" i " - A novel algorithm for Optical Character Recognition (OCR)

Sushruth Shastry, Gunasheela G, Thejus Dutt, Vinay D S and Sudhir Rao Rupanagudi
WorldServe Education,
Bangalore, India
sudhir@worldserve.in

***Abstract*—Computer vision, artificial intelligence and pattern recognition have been important areas of research for a while in the history of electronics and image processing. Optical character recognition (OCR) is one of the main aspects of computer vision and has evolved greatly since its inception. OCR is a method in which readable characters are recognized from optical data obtained digitally. Many methodologies and algorithms have been developed for this purpose using different approaches. Here we present one such approach for OCR named " i ". Amongst all other OCR systems available, " i " aims at a high speed, simple, font independent and size independent OCR system based on a unique segment extraction technique. This algorithm can be used as a kernel for single alphabet detection within a complete OCR solution system without the need for any complex mathematical operations. The highlight of this methodology is that, it does not use any libraries or databases of image matrices to recognize alphabets, but it has a unique algorithm to recognize alphabets instead. This algorithm has been implemented in MATLAB 7.14.0.739 build R2012a on a test set of 500 images of text and an accuracy of 100% for three font families namely *Arial*, *Times New Roman* and *Courier New* has been obtained.**

Keywords- Feature extraction, image processing based OCR, OCR, segment extraction, segment storage, segment profiling

I. INTRODUCTION

Optical Character Recognition (OCR) is a process of converting alphabets in images to computer readable coded text. Early implementations of conversions from characters in one domain to another can be traced back to 1913 [1]. Devices called Optophones were introduced in 1913 to convert Braille symbols on paper to audio tones for faster reading in order to hear Braille code. Later on, technologies were developed for actual OCR systems which recognize alphabets in an image. Fonts like OCR - A (Defined by ANSI INCITS 17-1981 (R2002)) were developed to make the process of OCR easy. Today, technology behind OCR has evolved hugely since its inception and many possibilities exist including font recognition [2], image to document conversion, etc., in OCR online applications as well as software packages [3].

Present day OCR systems are usually based on certain types of OCRs. Structural type, feature type and neural network type of OCRs are some of the most common used [4]. Some implementations even use State Vector Machines (SVM) along with wavelets as the input variables for the OCR process [5]. The above mentioned methods use extensive mathematical operations and involve several calculations to deduce the wavelets. These in turn use matrix multiplications, summations and the SVM essentially requires time to take different input-output pairs and calculate the relation between them. Approaches using artificial neural networks also use training mechanism for OCR [6]. These approaches are equally exhausting in terms of mathematical processing as the previous technique discussed above. There are also systems which do not need training and memory based training or recognition. Some use fuzzy logic and histogram type area-weight detection of the areas of alphabet [7]. The main disadvantage of these methods would be the complex computations and the processing time taken. Yet another methodology used for OCR is feature recognition based OCR. In feature recognition type [8] OCRs, different features are extracted from an alphabet present in the input image. This is a highly advantageous approach in terms of memory utilization and computations, since only a certain set of features of the character are sufficient to identify the letter.

In this paper we propose an algorithm which works on basis of the feature recognition based OCR. The proposed algorithm extracts details about the lines or curves which the alphabet is made of, and then based on the extracted details, it takes a decision about the alphabet in the image. Here we use no databases or libraries to compare with, but only logic based on patterns and shapes which decides the alphabet. What is really interesting about the algorithm is that it is font independent while having no database, training or SVM mechanisms. The algorithm is efficient in the terms of small footprint and better efficiency.

The next section describes the proposed algorithm in detail. Section III concentrates on the results obtained and Section IV deals with the conclusion and what future lies in store, in the field of OCR.

II. THE PROPOSED ALGORITHM

The proposed algorithm for optical character recognition has been explained in detail in this section.

A. Overview of the algorithm

The algorithm is represented in a block diagram as shown in Figure 1.

978-1-4673-5090-7/13/$31.00 ©2013 IEEE

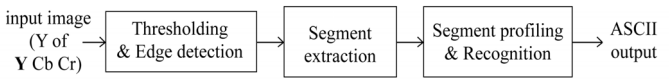


Figure 1. Main block diagram.

First, an image is acquired through any of the standard image acquisition techniques. The input image is assumed to be in the *Y Cb Cr* color format. The algorithm works on the *Y* part of the input image, which is a gray level image.

