recall**: is the ratio of the number of relevant pixels retrieved to the total number of relevant pixels in the database:

  \[
  Recall = \frac{(\text{Relevant pixels retrieved})}{((\text{Relevant pixels retrieved}) + (\text{Relevant pixels not retrieved}))} \times 100
  \]

  \text{Equation 8: Recall}

- **Precision**: is the ratio of the number of relevant pixels retrieved to the total number of irrelevant and relevant pixels retrieved:

  \[
  Precision = \frac{(\text{Relevant pixels retrieved})}{((\text{Irrelevant pixels retrieved}) + (\text{Relevant pixels retrieved}))} \times 100
  \]

  \text{Equation 9: Precision}
Results are shown in table 3:

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Otsu-threshold:</td>
<td>39</td>
<td>65</td>
</tr>
<tr>
<td>Adaptive threshold</td>
<td>36</td>
<td>69</td>
</tr>
</tbody>
</table>

Results are similar for both types of thresholding. Adaptive has higher recall and lower precision than Otsu. Algorithm must be improved for get better results especially in Precision. In next chapter, future work it is explained some improvements that could be implemented.
3. Future work

As it commented before, algorithm must be improved to obtain better results. The part that could be improved significantly is the process of character location the letters. In our algorithm, image is segmented and after that filtered for try to separate letters than noise in background. One great improvement will be first detected where text segmentation is and segment only this part. It is complicated as letters has different colors, size and algorithm becomes very complicated but results will be really good.

Moreover, we will try to find a solution to the problem that some letters are darker and some of them are lighter in the same image. Nevertheless, if letters are detected before, this problem will be easy solved checking each letter is darker or lighter.
4. Conclusions

I can make some different conclusions of this thesis. On one hand, creation of an algorithm for all different images is really complicated. Best algorithm that has been created for different contests as ICDAR can not segment all the text in all pictures without mistakes or noise. The algorithm created is not as good as others, but with some improvements that have been commented it could yield good results.

On the other hand, image processing is a very interesting world that will have my attention in the future. I really discover an interesting field on engineering that I didn’t know it before.

Finally, this research really has helped me to improve my skills in programing. Working with C++ and visual studio has been a challenge for me and now I can use this language.
5. Acknowledgments

I am very grateful to my mentor on this thesis, Dr. Sc. Matko Šarić for give me the opportunity to work with him and for all his advices and recommendations.

As well to the Sveučilište u Splitu in general and to the Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture (FESB) for let me study and learn in this institution.

Finally, thanks to ESN Split and all friends I have met here for help me to adapt to Split and let me feel like at home.
6. References


[22] Sergey Milyaev, Olga Barinova, Tatiana Novikova, Pushmeet Kohliy, Victor Lempitskyy Lomonosov “Image binarization for end-to-end text understanding in natural images”.


