International Journal of Information Technology, Vol. 12 No.4 2006

Inertial and Big Drop Fall Algorithm

Xiujuan Wang, Kangfeng Zheng, and Jun Guo

School of Information and Engineering, Beijing University of Posts and Telecommunication,
Beijing, China 100876
xjwangbupt @yahoo.com zhengkfbupt @163.com guojun@bupt.edu.cn

Abstract

Segmenting connected characters is an essential preprocessing step in character recognition applications. Traditional drop fall algorithm has proved to be an efficient segmenting method due to its simpleness and effectiveness. But it is subject to small convexness on the contour of characters. This paper presents Inertial Drop Fall algorithm and Big Drop Fall algorithm to overcome this defect. The experimental tests show the effectiveness of the two methods in obtaining high-confidence segmentation traces.

Keywords: Drop Fall, Inertial Drop Fall, Big Drop Fall, segmentation

I. Introduction

Segmentation is pivotal work in character recognition especially in case hand-written characters are connected. During past 50 years, many methods have been set forth in segmenting connected characters.[1-7]Drop Fall algorithm is a classical segmentation algorithm often used in character segmentation because of its simpleness and effectiveness in application. Firstly advanced by G. Congedo in 1995, Drop Fall algorithm mimics the motions of a falling raindrop that falls from above the characters rolls along the contour of the characters and cuts through the contour when it cannot fall further. The raindrop follows a set of movement rules to determine the segmentation trace. Concretely, the Drop Fall algorithm selects one pixel out of the neighbors of the current pixel as a new pixel of the segmentation trace.

Although Extended Drop Fall algorithm has been advanced [9,10] to improve the performance of drop fall algorithm, when the raindrop falls into the concave pixel between the small convexnesses on the contour of characters, these algorithms will treat it as connected strokes and therefore start splitting it. Obviously it could split a single character and result in invalid segmentation. In this case, we introduce Inertial Drop Fall algorithm which follows the previous direction in the segmentation. Furthermore, Big Inertial Drop Fall algorithm is advanced to increase the size of the raindrop. When there is no big enough free space for the big raindrop to fall down, it will search for other direction and thus can avoid fall into the concave.

Note that, for ease to study, only two connected characters are dealt with as primary solution which could be expanded to more than two connected characters. In this paper, connected hand-written numerals are used as examples to check up whether the algorithms are valid.

II. Traditional Drop Fall algorithm and Big Inertial Drop Fall algorithm

A. *Traditional Drop Fall algorithm*

As mentioned above, the basic idea of Traditional Drop Fall (TDF) algorithms is to simulate a "drop-falling" process.[8] The cut tracing is defined with both the information of neighbor pixels and perhaps of more pixels. The algorithm considers only five adjacent pixels: the three pixels blow the current pixel and the pixels to the left and right .Upward moves are not considered, because the rules are meant to mimic a falling motion. Designating n_0 as the current pixel in the cut trace as illustrated in Figure 1:

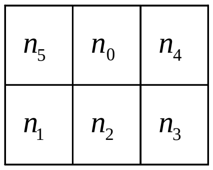


Figure 1:The neighbor pixels of n_0

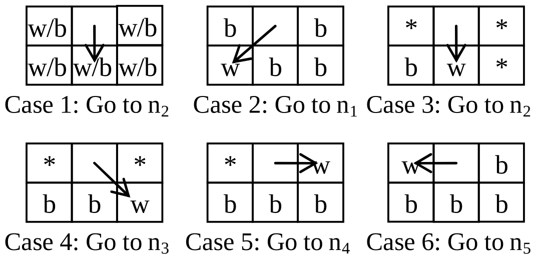


Figure 2 Select next pixel in TDF

Then the next pixel will be selected from $n_i\,(i = 1,2,3,4,5)$ .In TDF algorithm, the relation between the current pixel and the next one is as illustrated in Figure 2, which shows to us when all of the neighbor pixels of n_0 are white or black, then the next pixel is n_2

In order to give explicit definition in determining the segmenting trace T, we provide a mathematical description of the TDF algorithm. Denote the picture to be segmented as P. And the size of P is $M \times N$, in which M is the width of P and N is its height. Furthermore, we establish a coordinate in which the origin O is just at the most left-top point of P(as shown in Figure3).

