A STANDALONE OCR SYSTEM FOR MOBILE CAMERAPHONES

Mikael Laine
University of Turku, Department of Information Technology
and Turku Centre for Computer Science (TUCS)
FIN-20014 Turku
Finland
mikael.laine@it.utu.fi

Olli S. Nevalainen
Department of Information Technology, University of Turku
and Turku Centre for Computer Science (TUCS)
FIN-20014 Turku
Finland
olli.nevalainen@it.utu.fi

ABSTRACT

In optical character recognition (OCR), visible characters appearing as images (i.e. on paper) are recognized as symbolic characters and stored in a computer's memory or similar device. The purpose of this work is to find whether current mobile cameraphones are able to run OCR software without relying on dedicated hardware or facilities provided by the network. This article describes a simple OCR system that was implemented in Symbian C++ to be run on a stock Nokia 6630 cameraphone. The system is limited to recognizing English capital letters printed in black, against a white background. The opportunities and hardships related to bringing OCR to run on mobile platforms having image capturing capability are also discussed in more general terms.

Keywords: optical character recognition, pattern recognition, segmentation, mobile devices, Symbian C++

I. INTRODUCTION

Modern high-end mobile cameraphones are capable of capturing color images whose sizes are at least 1280×960 pixels, which is usually quite adequate for performing Optical Character Recognition. Having OCR software available for a camera phone has obvious benefits: it can be used to make quick translations of foreign words, to store notes space-efficiently from lecture slides or notice boards and, as technology improves, it can become an invaluable tool for visually challenged people.

In this work a standalone OCR system is presented, that was implemented in Symbian C++ and tested on a Nokia mobile phone. The work is an attempt to evaluate the feasibility of OCR on a mobile phone, without relying on external resources such as a server computer or specialized image processing hardware.

The rest of this work is organized as follows. Previous work on OCR systems with mobile phones are shortly discussed in Section II. The design of the proposed OCR system is described in Section III. Finally, experiments with the new system are summarized in section IV.

II. PREVIOUS WORK

OCR involves a number of demanding image-processing steps and other complicated algorithms [4, 5]. It is obviously for this reason, that an entire OCR system built to run on a mobile phone alone is rather laborous to construct. A working system of this kind, with all the segmentation phases, has not been documented in the literature. Hitachi has, however, developed an OCR software for mobile phones that sends images to an external server that performs the actual OCR task [1]. Pantech-Curitel has also constructed a mobile phone with an OCR function for recognizing text from previously captured JPEG-images [2]. The algorithms of that system have not been published and the device is currently only marketed in South Korea. There are some other OCR implementations for mobile phones, such as [3], but they are often poorly documented, very simple and only focus on the feature extraction and classification phases.

III. THE OCR SYSTEM

An optical character recognition system can be viewed as a combination of several subsystems, each specialized to solving one of several subproblems. After the system has received an input image from a camera (or some other device), there are several preparative stages the data has to undergo before the actual recognition of characters can take place.

The implemented system has the following subsystems, listed in chronological order:

1. image capturing;
2. preprocessing;
3. segmentation;
4. feature extraction;
5. character recognition.

A. Image Capturing and Preprocessing

The images captured on the test platform are initially 24bpp color (RGB) images, but are converted to grayscale with 256 levels of brightness. This is done by simple addition of the tristimulus (RGB) values and division by three.

The images are assumed to contain one centered block of text, with black capitalized letters whose skew angle is the same.

Binarization is the only preprocessing operation used in the proposed system, although morphological opening would have improved segmentation results by increasing the gaps between characters. Global binarization was chosen because it is fast and seemed to work adequately in most cases. Niblack's binarization method [6] was also tried but despite its good results, it was not included in the system because it was too demanding computationally.

B. Segmentation

Based on the assumptions made about the input image (black capitalized text printed on a white background), the segmentation process can be completed using *skew angle detection* and *X-Y-Tree decomposing*.