Annex 1: Algorithm with Adaptive threshold

```cpp
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <iostream>
#include <dirent.h>

using namespace cv;
using namespace std;

#define CONTOUR_AREA 75
#define CONTOUR_AREA_MAX 25000

bool process_image(std::string name, std::string src_folder, std::string dst_folder) {
    // Read image and transform to grayscale
    cv::Mat image = cv::imread(src_folder + name, CV_LOAD_IMAGE_GRAYSCALE);
    if (image.empty()) return false;
    image.convertTo(image, CV_8U);

    // Blur the image
    Mat blurred;
    GaussianBlur(image, blurred, Size(3, 3), 1.5);

    // Filter opening
    cv::Mat const structure_elem = cv::getStructuringElement(
        cv::MORPH_RECT, cv::Size(5, 5));
    cv::Mat open_result;
    cv::morphologyEx(blurred, open_result, cv::MORPH_OPEN, structure_elem);

    // Threshold
    cv::Mat image_threshold;
    cv::adaptiveThreshold(open_result, image_threshold, 255,
        CV_ADAPTIVE_THRESH_GAUSSIAN_C, CV_THRESH_BINARY, 51, 10);

    // Filter closing
    cv::Mat const structure_elem2 = cv::getStructuringElement(
        cv::MORPH_ELLIPSE, cv::Size(3, 3));
    cv::Mat close_result;
    cv::morphologyEx(image_threshold, close_result, cv::MORPH_CLOSE, structure_elem2);

    // Find contours
    vector<vector<Point>> contours;
    vector<Vec4i> hierarchy;
    cv::findContours(close_result, contours, hierarchy, CV_RETR_CCOMP,
        CV_CHAIN_APPROX_SIMPLE, Point(0, 0));
    Mat drawing = Mat::zeros(close_result.size(), CV_8UC3);
    vector<vector<Point>> contours_poly(contours.size());
    vector<Rect> boundRect(contours.size());
```
for (int idx = 0; idx >= 0; idx = hierarchy[idx][0])
{
    // calculate parameters for filter the letters
    approxPolyDP(Mat(contours[idx]), contours_poly[idx], 3, true);
    boundRect[idx] = boundingRect(Mat(contours_poly[idx]));
    float occupyrate;
    occupyrate = (contourArea(contours[idx]) / (boundRect[idx].width * boundRect[idx].height));

    float aspctratio;
    aspctratio = max(boundRect[idx].height, boundRect[idx].width) / min(boundRect[idx].height, boundRect[idx].width);
    float perimeter;
    perimeter = arcLength(contours[idx], true);

    float compactness;
    compactness = contourArea(contours[idx]) / (perimeter * perimeter);
    // filter contours by region areas and parameters and draw
    RNG rng(12345);

    if((contourArea(contours[idx]) > CONTOUR_AREA) & (contourArea(contours[idx]) <= CONTOUR_AREA_MAX))
    {
        if ((occupyrate >= 0.03) & (occupyrate <= 0.95))
        {
            if (aspctratio <= 6)
            {
                if (compactness > 0.003 & compactness <= 0.95)
                {
                    Scalar color(rand() & 255, rand() & 255, rand() & 255);
                    drawContours(drawing, contours, idx, color, CV_FILLED, 8, hierarchy);
                }
            }
        }
    }
    cv::imwrite(dst_folder + name, drawing);

    return true;
}
```c
int wmain(int argc, wchar_t* argv[]) {
    // Read images and save results in folders
    DIR *directory;
    struct dirent *entry;
    std::vector<std::string> names;

    std::string src = "C:/icdar/icdar2013/";
    std::string dst = "C:/Final/Adaptativethreshold/";

    if ((directory = opendir(src.c_str())) == NULL) {
        std::cerr << "Could not open directory" << std::endl;
        return 1;
    }

    while ((entry = readdir(directory)) != NULL) {
        std::string name = entry->d_name;
        if (name != "." && name != "..") names.push_back(name);
    }

    for (std::vector<std::string>::iterator it = names.begin(); it != names.end(); it++) {
        if (!process_image(*it, src, dst)) std::cout << "Problems processing image: " << *it << std::endl;
    }

    return 0;
}
```
Annex 2: Algorithm with Otsu threshold

```cpp
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <iostream>
#include <dirent.h>

using namespace cv;
using namespace std;

#define CONTOUR_AREA 100
#define CONTOUR_AREA_MAX 50000

bool process_image(std::string name, std::string src_folder, std::string dst_folder) {
    // Read image and transform to grayscale
    cv::Mat image = cv::imread(src_folder + name, CV_LOAD_IMAGE_GRAYSCALE);
    if (image.empty()) return false;
    image.convertTo(image, CV_8U);

    // Blur the image
    Mat blurred;
    GaussianBlur(image, blurred, Size(3, 3), 1.5);

    // Threshold
    Mat otsu_threshold;
    cv::threshold(blurred, otsu_threshold, 0, 255, CV_THRESH_BINARY | CV_THRESH_OTSU);

    // find contours
    vector<vector<Point>> contours; vector<Vec4i> hierarchy;
    cv::findContours(otsu_threshold, contours, hierarchy, CV_RETR_CCOMP,
                     CV_CHAIN_APPROX_SIMPLE, Point(0, 0));

    Mat drawing = Mat::zeros(otsu_threshold.size(), CV_8UC3);
    vector<vector<Point>> contours_poly(contours.size());
    vector<Rect> boundRect(contours.size());

    // Find contours
    vector<vector<Point>> contours;
    vector<Vec4i> hierarchy;
    cv::findContours(otsu_threshold, contours, hierarchy, CV_RETR_CCOMP,
                     CV_CHAIN_APPROX_SIMPLE, Point(0, 0));

    Mat drawing = Mat::zeros(otsu_threshold.size(), CV_8UC3);
    vector<vector<Point>> contours_poly(contours.size());
    vector<Rect> boundRect(contours.size());
}
```
for (int idx = 0; idx >= 0; idx = hierarchy[idx][0])
{
    // calculate parameters for filter the letters
    approxPolyDP(Mat(contours[idx]), contours_poly[idx], 3, true);
    boundRect[idx] = boundingRect(Mat(contours_poly[idx]));
    float occupyrate;
    occupyrate = (contourArea(contours[idx]) / (boundRect[idx].width * boundRect[idx].height));
    float aspectratio;
    aspectratio = max(boundRect[idx].height, boundRect[idx].width) / min(boundRect[idx].height, boundRect[idx].width);
    float perimeter;
    perimeter = arcLength(contours[idx], true);
    float compactness;
    compactness = contourArea(contours[idx]) / (perimeter * perimeter);
    // filter contours by region areas and parameters and draw
    RNG rng(12345);
    
