SEGMENTATION ALGORITHMS FOR EXTRACTION OF IDENTIFIER CODES IN CONTAINERS

Juan A. Rosell Ortega, Alberto J. Pérez Jiménez and Gabriela Andreu García
Grupo de Visión por Computador , DISCA.
Universidad Politécnica de Valencia
Camino de Vera s/n, Valencia, Spain

Keywords: segmentation algorithms, characters.

Abstract: In this paper we present a study of four segmentation algorithms with the aim of extracting characters from containers. We compare their performance using images acquired under real conditions and using results of human operators as a model to check their capabilities. We modified the algorithms to adapt them to our needs. Our aim is obtaining a segmentation of the image which contains all, or as much as possible, characters of the container’s code; no matter how many other non-relevant objects may appear; as irrelevant objects may be filtered out by applying other techniques afterwards. This work is part of a higher order project whose aim is the automation of the entrance gate of a port.

1 INTRODUCTION

A lot of work has been done in the area of comparing segmentation algorithms attending to different categorizations. There have been a number of survey papers on thresholding: for instance, in (Sezgin and Sankur, 2004), a taxonomy of thresholding techniques is presented, and several algorithms are categorized, expressed under a uniform notation and compared according to unified criteria.

We present in this paper, however, a comparison of four segmentation algorithms aimed to deal with pictures containing container code characters. The goal of the work is not to compare the performance of the algorithms in general conditions, but with a wide set of images that correspond to images representative of what the system will have to deal with in real situations (see figure 1 as a sample of the images we used). Our criterion to check results was comparing the results of each algorithm with the results obtained by a human operator.

Currently in most trading ports, gates are controlled by human inspection and manual registration. This process can be automated by means of computer vision and pattern recognition techniques. Such a prototype should be built by developing different techniques, such as image preprocessing, image segmentation, feature extraction and pattern classification. The process is complex; because it has to deal with outdoor scenes, days with different climatology (sunny, cloudy...), changes in light conditions (day, night) and dirty or damaged containers.

A first approach to the process of code detection is presented in a previous work (Salvador et al., 2001) and the overall process is discussed also in (Salvador et al., 2002). In these works, authors use a morphological operator called top-hat to segment the same kind of images as we do. This operator has to be applied twice per image, once trying to distinguish
clear objects on a dark background and again looking
dark objects on a clear background. Though this
method had good results, we tried to improve their
performance by using the methods we expose in this
article.

Another work (Atienza et al., 2005), presents an
investigation which is currently being developed. The
aim of the authors of this paper is to use the optical
flow in order to shrink the area where the container
code could be found and speed up the segmentation
process; however, the method is time consuming and
an effort is currently being done in order to optimize
it.

Our aim was finding a suitable successful segmen-
tation algorithm for the process of code detection
mentioned previously. To achieve this goal, we tested
several segmentation algorithms found in the litera-
ture. By comparing the results obtained from the four
algorithms we tested, we extracted conclusions about
which could perform better under the conditions we
work with. We wanted to know which algorithm (or
algorithms) could fit better to our needs, and, if it was
the case, which we could merge into one, that could
produce better results than any other algorithm on its
own.

In summarize, we are looking for an algorithm that
meets the following constraints:

- It must detect all characters in the code, or as much
  as possible.
- It must find characters independently of their
  colour (white characters on a dark background and
  vice versa).
- It must be independent of image light conditions.
- It must create the minimal list of found objects; as
  segmentation algorithms will always find objects
  which are not relevant (see picture 4), the best will
  be the one which creates a list that only contains
  objects which correspond to characters in the code.

We have organized the paper as follows, in next
section we describe the algorithms used, section 3 will
describe the pictures used and the experiments done;
in section 4 we show the results we obtained in the
experiments and in section 5 we discuss our conclu-
sions.

2 ALGORITHMS

As mentioned before, we have implemented and com-
pared four different algorithms. In this section we
describe them briefly and we give a formal introdun-
tion to each one. These algorithms are: the water-
shed algorithm (Beucher and C-Lantuéjoul, 1979),
LAT (Kirby and Rosenfeld, 1979), local variation
algorithm (Felzenszwalb and Huttenlocher, 1998) and
thresholding algorithm. Each algorithm represents a
different approach to the segmentation problem. Our
election was based on previous knowledge of the al-
gorithms and also on the comparison of the literature
descriving them.