1-4244-0330-8/06/$20.00©2006 IEEE

Authorized licensed use limited to: Reprints Desk. Downloaded on July 23,2025 at 20:35:56 UTC from IEEE Xplore. Restrictions apply.

1) Skew Angle Detection

The skew angle (see Fig. 1) is obtained by running two algorithms that basically rely on the same principle of hypothesis-testing [5]. The first algorithm finds an approximation of the skew angle using a fast algorithm. The second algorithm is slower, but only a few angles are tested with it, since the approximate angle is already known.

The first algorithm determines an approximate skew angle by finding the direction $\alpha \in (-90°, 90°)$ that has the greatest amount of consecutive zero pixels. During testing, only 21 directions were tried from $-57°$ to $57°$, which means that the error of the result could be as great as $5.7°$. Finding the best direction is done for all locations on a vertical line in the middle of the image (see Fig. 1). It is assumed that the text is centered. After traversing every direction at every vertical location, a number of best directions are chosen. Quicksort is used to sort the directions according to the number of consecutive zero pixels and thus obtain the ten best directions. The average direction of these is the approximation α used in the next phase.

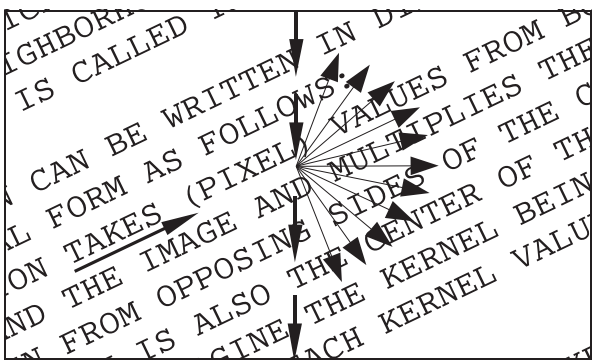

Figure 1: Finding the skew approximation by searching for the direction that has the greatest number of consecutive zero pixels. The actual skew angle is depicted on the left.

The accurate skew angle is obtained using the method of projections and alignment criterions, as in [5], pp. 19. The basic idea is to find the skew angle that maximizes a given alignment criterion. The projections are performed on a rectangular area, whose width and height (w, h) are half of those of the image and which is located in the middle of the original image. Reducing the size of the rectangular area reduces the computational effort and also reduces the effect of possible spurious features on the edges of the image (shadows, etc.)

The projections are not done for all directions $(-90°, 90°)$, but only for those few directions that are close to the approximate angle α, found in the first step. During testing, the final angle β is searched from among angles $\beta \in [\alpha - 11.46°, \alpha + 11.46°]$ with an increment of $0.57°$, thus allowing the error to be as low as $0.29°$.

The projection P_γ in the direction γ is calculated at the center (x_o, y_o) of the original image and is obtained in the following way. The projection is a vector holding the sums of the pixel values taken from the direction γ starting from each location $(x_o, y_o + y)$, $y \in [-\lfloor \frac{h}{2} \rfloor, \lfloor \frac{h}{2} \rfloor]$.

$$P_\gamma(y) = \sum_{d=-w/2}^{w/2} I(x_o + d \cdot \cos(\gamma), y_o + y + d \cdot \sin(\gamma)) \quad (1)$$

The alignment criterion $A(\gamma)$ for choosing the best projection and angle, is the number of projection values $P_\gamma(y)$ below a certain threshold. During testing, the value of 1 was used for the threshold, which means that $A(\gamma)$ is the number of zero values in the projection vector. In other words, the alignment criterion was the number of empty lines between characters.

$$A(\gamma) = \sum_{y=-\lfloor \frac{h}{2} \rfloor}^{\lfloor \frac{h}{2} \rfloor} a(P_\gamma(y)),$$
$$\text{where } a(P_\gamma(y)) = \begin{cases} 1 & \text{if } P_\gamma(y) \geq 1 \\ 0 & \text{if } P_\gamma(y) < 1 \end{cases} \quad (2)$$

The angle β that gives the highest alignment criterion value is then chosen to be the final skew angle:

$$\beta = \{\beta | A(\beta) \geq A(\gamma), \forall \gamma \in [\alpha - 11.46°, \alpha + 11.46°]\}, \quad (3)$$

where α is the approximate angle found by the first skew angle algorithm.