Assume binarization has been implied to the P, i.e. all the values Z of the image dots are either '1' or '0'. '1' indicates black pixel and '0' white pixel here. Therefore T depends on the current pixel and its neighbors:

$$T(x_{i+1}, y_{i+1}) = f(x_i, y_i, W_i), \text{ i=0,1,2,....} \qquad (2\text{-}1\text{-}1)$$

Here (x_i, y_i) presents the coordinate of the current pixel and (x_0, y_0) when i equals to zero presents the start point from which the segmentation begins. How to find the start point will be explained in detail thereinafter. The W_i is a measurement of the weight of the raindrop. It's value depends on the Z_i of the neighbor pixels n_1, n_2, n_3, n_4, n_5 as shown in Figure 2.

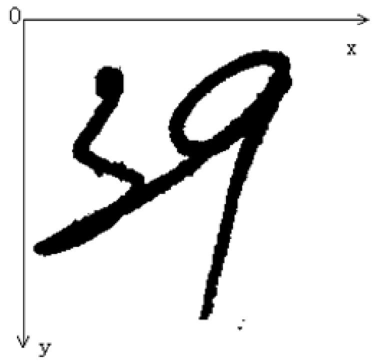


Figure 3 Establish a coordinate of *P*

According to the TDF algorithm, n_1, n_2, n_3, n_4, n_5 have various priorities to become a point in the T. Then we define different weights w_j for them:

$$w_j = 6 - j, (j = 1, 2, 3, 4, 5) \qquad \text{(2-1-2)}$$

Then W_i can be defined as:

$$Wi = \begin{cases} 4 & if \sum = 0 or 15 \\ \max\limits_{j=1}^{5} (1 - Z_j) w_j & else \end{cases}$$

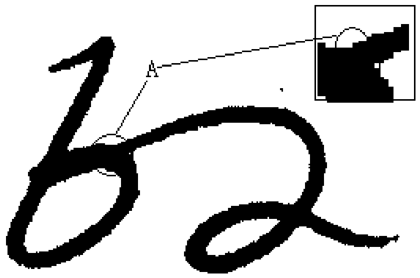
(2-1-3)

where $\sum = \sum\limits_{j=1}^{5} (1 - Z_j) w_j$, and

$$T(x_{i+1}, y_{i+1})_{TDF} = f(x_i, y_i, W_i)$$
$$= \begin{cases} (x_i - 1, y_i) & if W_i = 1 \\ (x_i + 1, y_i) & if W_i = 2 \\ (x_i + 1, y_i + 1) & if W_i = 3 \\ (x_i, y_i + 1) & if W_i = 4 \\ (x_i - 1, y_i + 1) & if W_i = 5 \end{cases}$$

(2-1-4)

But it is found that cursive stroke tends to bewilder the judgment of the condition and lead to error segmentation. As illustrated in Figure 4a, point A is where the cursive stroke is connected with the normal numeral 6. When the raindrop falls into point A, although we know that the segmentation should keep going right, the TDF algorithm regard it as case 1 in which the raindrop would go down to cut and hence invalid segmentation would be obtained. In addition, burrs of the numerals can also influence the segmentation greatly. As shown in Figure 4B, point A is between burrs of the numeral 6. It is ease for human to be wise to let the segmenting trace get across point A. But when the raindrop falls into the point A, the TDF judge the condition and cut down as defined in case 1. Obviously incorrect segmentation is obtained. In order to overcome those disadvantages in the TDF, we extend it to Inertial Drop Fall algorithm (IDF) and Big Inertial Drop Fall algorithm (BIDF).



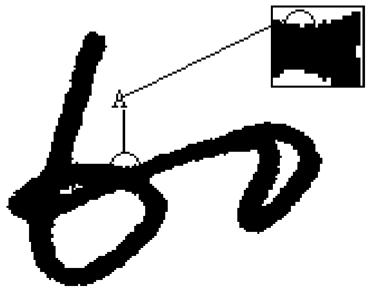


Figure 4a Cursive stroke leads to false judgement   Figure 4b Local convex leads to false judgement

B. ***Inertial Drop Fall algorithm (IDF)***

As described above, the TDF algorithm didn't take it into account that handwritten numerals are generally silken, so do printed numerals. It can be conceived that when a raindrop falls along a smooth marble, the drop will get inertia. Therefore this character should be embodied in the segmenting trace. Then we presume that when all of the five neighbor pixels are black, which one out of them is the next pixel will also depend on how the trace crawls before as well as the gravity of the drop. In another words, the inertia of the drop will affect the direction of segmentation together with the weight. That is:

$$T(x_{i+1}, y_{i+1})_{IDF} = f'(x_i, y_i, W_i, \overline{I_i}) \quad (2\text{-}2\text{-}1)$$

where $\overline{I_i}$ represents the inertial of the raindrop at the current pixel (x_i, y_i). For ease in study, we consider only the direction of the inertial without its magnitude. And

$$\overline{I_i} = \begin{cases} \overline{V_{04}} & if x_i > x_{i-\alpha} \\ \overline{V_{05}} & if x_i < x_{i-\alpha} \\ 0 & else \end{cases} \quad (2\text{-}2\text{-}2)$$

where $\overrightarrow{V_{04}}$ and $\overrightarrow{V_{05}}$ refer to the vector from n_0 to n_4 and the vector from n_0 to n_5 respectively. α is a non-negative integer.

