Processing and recognition of characters in container codes *

JUAN ROSELL
GABRIELA ANDREU
ALBERTO PÉREZ
Universidad Politécnica de Valencia
DISCA
Camino de Vera s/n,Valencia 46022
SPAIN

Abstract: This paper describes the process of location and recognition of container code characters. The system has to deal with outdoor images. Top hat transformation, segmentation algorithms and filters have been applied in order to locate the container's registration number. Our aim is to obtain a list of characters which contains all, or as much as possible, characters of the container's registration number in order to recognize them. This work is part of a higher order project whose aim is the automation of the entrance gate of a port.

Key–Words: computer vision, segmentation, character recognition.

1 Introduction

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 process 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. Digits and characters may be clear and/or dark and some may appear framed to distinguish them from the rest of the code.

A first approach to the process of code detection is presented in a previous work [1] and the overall process is discussed also in [2]. In these works, authors use a morphological operator called top hat [3] to segment the images of containers. Though this method had good results, we tried to improve its performance.

The task of finding the best segmentation method of an specific application is still a difficult challenge. We have the classical definition of segmentation of an image I defined as a partition of the image into components which verify that $C_i \cap C_j = \emptyset \wedge i \neq j$ and that $\bigcup C_i = I$. In [4] we compared the performance of four segmentation algorithms applied to images containing truck containers. In that work, our segmentation was based on applying the classical definition of segmentation to an image, varying the algorithm's parameters and using two or more techniques at a time on the same image, in an effort to reduce errors.

Our input data are grayscale images I, being $f(x, y)$ the gray level of pixel located in coordinates (x, y). Our goal is to segment the image I into a set of objects which contain the container's code.

Formally, we represent as $\theta(I, S, k_i)$ the set of objects obtained as result of applying the segmentation method S to image I with a certain set of parameter values k_i. The set k_i may be empty if the segmentation technique S has no parameter. The set $\theta(I, S, k_i)$, will contain either relevant and irrelevant objects for our search of the code. Given $\theta(I, S, k_i) = \{p_0, p_1 \dots p_n\}$ where each p_j is determined by a minimum rectangle that encloses the object contained in the region. This rectangle is defined by two corners, being cs_j the top left corner and cf_j the bottom right corner. It is trivial to deduce that given $cs_j = (x_j, y_j)$ then we will have that $cf_j = (x_j + m_j, y_j + n_j)$ where $m_j, n_j \in \mathcal{N}$, also that the size of this minimum rectangle R_{p_j} enclosing object p_j is $m_j \times n_j$.

As we mentioned above, our final segmentation of an image, will be the result of adding previous segmentations of the same image with different values in the algorithm's parameters. If we consider a set of parameters K as $K = \{k_0, k_1 \dots k_q\}$ in a segmentation process, we will obtain a set of objects

*Acknowledgements:This work has been partially supported by grant FEDER- CICYT DPI2003-09173-C02-01.

$\Upsilon(I,S,K) = \bigcup_{i=1}^{q} \theta(I,S,k_i)$ which will be the result of the final segmentation.

However, this set $\Upsilon(I,S,K)$ contains objects which are irrelevant for our search of characters, and we need to define a way to clean up these objects. The way is to implement filters. Applying a filter to $\Upsilon(I,S,K)$ means subtracting a determined set of objects Δ from $\Upsilon(I,S,K)$, such that $\Delta \subseteq \Upsilon(I,S,K)$ where $\Delta = \{p \in \Upsilon(I,S,K) : \sigma(p) = 0\}$. We call the function $\sigma(p)$ the filtering function, which will be a map $\Upsilon(I,S,K) \rightarrow \{0,1\}$. If we have a family of filters Σ, we can define the final set, result of the segmentation and the filtering as:

$\Gamma(I,S,K,\Sigma) = \{\Upsilon(I,S,K) - \bigcup \Delta_{\sigma_i \in \Sigma}\}$

Abusing notation, we will call this set $\Gamma(I,S,K)$. In this paper, we explain which filters we used to remove this useless regions.

Authors of [5], present an investigation which is currently under development. 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, this method has proved to be time consuming, and currently efforts are done in order to optimize it. In a future, we expect both investigations to merge.

We have organized the paper as follows, in section 2 we describe the complete process that we follow to extract the characters from pictures, in section 3 we will describe the data we used, in section 4 we describe he experiments done; in section 5 we show the results we obtained in the experiments and in section 6 we discuss our conclusions.