Using an appropriate thresholding algorithm, the gray level image is then thresholded to obtain a binary image which quantizes alphabets and background to black and white colors respectively.

The obtained binary image is then passed on to a specific edge detection process. The edge detection algorithm is performed such that only the right sided edges of each alphabet are obtained and the other edges are eliminated.

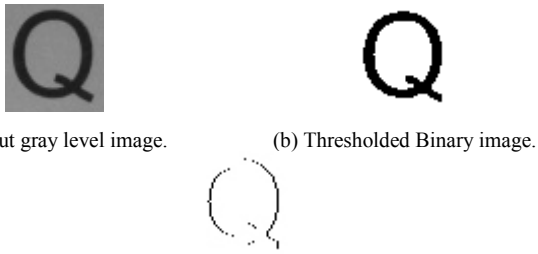
After edge detection, the image is then segmented and feature extraction is performed. A segment is defined as a continuous path of black pixels in the edge detected image, for this algorithm. In this step, different details of the segments, which are required for further processing, are stored. The next step is to profile stored line segments. Profiling of segments is the process of categorizing them into different types of segments such as short, long, line or curve, etc. The qualities of a segment such as above, which are required for this algorithm, are defined later on in this section.

Every alphabet in the present constraint set is made up of combination of segments of particular profiles. So, each of the alphabets is given a feature vector which contains a set of flags (bits). Each of the flags corresponds to one of the segment profiles which the alphabet is made of. The segments extracted and profiled are used to update these feature vectors of each alphabet. When the complete alphabet is processed, the feature vector which contains all high bits represents the recognized alphabet which that feature vector belongs to. Finally, ASCII equivalent of the recognized alphabet is the output.

The next few sections describe the above steps in detail.

B. Thresholding and Edge detection

The Y component of the input image is thresholded in this block. The results of thresholding and edge detection for the alphabet Q are shown in Figure 2. Conventional thresholding with a threshold value of 64 is used for this particular input image. The threshold factor was obtained after experimentation on over 500 images of alphabets.



(a) Input gray level image. (b) Thresholded Binary image.

(c) Edge detected image.

Figure 2. Output of thresholding block.

C. Segment extraction

1) Segments and segment storage: Segments are the features on which the proposed algorithm works on. Some of the general terms we use while discussing about segments are start pixel, continuation pixel and end pixel. Start pixel of a segment is the pixel starting from which a path of black pixels, each of which are one of the lower three 8-neighbours to the one above, exists. This path of black pixels consists of continuation pixels and ends at a pixel which does not have any black pixel below it in the above said manner; the end pixel.

A set of details corresponding to each segment are stored in a particular format. This format is given in Figure 3. It is to be noted that each segment is given an ID. This ID is then copied into each pixel memory location, which belongs to that segment in this process. This ID helps in addressing the memory space reserved for that particular segment's data.

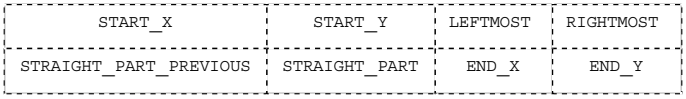


Figure 3. Segment data format

The explanation of the fields of which the segment data format is made of, are given in Table I.

TABLE I. DATA FIELDS IN THE SEGMENT DATA FORMAT

Field	Description
START_X	The *x* co-ordinate (value) of the start pixel.
START_Y	The *y* co-ordinate of the end pixel.
LEFTMOST	The leftmost horizontal co-ordinate encountered in that segment.
RIGHTMOST	The rightmost horizontal co-ordinate encountered in that segment.
STRAIGHT_PART_ PREVIOUS	The length of longest straight trail of black pixels encountered in that line segment.
STRAIGHT_PART	The length of the present straight trail of black pixels which may or may not proceed further to be a straight trail.
END_X	This is the *x* co-ordinate of the end pixel.

Field	Description
END_Y	This is the *y* co-ordinate of the end pixel.

Each of these fields for every segment found in the image is updated according to the pattern of pixels surrounding the present pixel being processed.

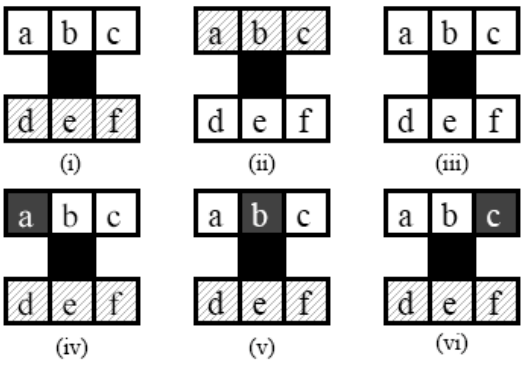
2) Segment Updating: Use of the word 'updating' indicates different set of actions for different fields. Updating procedure for LEFTMOST, RIGHTMOST and STRAIGHT_PART fields is given in Table II. Updating of the remaining fields has been explained in the subsequent subsection.