2) Character Extraction

X-Y-Tree decomposition is a simple segmentation algorithm that works well for images whose content is mostly text lines ([5] pp. 35). The method uses projections to separate text lines and individual characters, see Fig. 2.

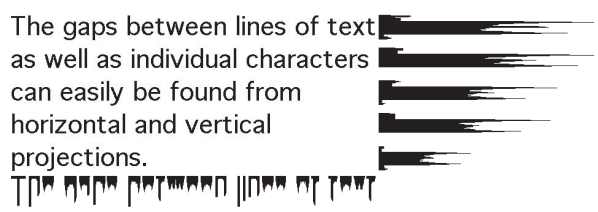

Figure 2: Horizontal projection of a page and vertical projection of the first line.

The X-Y-Tree decomposition method is used in the implemented system. The prior assumption is made, that the document consists of only one block of text-lines. Instead of skew-correcting the image first, the needed projections are done on the whole image in a similar fashion to formula (1), by moving in increments of $(\cos(\beta), \sin(\beta))$, where β is the skew angle.

For finding the text lines, the following procedure was conceived. After projecting the whole image in the direction of the skew angle, *baselines* of text lines are found along with their so-called *x-height*, *ascent* and *descent* measures. The baseline is an imaginary line on which the characters rest. Parts of some characters may span below the baseline. X-height, ascent and descent are length measures which indicate how high or low the

Authorized licensed use limited to: Reprints Desk. Downloaded on July 23,2025 at 20:35:56 UTC from IEEE Xplore. Restrictions apply.

characters may span up and down from the baseline (see Fig. 3). This procedure allows lower case letters to be correctly extracted, although the system was only tested using capital letters.

These variables are obtained in the following simple way that was conceived for the purposes of this system. For the projection, two averages are calculated: $a1$ - the average of all values in the projection and $a2$ - the average of all values below $a1$. Vertical locations are then classified as being G, N or B locations. Locations of type G are gap locations (between the text lines). Locations of type N are near the actual *body* (the thickest part) of the characters (in the ascent or descent part). Locations of type B belong to the actual body of characters (from baseline to the top of x-height). The classification for all projection locations l with a value of $v(l)$ is done as follows:

condition	**class**
$v(l) \leq a2$	G
$v(l) > a2$ AND $v(l) < a1$	N
$v(l) \geq a1$	B

The projection is traversed from top to bottom to find the locations of baselines and the three length measures.

The baseline is defined as being the next location after a B location that is a G or an N location. In other words: the first location under the main body of characters is the baseline.

X-height is the distance from the baseline to where a row of B locations starts (see Fig. 3).

Figure 3: Two lines of text and their projections (on the left). Also illustrated are the length measures x-height, ascent and descent. In the proposed OCR system, these parameters are found by traversing the projection from top to bottom.

Ascent is the length of the first row of N or B class locations before a baseline. If no N class locations exist (which is always the case in capitalized text), ascent will be the same as x-height.

Descent is similarly the length of the first row of N class locations after a baseline. It can also be zero.

Having found the skew angle and the vertical position of the text lines, the actual characters can be located by traversing the text lines (in the direction $(\cos(\beta), \sin(\beta))$. At this point, the characters are treated as connected components and their separation from each other is done by checking if there are touching pixels in between. If the number of touching pixels is zero (see Fig. 4), the starting location or ending of a character is recorded.