2 Process

We propose a process to extract and identify characters in images representing truck containers that can be divided in the following four phases (see figure 1):

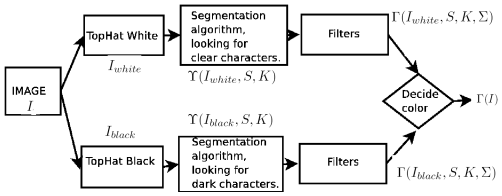


Figure 1: The recognition process step by step.

- step 1. Pre-process the image by applying the top-hat technique for clear and for dark levels. After this step, we have two different images I_{white} and I_{black}.
- step 2. Apply a segmentation algorithm to each one of the images resulting of the previous step. We look for clear characters in the image result of applying the white top hat and, we look for dark characters in the result of black top hat.
- step 3. Filter the output of the segmentation algorithm for each image.
- step 4. Decide whether characters in the original image were clear or dark based on the number of objects left in each set $\Gamma(I_{white},S,K)$ and $\Gamma(I_{black},S,K)$.

2.1 Top hat operator

In our task, we have to deal with clear and dark characters. Top hat is a useful operation for enhancing details when there are shadows. Let I be a grayscale image and b an structuring element (SE). The opening and closing grayscale operations are defined in terms of erosions and dilations as:

$$\omega(f(x,y)) = max_{(i,j)\in b}(min_{(i,j)\in b}(f(x+i,y+j) - b(i,j))) \quad (1)$$

$$\phi(f(x,y)) = min_{(i,j)\in b}(max_{(i,j)\in b}(f(x+i,y+j) + b(i,j))) \quad (2)$$

The tophat transformation ([3],[6]) consists on using knowledge about the shape characteristics that are not shared by the relevant image structures. Relevant structures can be removed by opening and closing operations with a SE that does not fit them. White tophat is defined as $WTH(I) = I - \omega(I)$ and black tophat as $BTH(I) = \phi(I) - I$.

2.2 Filters

In this section, we explain the filters we used. They are introduced in the same way they are applied in the process.

2.2.1 Shape filter

This filter removes all those regions which do not fit to the expected dimensions of characters (height and width measured in pixels). It is executed first because it is the filter that can remove more objects and one of the fastest to be applied. For each object $p \in \Upsilon(I,S,K)$ whose enclosing rectangle is R_p. We define the filtering function as:

$$f_{shape}(p) = \begin{cases} 1 & \text{if } height(R_p) \in [20,50] \\ & \wedge\ width(R_p) \in [6,30] \\ 0 & \text{any other case} \end{cases}$$

where, the functions $height(R_p)$ and $width(R_p)$ return the height and width respectively of the rectangle R_p. Formally, we can define this filter as:

$$\Delta_{shape} = \{p \in \Upsilon(I,S,K) : f_{shape}(p) = 0\}$$

2.2.2 Contrast filter

This filter gets rid of objects whose contrast is too low. Regions which do not show enough variability are removed. If we define the variance of each object $p \in \Upsilon(I,S,K)$ as $\mu(p)$ we can define the filtering function as:

$$f_{contrast}(p) = \begin{cases} 1 & \text{if } \mu(p) > 15 \\ 0 & \text{any other case} \end{cases}$$

and the set $\Delta_{contrast}$ as:

$$\Delta_{contrast} = \{p \in \Upsilon(I,S,K) : f_{contrast}(p) = 0\}$$

2.2.3 Classifier based filter.

This filter labels each element in the list of objects by using a k-Nearest Neighbours classifier. We trained the k-NN classifier with symbols extracted from real images of truck containers. We obtained these symbols from the images by segmenting automatically each image and labelling each symbol by hand. We assigned a class to each character up to a total of 36 classes (letters and numbers), and an extra class for noise. Considering noise as an extra class is important because, though contrast and shape filter do get rid of most objects which are not valid, there still remains a big amount of objects which are not characters. By applying the classifier only trained with characters, we would label these invalid regions as character and they are not. Noise class allows us to improve the recognition step by labelling regions of the image as noise. Once the classifier has labelled all objects, all those which end up labelled as noise are removed from the list.