2.1 Watershed

Application of the watershed transform to image
segmentation was proposed in (Beucher and C-
Lantuéjoul, 1979). The algorithm takes a grayscale
image and considers it as a topographic surface. A
process of flooding is simulated on this surface; dur-
ing this process two or more floods coming from dif-
ferent basins may merge. To avoid this, dams are built
on the points where the waters flooding from different
basins meet; at the end of the algorithm, only dams
are over water level. These dams define the watershed
of the image.

One of the problems of this algorithm is over seg-
mentation. We took no direct action to avoid it; in-
stead, we used filters (as mentioned later in section 3)
to remove regions from the segmentation which were
not relevant.

2.2 LAT

This algorithm was proposed by Kirby and Rosenfeld
in (Kirby and Rosenfeld, 1979). Adaptive threshold-
ing typically takes a grayscale or colour image as in-
put and, outputs a binary image representing the seg-
mentation. For each pixel in the image, a threshold
has to be calculated. If the pixel value is below the
threshold it is set to the background value, otherwise
it assumes the foreground value; or vice versa.

The algorithm uses a window \( w_{n \times n}(p_i) \) of \( n \times n \) ele-
ments (the neighbourhood to consider for each pixel
\( p_i \)), \( n \) is odd in order to centre the matrix on the con-
sidered pixel of the image. This matrix is used to cal-
culate statistics to examine the intensity \( f(p_i) \) of the
local neighbourhood of each pixel \( p_i \). The statistic
which is most appropriate depends largely on the in-
put image. We used the mean of the local intensity,
computed as:

\[
\mu(p_i) = \frac{1}{n \times n} \sum_{p_j \in w_{n \times n}(p_i)} f(p_j) \tag{1}
\]

The size of the considered neighbourhood has to be
large enough to cover sufficient foreground and
background pixels, otherwise a poor threshold is cho-
en. On the other hand, choosing regions which are
too large can violate the assumption of approximately
uniform illumination.

A factor \( k \) may be used to adjust the comparison of
the statistic used with the pixel intensity value. This
factor multiplies the average gray level when comparing it to the gray level of the pixel.

The algorithm forms the segmentation of image $I$ by calculating the union of the segmentations result of using different factors, converting image $I$ into binaryized image $I'$. Each pixel in $I'$ is defined depending on its counterpart in $I$:

$$
\forall p_i' \in I', p_i' = \text{thr}(p_i) : p_i \in I
$$

with $\text{thr}(p_i)$ defined as:

$$
\text{thr}(p_i) = \begin{cases} 
1, & \text{if } \mu(p_i) \geq k \times f(p_i) \\
0, & \text{otherwise}
\end{cases}
$$

We looked for connected regions after each thresholding and merged the results of each segmentation with the previous. So, if a set of pixels belonged to any region for a certain segmentation, they did in the final segmentation.

### 2.3 Thresholding

It is the simplest approach to image segmentation. The input is a gray level image $I$ and the output is a binary image $I'$ representing the segmentation where black pixels correspond to background and white pixels correspond to foreground (or vice versa). In a single pass, each pixel in the image is compared with a given threshold $k$ which is a gray level. If the pixel's intensity $f(p_i)$ is higher or equal to the threshold $k$, the pixel $p_i$ is set to, say, white in the output. If it is under $k$, it is set to black. The segmentations result of applying each threshold $k$ are joined in a similar way as in LAT.

### 2.4 Local Variation

The algorithm we implemented is based on the work introduced in (Felzenszwalb and Huttenlocher, 1998). An important feature of this algorithm over the other three is that it does not need a priori information on which the colour of the objects is.

Felzenszwalb’s approach consists on considering a criterion for segmenting images based on intensity differences between neighbouring pixels. The algorithm is based on the idea of partitioning an image into regions, such that for each pair of regions the variation between them is bigger than the variation within regions. The criterion for determining the similarity of image regions is based on measures of image variation. The measure of the internal variation of a region is an statistic of the intensity differences between neighbouring pixels in the region. The measure of the external variation between regions is the minimum intensity difference between neighbouring pixels along the border of the two regions. The original algorithm uses two parameters, the minimum size of the regions in the final result, and one used to smooth the image before processing it.