Each character is extracted into a small rectangular raster whose width and height is $ascent + descent$. At each starting location, the algorithm starts to "record" pixels into the character raster, until all pixels of a connected component are recorded. After the termination of this process, the rest of the raster is filled with zeros.

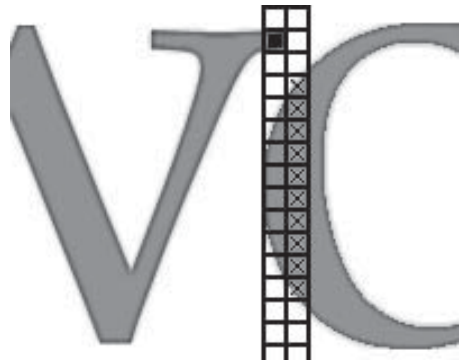

Figure 4: Distinguishing two characters from each other. The all-black square is a pixel that belongs to the connected component that comprises the current character. Since none of the pixels belonging to the next character touches the current character, an ending location is set here.

C. *Feature Extraction Using the Centroid-to-Boundary Distances*

In this work, the centroid-to-boundary distance features are used to quickly obtain a greatly simplified feature vector from a character image. We have not seen this method applied to OCR in any previous work although it is well-known in pattern recognition [8, 9]. In this method, a 1D representation is created from a binarized image of a character by measuring distances in different directions from the centroid of a character (see Fig. 6 and pp. 12 in [12]). The centroid (cd) of a character image is defined as the average location of black pixels according to $cd = \frac{1}{n}\sum_{i=0}^{n} p_i$, where $p_i = (p_{xi}, p_{yi})$ is the location of black pixel i in the character and n is the total number of black pixels. The centroid-to-boundary distances are then acquired by determining the outer boundary of the character in every direction $\alpha_i = 0 + i\frac{2\pi}{res}$ where res is the resolution that tells the number of directions to be scanned (see Fig. 5). In the implemented system, the value of 58 was used for the resolution. This value was chosen based on tests done previously [12] that showed that the errors do not significantly decrease if the resolution is higher than 48.

Having obtained the centroid-to-boundary feature vector of an unknown character image, the system classifies it by simply comparing it to the stored feature vectors representing all the used characters (see pp. 12 in [12]). Manhattan-distance is used as a dissimilarity measure mainly because it is very fast to calculate and worked adequately well.

The strength of the centroid-to-boundary distance method lies in that it allows rotation and size invariant recognition to be performed very easily. Rotation invariance can be implemented by shifting distance vectors (see Fig. 6). Size invariance can be attained by using a suitable coefficient C, such as

Authorized licensed use limited to: Reprints Desk. Downloaded on July 23,2025 at 20:35:56 UTC from IEEE Xplore. Restrictions apply.

```
GETDISTANCES(Point centroid)
  1. Array d[1..res]
  2. i ← 1
  3. for α ← 0 to 2π step 2π/res
  4.    inc ← (cos(α), sin(α))
  5.    len ← char_height
  6.    cloc ← centroid + len · inc
  7.    while len ≥ 0 and
        the image location cloc is white
  8.       len ← len − 1
  9.       cloc ← centroid + len · inc
  10.   endwhile
  11.   d[i] ← len
  12.   i ← i + 1
  13. end for
  14. return d
```

Figure 5: The algorithm for measuring centroid-to-boundary distances. It takes the centroid as its parameter and returns a vector containing centroid-to-boundary distances in all directions. The first black pixel is found by traversing each direction towards the centroid.

Figure 6: An example of extracting the centroid-to-boundary distances for the character 'R'.

the quotient of the average height of letters in two sets of letters (input and library) or the quotient of the average distances of individual feature vectors. The feature vectors $\bar{a}$ and $\bar{b}$ would then be compared by multiplying the other by C. Alternatively a length-invariant similarity measure could be used, such as the inner product. In the implemented system, the average distances were used as a coefficient.