A function $class(p)$ models the k-NN classifier, returning a symbol for each $p \in \Upsilon(I,S,K)$ or RR for objects classified as noise. The filtering function $f_{noise}(p)$ will be defined as:

$$f_{noise}(p) = \begin{cases} 0 & \text{if } class(p) = RR \\ 1 & \text{any other case} \end{cases}$$

and the set Δ_{noise} as :

$$\Delta_{noise} = \{p \in \Upsilon(I,S,K) : f_{noise}(p) = 0\}$$

2.2.4 Fusion filter

As segmentation algorithms were applied to images varying their parameters, more than one object was found with similar coordinates corresponding to the same character in the code. This filter fused into one all those objects which presumably represented the same character in the code. For a given pair of rectangles R_p and R_q enclosing objects p and q, $p \wedge q \in \Upsilon(I,S,K)$ we define their intersection $R_p \bigcap R_q$ as the percentage of surface that R_p overlaps on R_q. We define then the filter's function as:

$$f_{fusion}(p) = \begin{cases} 0 & \text{if } \exists\, q \in \Upsilon(I,S,K) : R_p \bigcap R_q \geq 0.65 \\ & \wedge\, p \neq q \\ 1 & \text{any other case} \end{cases}$$

and the set Δ_{fusion} as :

$$\Delta_{fusion} = \{p \in \Upsilon(I,S,K) : f_{fusion}(p) = 0\}$$

This filter is applied after the classifier to avoid fusing objects which will end up labelled as noise with objects labelled as a character or a number.

2.2.5 Confidence criterion

This criterion counts how many objects in the result set $\Upsilon(I,S,K)$ have a confidence given by the classifier over 80%. This gives us an indicator of the global confidence of the objects in the result set, the more objects with a high confidence, the higher the probability the set contains the code characters. The function $confidence(p)$ for $p \in \Upsilon(I,S,K)$ can be calculated as:

$$confidence(p) = \frac{votes}{neighbours}$$

Where $votes$ is the number of votes of the label assigned to p by the k-NN classifier and $neighbours$ is the number of neighbours that voted, $neighbours$ must be bigger than 1. The set associated to this criterion is simply:

$$\Xi(I,S,K) = \{p \in \Upsilon(I,S,K) : confidence(p) > 0.8\}$$

Recalling figure 1, the process can be seen as: given an image I, we pre-process it by applying tophat, we will call I_{white} to the result of applying the white tophat to I and I_{black} to the result of applying black tophat to I. We then apply a segmentation technique S to I_{white} and to I_{black} with a set of parameters K, and produce two result sets, $\Upsilon(I_{white},S,K)$ and $\Upsilon(I_{black},S,K)$.

Applying the filters to these two sets we get:

$$\begin{aligned}\Gamma(I_{white},S) &= \Upsilon(I_{white},S,K) - \Delta_{shape} \\ &\quad - \Delta_{contrast} - \Delta_{classifier} \\ &\quad - \Delta_{fussion}) \end{aligned} \quad (3)$$

$$\begin{aligned}\Gamma(I_{black},S) &= \Upsilon(I_{black},S,K) - \Delta_{shape} \\ &\quad - \Delta_{contrast} - \Delta_{classifier} \\ &\quad - \Delta_{fussion} \end{aligned} \quad (4)$$

We use the uniformity criterion to decide which is the correct set. If $|\Xi(I_{white},S)| > |\Xi(I_{black},S)|$ we will decide that characters were white, if $|\Xi(I_{black},S)| > |\Xi(I_{white},S)|$ we will decide that characters were black, and in the case that $|\Xi(I_{black},S)| = |\Xi(I_{white},S)|$ we can decide nothing.

3 Data

We used 1275 real images to perform our experiments. These images represent truck containers and have a size of 720×574 pixels in gray levels. They were acquired under real conditions in the admission gate of the port of Valencia; in several days under different light conditions. Digits and characters can be clear or dark and they appear in both plain and non-plain surfaces. We selected randomly a set from a large amount of pictures and assured all variability was represented in this set of pictures (sunny or cloudy days, daytime or night-time, damaged containers...).

4 Experiments

We used three well known segmentation algorithms in our experiments: LAT ([7]), Watershed ([8]) and Thresholding.

4.1 Watershed

Algorithm proposed in [8]. It takes a gray scale image and considers it as a topographic surface. A process of flooding is simulated on this surface; to avoid the merging of two or more floods coming from different basins, dams are built on the points where the waters meet. At the end, dams define the watershed.

4.2 LAT

Algorithm proposed by Kirby and Rosenfeld in [7]. It takes a grayscale or colour image as input and outputs a binary image representing the segmentation. For each pixel in the image, a threshold is calculated. If the pixel's value is below the threshold it is set to the background value, otherwise it assumes the foreground value; or viceversa.