Felzenszwalb starts by creating a graph that represents the image. Arches in the graph are given a weight that corresponds to the difference in intensity of pixels represented by their vertices. Arches join pixels to their 8-connected neighbourhood. To achieve the fastest ordering, original authors recommend in their paper the bucket sort algorithm (Cormen et al., 1990). After this step, all arches are ordered by non-decreasing weight. The algorithm then follows by taking an arch at a time and compare the regions to which each of its ends belongs to. Both regions will be merged if they accomplish with the established criteria. The output of the algorithm is a set of regions in which the image is segmented.

Formally, the graph $G$ on the image $I$ is defined as follows. Each pixel $p_i$ in the image will correspond to a vertex $v_i$ in the graph. Arches connect neighbouring pixels and each arch is assigned with a weight. The function used to calculate the weight of the arches is defined as follows:

$$
w(v_i, v_j) = \begin{cases} 
|I(v_i) - I(v_j)|, & \text{if } (v_i, v_j) \in E \\
\infty, & \text{otherwise}
\end{cases}
$$

$E$ is the set of all edges defined. As we said, for each vertex $v_i$ in the graph exists a pixel $p_i$ in the image, thus, $E$ is the set of all arches connecting vertices in $G$ for some given distance $d$ (in our case, $d = 1$, so we only consider immediate neighbours), $E = \{(v_i, v_j) : \|p_i - p_j\| \leq d\}$. The segmentation $S$ of an image $I$ is then a partition of $I$ with a corresponding set of edges, $G \subseteq E$, such that each component $C$ in the segmentation, $C \in S$, corresponds to a connected component in the graph $G$.

We used the mean intensity of the region to decide whether two regions should merge or not instead of the variance as in the original algorithm. Following a similar approach as Felzenszwalb does in his algorithm, we decided that two different regions should join into one if the mean intensity of both regions is similar. Another difference is that we didn’t use a minimum size of region, in fact, regions were merged until the process reaches a situation in which no more regions could be merged.

We modified the original algorithm because it was difficult to find values for its parameters that could perform well with the variety of images we had. We used the mean of gray intensity of the region which, after several tests, proved to perform similar to the original algorithm. In our case, however, we used one parameter, the percentage of similarity $k$. For each value of $k$ we have a different segmentation, all these resulting segmentations are then joined as in the previous algorithms.
3 EXPERIMENTS

We used 309 real images with each algorithm in order to obtain experimental results to evaluate their performance. These images represent truck containers and have a size of $720 \times 574$ pixels in gray levels. They were acquired under real conditions in the admission gate of the port of Valencia. In figure 2, there is a sample of the kind of pictures we used. They were selected randomly between a large set of pictures, but we assured all variability was represented in this set of pictures (sunny or cloudy days, daytime or nighttime, damaged containers...).

In order to extract conclusions about the performance of the algorithms; we labelled by hand all characters in the images. We made this by drawing the inclusion box of each character in the image and saving the coordinates, an example is shown in figure 3. As a result of this labelling, we had the number of code characters and the coordinates of the inclusion boxes of each code character of each image. We automated the processing of the experiment’s results.

During experiments, algorithms were not provided with concrete information about light conditions of each image during their executions, because we wanted to know how they would perform in real conditions.

Algorithms’ parameters vary in a range wide enough to cover all possible situations. As watershed needs no arguments, we need not to use any parameter. For LAT we used a value of $k$ in the range of $[0.9, 1.6]$ with a step value of 0.03, which makes a total of 23 segmentations per image. For thresholding algorithm we used a value of $k$ ranging from 20 to 220 with a step value of 5, which makes up 40 segmentations per image. With Local Variation we used a value of the similarity percentage in the range $[70\%, 85\%]$ with a step of $5\%$ which makes a total of 4 segmentations. Besides, each algorithm (but Local Variation) had to be executed twice per image, once searching for white characters and again for black characters. Because of its nature, Local Variation finds directly characters no matter which colour they are.

We obtained of each algorithm a list of found objects in each image. These objects correspond to connected regions with equal gray intensity levels. Some of these objects were not relevant to our purposes, and some others corresponded to the code characters. As it may be seen in picture 4, the number of objects is bigger than the number of code characters.