TABLE II. UPDATING PROCEDURES FOR CERTAIN DATA FIELDS

Field	Updating procedure
LEFTMOST	The present horizontal co-ordinate is checked with the stored LEFTMOST value of that segment. If it is less than the stored value, the current horizontal co-ordinate is copied to LEFTMOST value of that segment. Nothing is done otherwise.
RIGHTMOST	The present horizontal co-ordinate is checked with the stored RIGHTMOST value of that line segment. If it is greater than the stored value, the current horizontal co-ordinate is copied to RIGHTMOST value of that line segment. Nothing is done otherwise.
STRAIGHT_PART	The STRAIGHT_PART value of a segment is checked with the STRAIGHT_PART_PREVIOUS value of the same segment and whichever is greater is stored in STRAIGHT_PART_PREVIOUS field of that line segment's data.

3) Segment extraction process: Each pixel in the output of thresholding block is examined in this process. The algorithm is set off by starting to process each pixel from the top left of the image. While iterating through the image, different patterns of neighbor pixels are possible. All such possible patterns are given in Figure 4. Pixels to the right and left of the present pixel being processed are omitted as they will surely be white. All these cases are applicable for a black pixel encountered while iterating through the image and no process is carried out for a pixel under observation if it is white. For each of these cases, a set of operations are executed.

The top three pixels of the pixel being processed are named *a*, *b* & *c* from left to right. Similarly the bottom three pixels are names *d*, *e* & *f*.



Legend:

Pixel with any gray level except the case where all the three in a single row being white.

Processed pixel.

Black pixel (being processed currently).

White pixel.

Figure 4. Different neighbor pixel patterns

The different cases and operations executed for each case involve updating of certain fields in the data format for each segment. These have been explained below:

Case (i): *a*, *b* & *c* are all white irrespective of the state of *d*, *e* & *f* except the case where *d*, *e* & *f* are all white. In this case, the segment is starting and its start pixel co-ordinates are stored in a new storage block.

Case (ii): *d*, *e* & *f* are all white irrespective of the state of *a*, *b* & *c* except the case where *a*, *b* & *c* are all white. In this case, the line segment is ending and its end co-ordinates are stored in the respective storage block of that segment.

Case (iii): The case where *a*, *b*, *c*, *d*, *e*, & *f* are all white. This means that the pixel encountered is a shot noise. This is not considered and the process continues.

Case (iv): In this case, top pixel *a* has the ID of the line segment and other two top pixels are white. Here, the STRAIGHT_PART_PREVIOUS and the RIGHTMOST values of that line segment are updated.

Case (v): In this case, top pixel *b* has the ID of the line segment and other two top pixels are white. Here, only the STRAIGHT_PART_PREVIOUS value of that segment is updated.

Case (vi): In this case, top pixel *c* has the ID of the line segment and other two top pixels are white. Here, the STRAIGHT_PART_PREVIOUS and the LEFTMOST values of that line segment are updated.

In each of the cases - (iv), (v) and (vi), the state of pixels *d*, *e* & *f* are not considered. Also, additionally in cases (iv) and

(vi), STRAIGHT_PART field of that segment's data is made zero.

By the end of this process, we obtain the data of all segments that the alphabet in the image is made of. Now the segments extracted are to be profiled.

D. Profiling and recognition

1) Segment properties: As soon as the END_X and END_Y of a segment are updated, we profile the segment and update the corresponding bits of feature vectors of all the alphabets which are related to that segment's profile. Different properties of a segment are calculated and are used to profile the segment. These properties are computed as given here:

Height (η): It is the difference between START_Y value and END_Y value of that line segment.

$$\eta = START\_Y \sim END\_Y. \quad (1)$$

Gap (γ): It is the difference between START_X value and END_X value of that line segment.

$$\gamma = START\_X \sim END\_X. \quad (2)$$

Width (ω): It is the difference between LEFTMOST value and RIGHTMOST value of that line segment.

$$\omega = LEFTMOST \sim RIGHTMOST. \quad (3)$$

Left distance (λ_l): It is the difference between LEFTMOST value and START_X value of that line segment.

$$\lambda_l = LEFTMOST \sim START\_X. \quad (4)$$

Right distance (λ_r): It is the difference between RIGHTMOST value and START_X value of that line segment.

$$\lambda_r = RIGHTMOST \sim START\_X. \quad (5)$$

Using these properties, the following characteristics of a segment are computed. These are the profile metrics. It is to be noted that the terms 'alphabet height' and 'alphabet width' used below correspond to height and width of the image if the image contains only one alphabet as shown in Figure 2.