    if ((contourArea(contours[idx]) > CONTOUR_AREA) &
        (contourArea(contours[idx]) <= CONTOUR_AREA_MAX))
    {
        if ((occupyrate >= 0.0025) & (occupyrate <= 0.95)) {
            if (aspectratio <= 15){
                if (compactness > 0.005 & compactness <= 0.95){
                    Scalar color(rand() & 255, rand() & 255, rand() & 255);
                    drawContours(drawing, contours, idx, color, CV_FILLED, 8, hierarchy);
                }
            }
        }
    }

    cv::imwrite(dst_folder + name, drawing);
}

return true;
Annex 3 Algorithm check results

Next functions are used in Matlab to check the results

```matlab
function [precision recall f_score]=get_results(final_extraction, gt_image)

    final_extraction=uint8(final_extraction)*255;
    [x y z]=size(gt_image);
    test=zeros(x,y,z);

    true_positive=0;
    for i=1:x
        for j=1:y
            if(gt_image(i,j)==255)
                gt_image(i,j)=0;
            else
                gt_image(i,j)=255;
                if(final_extraction(i,j)==255)
                    true_positive=true_positive+1;
                    test(i,j)=255;
                end
            end
        end
    end

    % gt_image=rgb2gray(gt_image);
    % gt_image=imcomplement(im2bw(gt_image));

    difference=gt_image-final_extraction;
    %true_positive=length(find((difference==0)&(final_extraction==255)));
    total=length(find(gt_image==255));
    total_detected=length(find(final_extraction==255));

    precision=true_positive/total_detected;
    recall=true_positive/total;
    f_score=2*precision*recall/(precision+recall);
end
```
function [precision, recall] = read
for i=100:328
    i2=num2str(i);
    file=strcat(i2,'.jpg');
    image=imread(file);
    A = rgb2gray(image);
    text=i2;
    textMedio='_GT';
    textFinal='.bmp';
    file2 = strcat(text,textMedio,textFinal);
    imagegt=imread(file2);
    B = rgb2gray(imagegt);
    [precision, recall, f_score]=get_results(A,B);
    F = [precision, recall, f_score];
    j = i;
    M (j,1) = precision
    L (j,1) = recall
    j = j+1;
end
end