Finally, the actual classification of an input character is done simply by comparing it to each stored character by calculating the difference, as described above. The class with the lowest difference is then chosen.

IV. Experimental Results

The OCR system of Section III. was implemented in C++ on a stock Nokia 6630 cameraphone, which runs Symbian OS and conforms to the Series 60 specification. It has an ARM (ARM 5 series) processor running at 220 MHz [11] and an integrated camera with a resolution of (1280×960) and bit depth of 24. Figure 7 shows a test image taken with it.

The OCR system was first tested and debugged on a desktop system, after which it was ported to Symbian C++ so that the entire operation could be performed on the mobile device. Fig. 8 shows two views from the OCRBench application that was written.

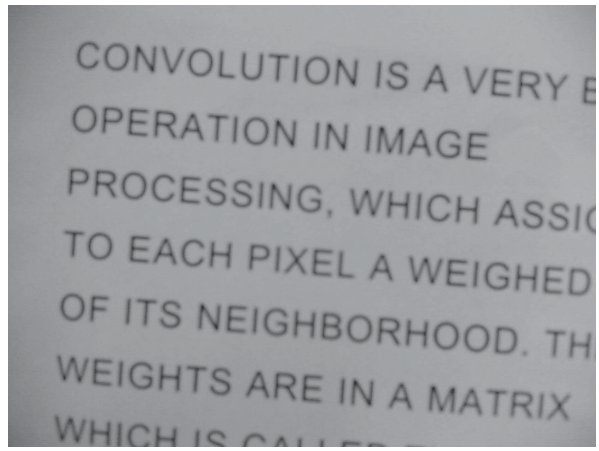

Figure 7: An image taken with the Nokia 6630 mobile camera phone from a distance of about 12 cm from the paper surface.

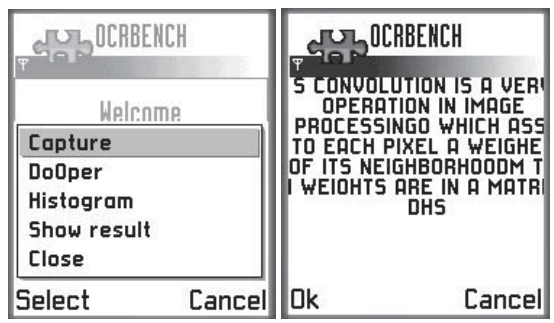

Figure 8: The OCRBench application. Menu (left) and the OCR result (right) for the image in Fig. 7.

The system was trained by feeding character images generated with a JAVA program to an implementation of the centroid-to-boundary distance routine that extracted the distances. The distances were then stored for later use. Only capital letters of the english alphabet in the Arial font were used.

After capturing the image (Fig. 7) and storing it in a suitable format for the OCR routines, the actual operation (binarization, skew angle calculation, X-Y-Tree decomposition, centroid-to-boundary-distance calculation and classification) took about 40 seconds , which is a very long time. Most of the time was consumed in determining the correct skew angle. Table 1 gives an idea of how much time each operation required. The table represents the observed running times of each routine on the Nokia 6630 phone during one test run. The last operation 'findChars' is the actual classifying stage, where features are extracted using the algorithm in Fig. 5, and comparisons are made. It must be noted, that the running times do vary slightly depending on the number of text lines.

Fig. 8 depicts an OCR result for the test image. It has several recognition errors, but can be recognized as the same text as in Fig. 7. The first letter 'S' in the result comes from the dark area on the left border of the image. This is due to poor binarization, which can not be helped without increasing the running-time

Authorized licensed use limited to: Reprints Desk. Downloaded on July 23,2025 at 20:35:56 UTC from IEEE Xplore. Restrictions apply.

Table 1: Observed relative running times for all subroutines during one test case.

Operation	**Processing time (% of total)**
binarization	1.9 %
findSkewApproximation	34.2%
findSkew	55%
findBaselines	3.1%
findChars	5.8%

of the algorithm significantly. Niblack's binarization with a window size of (100×100) would have taken an additional 2 minutes and 40 seconds to complete although the result would have been better especially at the edges.