All the characteristics defined for segments are enlisted below.

Zones: Based on the location of start pixel of a segment, the segment is characterized into four zones. They are:

- **Top / bottom zone:** If the START_Y value of the line segment is less than 50% of the alphabet height, that segment is categorized to be present in the top zone. It is categorized to be present in bottom zone otherwise.
- **Left / right zone:** If the START_X value of the line segment is less than 50% of the alphabet width, that segment is categorized to be present in the left zone. It is categorized to be present in right zone otherwise.

Short / long segment: If the η of segment is greater than 75% of the alphabet height, the segment categorized as short. It is categorized as a long segment otherwise.

Straight / slant: If γ of the segment is greater than 25% of height, then it is categorized as a straight segment. It is categorized as a slant segment otherwise. If a segment is categorized as a slant segment, it can only be a line segment. In other words, curve segments were not found to be slant in the preliminary analysis.

Line / curve: If STRAIGHT_PART_PREVIOUS value of the segment is greater than 75% of η, it is categorized as a line segment, or else it is categorized as a curve segment.

Right / left sided line: If START_X value of a segment is greater than END_X, then it is categorized as a left sided line (like '\'). Otherwise, as a right sided line (like '/').

Right / left sided curve: If the λ_l value of line segment is greater than its λ_r value, then it is categorized as a left sided curve, or else as a right sided curve.

Using these profiles to define the type of a segment, the component segments' profiles of each alphabet were defined. For example, the alphabet Q mandatorily has three types of segments irrespective of which of the standard fonts it belongs to. This is shown in Table III. All the 26 alphabets are defined in this way. All of the alphabets have the same number of flags in their feature vector as the number of segments mandatorily required to represent them.

TABLE III. SEGMENT PROFILES FOR THE ALPHABET Q.

Sl. No.	Profile
1	*Left top* zoned, *long left-sided curve* segment.
2	*Right top* zoned *long right-sided curve* segment.
3	*Right/left bottom* zoned *left-sided line* segment.

All definitions of alphabets in this manner are similar to how we, as humans, define and identify alphabets. These definitions are universal; even patterns which look like alphabets can be recognized using these. Also, we can define such characteristics for any language and its characters possible.

Now, the feature vector that contains all high bits represents the alphabet in the image. ASCII equivalent of the alphabet can be presented as the output once the alphabet is recognized.

III. Results

The developed algorithm was designed and implemented on MATLAB 7.14.0.739 build R2012a with a thresholding module and a simple alphabet segmentation module to locate alphabets before recognition using the aforementioned algorithm. It was tested on 500 different computer generated images of the three major fonts. For clarity one of the test images having four different test fonts has been shown in Figure 5a. The Figure 5b shows the thresholded image which was obtained by simple thresholding with a value of 128, assuming a total of 256 gray levels in the image. As mentioned in the previous section, an edge detection algorithm was used, which compares two adjacent pixels of the thresholded image and decides if the current pixel is part of the right edge of the alphabet or not. This result is shown in Figure 5c. An alphabet segmentation module was also devised to tell the exact location of the starting and ending co-ordinates of an alphabet. This module starts analysis from the top left corner of the image and works its way through the bottom right corner. First, it detects the starting and ending vertical co-ordinates of lines made of alphabets. Once the vertical start of the line is detected, this module starts keeping track of positions of black pixels in the line. This is achieved by updating a vector which has a single row and the same number of columns as the input image's horizontal width in pixels. Whenever a black pixel is found, the value in that corresponding horizontal position is made high in that vector. This is continued until the end of the line is detected. The same is done for all the detected lines in the alphabet. A graph of such a vector for the selected input image is shown in Figure 5d. Utilizing the algorithm mentioned in the previous section, the results were obtained in MATLAB and an accuracy of 100% was achieved for all 3 font families – Arial, Times New Roman and Courier New.