$$Wi = \begin{cases} 4 & if \sum = 0 \\ 6 & if \sum = 15 \\ \max\limits_{j=1}^{5}(1-Z_j)w_j & else \end{cases} \qquad (2\text{-}2\text{-}3)$$

where $\sum = \sum\limits_{j=1}^{5}(1-Z_j)w_j$, and the value of w_i is also from expressions (2-1-2).

We explain the meanings of the above expressions as followed: record the last α pixel $T(x_{i-\alpha}, y_{i-\alpha})$ in the segmenting trace before the current pixel $T(x_i, y_i)$ when $\sum = 0$ i.e. all the neighbor pixels $n_i (i = 1,2,3,4,5)$ around n_0 are black. In experiments, α is set to 5. If $x_i > x_{i-\alpha}$, it is determined that $T(x_i, y_i)$ is at the right of $T(x_{i-\alpha}, y_{i-\alpha})$, that is, the segmenting trace has the inertia toward right at the current point. As illustrated in Figure 5, the vector from n_0 to n_4 $\overrightarrow{V_{04}}$ is the inertia toward right and the vector from n_0 to n_2 represents the weight. The effects of these two vectors result in the vector from n_0 to n_3. In this case, let the next pixel be n_3. But if $x_i < x_{i-\alpha}$, i.e. $T(x_i, y_i)$ is at the left of $T(x_{i-\alpha}, y_{i-\alpha})$, the segmenting trace has the inertia $\overrightarrow{V_{05}}$ at the current point. But the inertia toward left is not desired because the segmenting trace is most probably following the right contour of the left character which means that inertia toward left would probably lead to failure. Experiments followed have proved it. Therefore in this case, the next pixel $T(x_{i+1}, y_{i+1})$ should remain to be n_2 just as the TDF algorithm.

Certainly when $\sum = 15$ i.e. all of the neighbor pixels of n_0 is white, the next pixel remains n_2 since usually the blank region is wide and the same processing as described above will probably result in wrong segmentation.

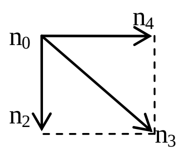


Figure 5 The effect of two vectors

Therefore we define:

$$T(x_{i+1}, y_{i+1})_{IDF} = f'(x_i, y_i, W_i, \vec{I_i}) = \begin{cases} (x_i - 1, y_i) & if W_i = 1 \\ (x_i + 1, y_i) & if W_i = 2 \\ (x_i + 1, y_i + 1) & if W_i = 3 \\ (x_i, y_i + 1) & if W_i = 6 \\ (x_i - 1, y_i + 1) & if W_i = 5 \\ (x_i + 1, y_i + 1) & if W_i = 4\, and\, \vec{I_i} = \overrightarrow{V_{04}} \\ (x_i, y_{i+1}) & if W_i = 4\, and\, \vec{I_i} = \overrightarrow{V_{05}}, 0 \end{cases} \quad (2\text{-}2\text{-}4)$$

C. ***Big Inertial Drop Fall algorithm (BIDF)***

Experiments show that the Inertial Drop Fall can improve the segmentation performance in some cases. But the result is not good enough to overcome the defect in the TDF that incorrect segmentation may occur when the drop falls into the recess between burrs. But assume that the drop is big enough; it is then more possible for the drop to escape from the recess just as the water in a river can flow forward without prevented by the unsmooth bank. It could be conceived that a rain strip can work as a rain ball in dealing with burrs. Therefore we adjust our scheme as following:

The current segmenting element is 2B ＋ 1 adjacent pixels ($p_1, p_2, \cdots p_{B+1}, \cdots p_{2B}, p_{2B+1}$) as shown in Figure 6, rain strip thereinafter, instead of single one, each of which has the same y-axis. Denote the rain strip as $S_i = S(x_{i-B}, x_{i+B}, y_i)$, in which the B, an integer larger than or equal to zero, represents a measurement related to the width of the rain strip, and (x_i, y_i) represents the center pixel in the current rain strip while x_{i-B}, x_{i+B} respectively indicate the abscissas of the start pixel and the end pixel of the rain strip.