We evaluate the good or bad performing of algorithms depending on the amount of code characters included in the list of objects it found, not in the total amount of found objects. Other procedures applied after this segmentation step may be implemented to reduce the number of irrelevant objects. With a geometrical filter we removed any object which did not meet the geometrical properties we expect of any character in the code (aspect ratio, minimum area size, height and width).

Other filters could have also been applied. For instance, contrast filters or classifier filters, that would remove all objects which were not classified as a character. But there was the chance that filters hided errors of the algorithms or add their own errors to the final result, masking the performance of the algorithms.

We set up a method to compare the inclusion boxes calculated by each algorithm with the inclusion boxes we had labelled manually. We considered the algorithm had done a hit if the inclusion box it had calculated and the hand labelled inclusion box overlapped one on each other in a certain percentage.

We obtained the number of hits per image of each algorithm, the number of failures (labels which had no counterpart in the result list of the algorithm), the
4 RESULTS

The first results we obtained are shown in table 1. In this table, we see that LAT is the one that performs better. It is the one that gets the biggest amount of images with up to 4 characters missed. Watershed is the second algorithm that performs best. On the other hand, the adaptation we made of the algorithm of local variation has an erratic behaviour, this is maybe due to the fact that we were only using the mean average of the regions to decide whether to merge them or not (recall section 2.4). Values on table 1 are plotted in figure 5.

In table 2, we show the mean time of execution for each algorithm (in seconds). LAT is the fastest, and its execution time is far away from the amount of time needed by the local variation algorithm (the slowest) to execute, which is also the algorithm with the worst performance.

After these first results, we thought it would be a good idea trying to merge the best two algorithms into one, in order to get benefits from both. Merging the algorithms consisted on applying them both on the same image and take results together applying the filters to the this merged output.

We repeated the tests taking both LAT and watershed algorithm as the basis of our merged algorithm. We merged also LAT and thresholding algorithm, though the last is not the second best, it is fast and this could mean more hits with less execution time. We thought this was a good enough reason to give it a try. We made also a test joining these three algorithms into one.

We were puzzled at first by the poor performance of the local variation algorithm. We concluded we needed to adapt the same philosophy we used for LAT and let the algorithm iterate several times over the same image looking for objects, and using in each iteration a different criterion \( k \) to merge regions. In table 3 we compare the results we obtained with the LAT-Watershed algorithm, the modified local variation algorithm and the LAT-Watershed-Thresholding.
algorithm (called LWT in the table). We can see also that the algorithm made by joining watershed algorithm and LAT performs better than the rest, and with the same performance as the serialization of thresholding algorithm with watershed algorithm and LAT.

In table 4 we show the times consumed by the merged versions; the algorithm formed by the serialization of LAT and thresholding algorithm is a bit quicker than the sum of the times of both algorithms taken separately. In 6 we can see a graphical comparison of the improved algorithms.

Table 3: Performance of the improved algorithms, compared with the two algorithms with best results.

<table>
<thead>
<tr>
<th>Missed char.</th>
<th>LAT-Thr.</th>
<th>LWT</th>
<th>LAT-Wat.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>237</td>
<td>253</td>
<td>253</td>
</tr>
<tr>
<td>1</td>
<td>53</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4: Mean time of execution of the improved algorithms.

<table>
<thead>
<tr>
<th>Mean time</th>
<th>LAT-Thr.</th>
<th>Lat-Wat-Thr.</th>
<th>Lat-Wat</th>
</tr>
</thead>
<tbody>
<tr>
<td>seconds</td>
<td>2.67</td>
<td>10.18</td>
<td>8.19</td>
</tr>
</tbody>
</table>

5 CONCLUSIONS

We present a comparison of four segmentation algorithms. We checked their performance dealing with pictures with character codes on containers. We used 309 real images selected randomly between a large set of real images, and compared the performance of algorithms against the results given by a human operator.

Our effort was driven by the fact that we wanted to find the algorithm that did not miss any character (or the lowest amount possible) in the container’s code. Our evaluation of the algorithms, then, penalizes algorithms which lose characters in the code. Future efforts will focus on developing filters which allow us to remove regions which are not relevant to our search.

As results show, LAT or watershed algorithm meet the criteria listed above, of having a big rate of success segmenting characters, finding a big percentage of them in the images. Mixing both of them into one improves results with the penalty of having longer execution times.

REFERENCES