4.3 Thresholding

The input is a gray level image and the output is a binary image representing the segmentation. Each pixel in the image is compared with a given threshold k which is a gray level. If the pixel's value is below the threshold k, the pixel p_i is set to, say, black in the output; otherwise it is set to white.

We made comparisons of these algorithms segmenting images representing truck containers and shew results in a previous work ([4]). As in this work, we made experiments with these algorithms on their own and also merging the results of applying more han one to the same image. Depending on the algorithm, it had to be applied several times to each image, varying its parameters in order to cover all possible situations.

Also, they had to be executed twice, once seeking for clear characters and again, seeking for dark characters. This was done this way, because segmentation algorithms were not provided with concrete information about each image (illumination, number of characters...).

A k-Nearest Neighbours classifier was trained using 654 real images of trucks what means a total of 9810 characters. We gathered 288 features representing gray levels from each normalized character (12×24 pixels). We applied PCA ([9]) to reduce the dimesionality of data, and at the end, only 60 features were used. When classifying objects with k-NN neighbours, we used $k = 3$. The contrast value was calculated experimentally and we tried to set it low enough as to not lose shady characters.

In order to extract conclusions about the performance of the process we labelled by hand all characters in the images. We made this by drawing the inclusion box of each character in the image and assigning it an alphanumeric label. An example is shown in figure 2. As a result of this labelling, we had the number of code characters, the coordinates of the inclusion boxes and the label of each code character of each image.

By applying the recognition process to each image we obtained a list of found objects. These objects correspond to connected regions with equal gray intensity levels, which had gone successfully through the different filters.

Experiments were made in a similar way as proposed in [4]; we considered the algorithm had done a hit if the inclusion box it had calculated and the man-

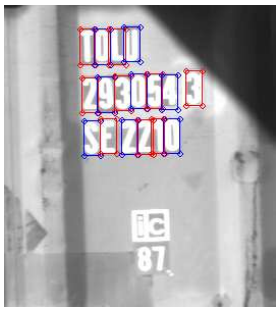


Figure 2: Inclusion boxes drawn by a human operator.

ually labelled inclusion box overlapped one on each other in a certain percentage and both labels matched.

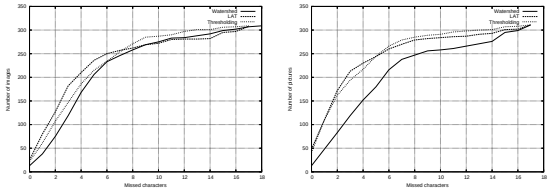
5 Results

Recalling figure 1, it may be seen that in the recognition process each step depends on the output of the previous one, so, errors accumulate from one step to another. It is easily seen, that the whole process will be influenced by the possible failures that may appear in each step.

In [4] we checked the performance of the segmentation algorithms. Now, we show results for the complete process. We made experiments with 309 images. The results are shown in figure 3a and in table 1, first row contains the number of images for each segmentation algorithm successfully recognized. Remaining rows show recognition results for the complete process depending on the segmentation algorithm used, are ordered according to the number of missed characters.

Missed characters	LAT	Wat.	Thr.
0	25	13	22
1	55	25	39
2	47	37	46
3	55	44	40
4	28	49	39
5	26	38	29
6	14	27	19
more than 6	59	76	75
Missed characters	LAT	Wat.	Thr.
0	48	13	41
1	62	35	69
2	62	35	52
3	41	36	32
4	17	33	23
5	14	28	28
6	16	36	21
more than 6	52	96	46

Table 1: Performance of the recognition process. Above, results with the first classifier. Results with the improved classifier are shown below.



(a) (b)

Figure 3: Cumulative plot of images according to the number of missed characters by the complete process. a) Results with the first classifier. b) With the second one.

We trained a new classifier with more instances in the hope of improving results. We improved the classifier with the misclassified objects of the first set of experiments. It was trained now with a corpus of 963 images. We repeated the experiments with a new set of 312 images. Results are shown in figure 3b and in table 1. By comparing results of both experiments, it may be seen that improving the classifier improves results.

We present a process for detecting the registration number of truck containers based on segmentation algorithms, k-NN classifiers and filters. We used images selected randomly from a large set of real images, and compared the performance of the recognition process against the results given by a human operator. Our effort was driven by the fact that we wanted to adjust the process, in such a way, that we did not miss any character (or the lowest possible amount) in the registration number. Our evaluation of the different solutions then penalties the lose of characters.