p_0	p_1	p_2	…	p_{B+1}	…	p_{2B+1}	p_{2B+2}
p'_0	p'_1	p'_2	…	p'_{B+1}	…	p'_{2B+1}	p'_{2B+2}

Figure 6 The neighbor pixels of rain strip

Then the segmenting trace T is:

$T(S_{i+1})_{BIDF} = f''(S_i, W_i, \vec{I_i})$ (2-3-1)

where:

$$Wi = \begin{cases} 4 & if \sum = 0 \\ 6 & if \sum = 15 \\ \max_{j=1}^{5}(1 - Z'_j)w_j & else \end{cases}$$

(2-3-2)

Where $\sum = \sum_{j=1}^{5}(1 - Z'_j)w_j$ and the value of w_i is also from expression (2-1-2). It should be noted that Z'_j is different from Z_j since when j equals to 1,2 and 3, Z'_j represents the value of $S(p'_0, p'_{2B})$, $S(p'_1, p'_{2B+1})$ and $S(p'_2, p'_{2B+2})$ respectively. So we define:

$$Z'_j = \begin{cases} \max_{i=j-1}^{2B+j} Z_i & j = 1,2,3 \\ Z_j & else \end{cases}$$ (2-3-3)

$$T(S_{i+1})_{BIDF} = f''(S_i, W_i, \vec{I_i}) =$$

$$\begin{cases} S(x_i - B - 1, x_i + B - 1, y_i) & ifW_i = 1 \\ S(x_i - B + 1, x_i + B + 1, y_i) & ifW_i = 2 \\ S(x_i - B + 1, x_i + B + 1, y_i + 1) & ifW_i = 3 \\ S(x_i - B, x_i + B, y_i + 1) & ifW_i = 6 \\ S(x_i - B - 1, x_i + B - 1, y_i + 1) & ifW_i = 5 \\ S(x_i - B + 1, x_i + B + 1, y_i) & ifW_i = 4\, and\, \vec{I_i} = \overrightarrow{V_{04}} \\ S(x_i - B, x_i + B, y_i + 1) & ifW_i = 4\, and\, \vec{I_i} = \overrightarrow{V_{05}}, 0 \end{cases}$$ (2-3-4)

in which $\vec{I_i}$ is up to expression (2-2-2).

In the BIDF, we consider the rain strip S_i comprising 2B+1 pixels from (x_{i-B}, y_i) to (x_{i+B}, y_i) as a whole pixel. And we define if at least one pixel out of the 2B+1 pixels is black, then the rain strip S_i is treated as a black pixel and S_i is looked as a white pixel only when all of those pixels are white. In doing this, we can perform the BIDF just as the IDF.

It can be easily understood that too large value of B can do no good to our segmentation. In the paper we would try several integers for the B, such as 0, 1 and 2. Especially, when B is equal to 0, then the BIDF algorithm is in fact the above-mentioned IDF algorithm. That's to say, the IDF is essentially a special case of the BIDF algorithm.

D. ***Find the start point***

Where to drop the marble from is important because if the algorithm starts in the wrong place, completely ineffective segmenting traces will be obtained.

Several methods are available to decide where to start the drop fall process from. Obviously, it is best to start as close as possible to the point at which the two characters are connected. Dimauro [8] outlined a method which does this quite robustly. In this method, the picture P is scanned row-by-row until a black boundary pixel Ω with another black boundary pixel to the right of it is detected, where the two pixels are separated by only white pixels. Pixel Ω is then used as the point from which to start the drop fall. In this paper, we adopt this method to find the start point. And the experimental results do not take those data into account whose start points are not correctly determined.

III. Experimental Results

In the experiment, we invite ten people to write down optionally 100 two-connected numerals from 00 to 99 to provide a test set of one thousand. Then the TDF, BIDF algorithms are applied to the test set in which B is respectively 0, 1, and 2. Therefore we can get four results to compare artificially and judge whether or not the algorithms developed in this paper are valid.

Figure 7 shows the segmenting results of TDF and BIDF. For about 15% cases, the BIDF can give better result than the TDF algorithm (see Figure 8). Experiments show that the algorithms do have improved the segmenting especially enhanced the performance of Drop Fall algorithm in case of local convexness.