Another obvious error in the test image is the comma that was recognized as an 'M' at the end of "NEIGHBORHOODM". This is due to the way in which size invariance was obtained (only the shape, not the size, of characters was taken into account) and could be helped by using a different coefficient.

Nevertheless, the results were encouraging in that binarization, segmentation and OCR could be done on a standard camera phone with just one processor. Some of the recognition errors could also be fixed by using the standard T9 predictive text engine (present on almost every phone) as a contextual processor.

V. Conclusions

Mobile devices are maturing rapidly and already the image capturing and processing capabilities of camera phones allow for limited OCR software to be run on them. It must be noted however, that software running on the CPU of a mobile phone is, and will continue for many years to be, severely constrained by low computational strength. Even this simple OCR system took over half a minute to complete, which is an unacceptably long time thinking of any practical application.

This seems to imply that OCR on mobile devices may not be possible unless a specialized physical chip (or several chips), capable of performing several image processing operations in parallel, is used.

If the average consumer, however, is to be the user of OCR software (text translation or other products), the software would have to run on almost any mainstream device. At the time of writing, some mobile phones have built-in (additional) hardware for playing mp3 (MPEG Layer-3) songs. It is therefore not inconceivable to have specialized image processing hardware similarly integrated into mainstream devices. This hardware would probably be motivated first and foremost by the needs of image editing and graphics-intensive games, but could conceivably also be used for other image processing tasks.

Finally, it can be said that OCR on mobile devices is held back by the lack of fast CPUs or specialized image processing hardware. Poor optics could also be identified as a problem, although those problems were not encountered in this work.

References

[1] www.Forbes.com. Hitachi Mobile phone with OCR. `http://forbes.com/technology/feeds/infoimaging/2005/03/18/infoimagingasiapulse_2005_03_18_ix_4427-0236-.html`

[2] www.3g.co.uk. Pantech&Curitel OCR Mobile phone. `http://www.3g.co.uk/PR/Jan2005/8935.htm`

[3] QuickTextScan OCR software. `http://jsscomputing.com/quicktextscan/index.html`

[4] Shunji Mori (Author), Hirobumi Nishida (Author), Hiromitsu Yamada (Author), 1999. Optical Character Recognition. Wiley.

[5] H. Bunke, P.S.P. Wang (editors), 1997. Handbook of Character Recognition and Document Image Analysis. World Scientific Publishing Company.

[6] Øivind Due Trier, Anil K, Jain, 1995. Goal-Directed Evaluation of Binarization Methods IEEE TPAMI Vol 17, No 12, pp. 1191-1201.

[7] Kongqiao Wang, Jari A Kangas, 2003. Character location in scene images from digital camera. Pattern Recognition Vol 36, pp. 2287-2299.

[8] C. C. Chang, S. M. Hwang, D. J. Buehrer, 1991. A Shape Recognition Scheme based on Relative Distances of Feature Points from the Centroid. Pattern Recognition, Vol. 24, No. 11, pp. 1053-1063.

[9] Thomas Bernier, Jacques-André Landry, 2003. A new method for representing and matching shapes of natural objects. Pattern Recognition, Vol. 36, No. 8, pp. 1711-1723.

[10] Leigh Edwards, Richard Barker, 2004. Developing Series 60 Applications. Addison Wesley.

[11] Answers.com. Nokia 6630 smartphone specifications. `http://www.answers.com/topic/nokia-6630`

[12] Mikael Laine, Johannes Tuikkala, Radial sweep algorithm for Character Recognition. `http://users.utu.fi/mialain/code/lab4/labra4.pdf`

Authorized licensed use limited to: Reprints Desk. Downloaded on July 23,2025 at 20:35:56 UTC from IEEE Xplore. Restrictions apply.