We made experiments with different classifiers and different segmentation algorithms. We concluded that the solution that better fits our needs would be taking together LAT and Thresholding to segment the images and keep on improving the classifier. Further efforts will focus on improving execution times by parallelizing the execution of both algorithms.

Missed characters	LAT+Thr.	LAT+Wat.	LAT+Thr.+Wat.
0	102	91	163
1	87	90	72
2	49	43	19
3	23	30	6
4	10	8	7
5	11	10	2
6	11	8	1
more than 6	108	32	40

Table 2: Performance of the merged algorithms. Amount of images according to the number of missed characters.

Improving the classifier is not the only way to improve results. As segmentation algorithms do also

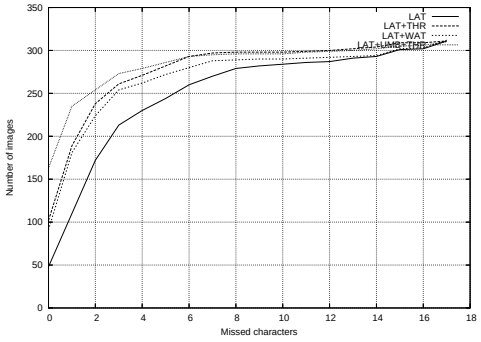


Figure 4: Results of the merged algorithms.

Mean time \ Alg.	LAT	Water.	Thres.
seconds	2.99	14.81	1.53
Mean time \ Alg.	L-T	L-W	L-W-T
seconds	4.36	16.56	18.33

Table 3: Mean time of execution of the algorithms. On the left, algorithms on their own. On the right, algorithms merged.

make failures, we decided to test what the gain in performance would be if we took together results from two or more algorithms applied to an image. We did this in [4] but now this merge is not done by filtering together all algorithms' results, because the fussion filter may join objects belonging to the result of different algorithms. We applied several algorithms to an image, filter their result separately, and join all results. Results are shown in table 2 and in figure 4. In table 3 we show the time consumption of the recognition process depending on the segmentation techniques used. The process reveals as very slow when we use the Watershed algorithm as segmentation technique; on the other hand, LAT and Thresholding working together improve results with low time cost. If, for instance, we can afford 3 characters missed, using LAT plus Thresholding nearly $83,65\%$ of images could be solved successfully.

6 Conclusion

We present a process for detecting the registration number of truck containers based on segmentation algorithms, k-NN classifiers and filters. We used images selected randomly from a set of real images, and compared the performance of the recognition process against the results given by a human operator. We wanted to adjust the process, in such a way, that we did not miss any character in the registration number. Our evaluation of the different solutions then penalties the lose of characters.

We made experiments with different segmentation algorithms. We concluded that the solution that better fits our needs would be taking together LAT and Thresholding to segment the images and keep on improving the classifier. Further efforts will focus on improving execution times by parallelizing the execution of both algorithms.

References:

[1] Salvador, I., Andreu, G., Pérez, A.: Detection of identifier codes in containers. Proc. SNRFAI-2001. Castellón, Spain. May de 2001. **1** (2001) 119–124

[2] Salvador, I., Andreu, G., Pérez, A.: Preprocessing and recognition of characters in container codes. ICPR2002, Canada, 2002 (2002)

[3] Woods, R.G.R.: Threshold selection using a minimal histogram entropy difference. Addison-Wesley (1993)

[4] Rosell, J., Pérez, A., Andreu, G.: Segmentation algorithms for extraction of identifier codes in containers. Int. Conf. (VISAPP-2006), Portugal (2006) 375–380

[5] Atienza, V., Rodas, A., Andreu, G., Pérez, A.: Optical flow-based segmentation of containers for automatic code recognition. Lecture Notes in Computer Science **3686** (2005) 636–645

[6] Soille, P.: Morphological image analysis: Principles and applications. Springer Verlag (1999)

[7] Kirby, R.L., Rosenfeld, A.: A note on the use of (gray level, local average gray level) space as an aid in threshold selection. IEEE Transactions on Systems, Man and Cybernetics SMC-9 (1979) 860–864

[8] Beucher, S., C-Lantuéjoul: Use of watersheds in contour detection. CCETT/INSA/IRISA IRISA Report n. 132, Rennes, France (1979) 2.1–2.12

[9] Fukunaga, K.: Statistical Pattern Recognition. Second edition edn. Academic Press (1990)