A conclusion section is not required. Although a conclusion may review the main points of the paper, do not replicate the abstract as the conclusion. A conclusion might elaborate on the importance of the work or suggest applications and extensions.

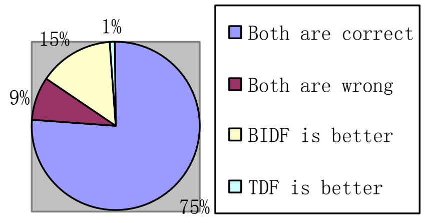


Figure 7 Segmenting results of TDF and BIDF

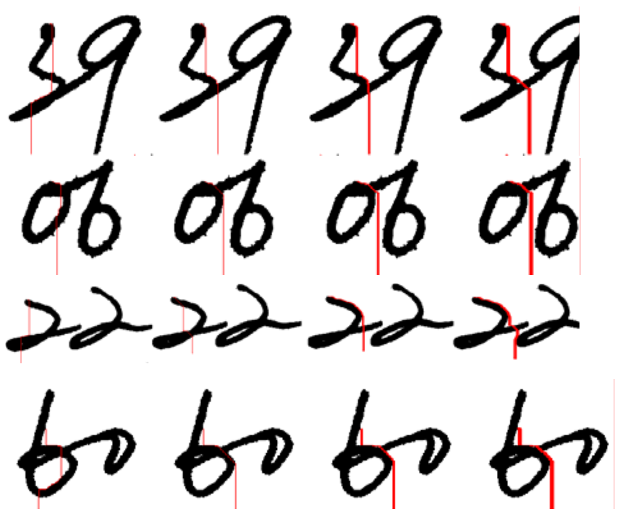


Figure 8: From left to right are: Traditional drop-falling, Big Inertial drop-falling of B=0,1,2

References

[1] Dimauro, G.; Impedovo, S.; Pirlo, G.; and Salzo, A.: “Automatic Bankcheck processing: A New Engineered System.” International Journal of Pattern Recognition and Artificial Intelligence. 11(4),1997,pp467-504

[2] Blumenstein, M. and Verma, S."A Neural Based Segmentation and Recognition Technique for Handwritten Words”,IEEE International Conference on Neural Networks, Vol.3,1998, pp1738-1742

[3] R.Fenrich,”Segmentation of automatically located handwritten numeric strings” in Form Pixels to Features III-Frontiers in Handwriting Recognition, S. Impedovo(ed.), North-Holland Publ. 1992,pp47-60

[4] Casey, R. G.; Lecolinet, E.; “A Survery of Methods and Strategies in Character Segmentation”, IEEE transactions on pattern analysis and machine intelligence.18(7), (1996),pp.690

[5] Lu, Y.; Shridhar, M.; “Character Segmentation in Handwritten Words- An Overvies”, Pattern recognition. 29(1) ,(1996)pp. 77

[6] Shridar, M.;Badredlin; “Recognition of Isolated and Simple Connected Handwritten Numerals”, Pattern Recognition, 19(1) (1986),pp.1,

[7] R. Fenrich and K. Krishnamoorthy, “Segmenting diverse quality handwritten digit strings in near real-time”, Procedings of the 4th Advanced Technology Conf. (1990),pp523-537

[8] G.Congedo, G.Dimauro, S.Impedovo,G.Pirlo, “Segmentation of Numeric Strings”,1995,pp1038-1041

[9] Salman Amin Khan “Character Segmentation Heuristics for Check Amount” Verification1998,pp14-29

[10] Jibu Punnoose "An Improved Segmentation Module for Identification of Handwritten Numerals"1999,pp20-22

Xiujuan Wang, born in 1979, received her B.Sc. degree and M.Sc. degree from Shandong University in Communication and Information System. She is currently a doctorial student associate in the School of Information and Engineering at Beijing University of Posts and Telecommunication. She majors in signal and information processing, intelligent information retrieval.

Kangfeng Zheng, born in 1975, received his B.Sc. degree and M.Sc. degree from Shandong University in Communication and Information System. He is currently a doctorial student associate in the School of Information and Engineering at Beijing University of Posts and Telecommunication. He majors in signal and information processing, information security.

Jun Guo, born in 1959, received his B.Sc. degree and M.Sc. degree from Beijing University of Posts and Telecommunication in Computer and Communication, received his Ph.D. degree from Tohoku Gakuin University in Japan. He is currently a professor in the School of Information and Engineering at Beijing University of Posts and Telecommunication, advanced member of Chinese Electronic Society ,and a member of Japanese Electronic Information Communication Society . His current research interests are in pattern recognition and Chinese information processing.