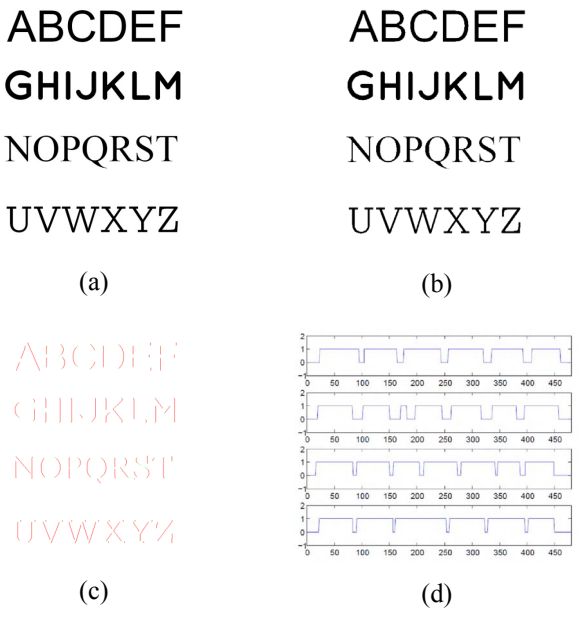


Figure 5. Results of the novel OCR algorithm (a) Test image used (b) Output obtained from thresholding (c) Edge detection output (d) Waveforms indicating start and end of line

IV. Conclusion

The motivation for the development of this algorithm was the simple fact that English alphabets are fixed glyphs and they shall not be changed ever. Due to this fact, usage of artificial neural networks and vector based data training provide almost accurate results, but these are performing a lot of redundant work. And also, most of the OCR processes today involves images from high resolution scanners and cameras. OCR technologies now can make use of this advancement in technology and consider techniques which were abandoned (the latest OCR innovation without shape training dates back to the year 2000 [9]) due to the lack of present day imaging technology. Our algorithm is one such approach. This algorithm has the advantage of speed, power, memory and area, since it does not include any training or learning mechanisms and also because of lack of image database which some OCR techniques require [10]. Also, this algorithm is the first, in being a multiple font OCR and also a no training type OCR.

To conclude, the developed algorithm can readily be used as a kernel within a complete OCR solution to recognize each alphabet after the segmentation and such adjustment operations to align the alphabet horizontally. The algorithm also gives us an accuracy of 100% on the current test set of alphabets for most of the fonts in all the three above mentioned font families currently. Future versions of this algorithm can be expected to extract more types of features from the input alphabet image to improve accuracy and to provide support for other alphabet sets.

References

[1] D'Albe, "On a Type-Reading Optophone", Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character, Volume 90, Number 619, July 1914.

[2] Ramanathan. R. et al., "A Novel Technique for English Font Recognition Using Support Vector Machines", in Advances in Recent Technologies in Communication and Computing, Kottayam, Kerala, 2009, pp. 766 - 769.

[3] L Priit. (2011, November 1). How to extract text from images: a comparison of 10 free OCR tools [online]. Available: http://www.freewaregenius.com/2011/11/01/how-to-extract-text-from-images-a-comparison-of-free-ocr-tools/

[4] Line Eikvil, "Optical Character Recognition", Norsk Regnesentral , Oslo, Norway, Rep. 876, 1993.

[5] Yang Guang, "License Plate Character Recognition Based on Wavelet Kernel LS-SVM", inComputer Research and Development (ICCRD) 3rd International Conference, Shanghai, 2011, pp. 222 - 226.

[6] M Usman Raza, et al., "Text Extraction Using Artificial Neural Networks", in Networked Computing and Advanced Information Management (NCM) 7th International Conference,, Gyeongju, North Gyeongsang, 2011, pp. 134 - 137.

[7] Fonseca, J.M., et al., "Optical Character Recognition Using Automatically Generated Fuzzy Classifiers", in Eighth International Conference on Fuzzy Systems and Knowledge Discovery, Shanghai, 2011, pp. 448 - 452.

[8] Kumar, M., et al., "k - Nearest Neighbor Based Offline Handwritten Gurmukhi Character Recognition", in International Conference on Image Information Processing, Himachal Pradesh, 2011, pp. 1 - 4.

[9] Tin Kam Ho and Nagy, G., "OCR with no shape training", in Proc. of Pattern Recognition 15th International Conference, Barcelona, 2000, pp. 27 - 30 vol.4.

[10] Mori, S., et al., "Historical review of OCR research and development", in Proc. of the IEEE, 1992, pp.1029 – 1058.