Correlated Run Length Algorithm (CURL) for Detecting Form Structure within Digitized Documents

Michael D. Garris
National Institute of Standards and Technology
Bldg. 225, Rm. A216
Gaithersburg, MD 20899

In: Third Annual Symposium on Document Analysis and Information Retrieval, pp. 415-424, UNLV, April 1994

Abstract

An algorithm designed to detect intrinsic form structure within binary digitized documents has been developed. This Correlated Run Length algorithm automatically locates and extracts line segments, line endings, and combinations of line intersections including corners, crosses, and T's from images. These structures, once detected, can be used as form features to identify the form type in an image, and they can be used to automatically identifying entry fields containing information in need of optical character recognition. This technique has several advantages over more conventional approaches in that form structures are detected without any a priori knowledge of the specific form in the image, and these structures are detected directly from the original image so that any distortions including translation, rotation, and scale are automatically handled. The algorithm performs extremely well on highly cluttered forms and noisy images and is well suited for implementation in a highly parallel processing environment.

1 Introduction

Imaging technologies have been rapidly advancing as the performance of computers and the intelligence of algorithms have continued to increase. Current document processing technologies rely heavily on the use of forms printed with drop-out inks and are overly constrained to rigid form design and printing specifications. Document processing applications frequently cannot be limited to these constraints rendering the technology useless or only partially useful. In order to effectively and efficiently handle electronically scanned documents, automated document processing technologies must continue to improve.

The Correlated Run Length algorithm (CURL) has been designed to support flexible forms identification and entry field location. These functions serve as font-ends to Optical Character Recognition (OCR) systems. CURL automatically detects a wide variety of structures frequently incorporated into the design of a form. These structures include line segments, line endings, and combinations of line intersections (corners, crosses, and T's). These low-level structures form higher-level constructs including lines, boxes, and grids denoting form regions and data entry fields. By detecting these intrinsic structures, the form type of an image can be identified.

The NIST Model Recognition System [1] currently uses localized spatial histograms to detect and locate the types of form structures

listed above. Every entry field on the forms processed by the system is demarcated by a bounding box like the entry field shown in Figure 7. The field isolation module of the system uses a static template to direct localized histogram searches in an attempt to locate the four edges of each box. This is accomplished by defining a region of interest in which both horizontal and vertical histogram projections are calculated. The corners of a box are located by finding coinciding bins containing relative maxima between the horizontal and vertical projections.

The localized histogram technique works well when the boxes are well defined by bold markings, when the boxes are more than sufficient in size to comfortably contain the data requested to be entered into the box, and when the boxes are adequately spaced apart from one another on the form. When a form does not conform to these constraints, the accuracy of the localized histogram technique quickly degrades due to ambiguities introduced in the histogram projections. These projections can be used to describe orthogonal densities, but they cannot be used to accurately detect complex shapes, such as box corners, on cluttered and cramped forms because the technique simply aggregates pixels in rows and columns without taking pixel contiguity into account.

In his paper, "Image Segmentation with Networks of Variable Scales", Hans Graf describes an experiment in which he studied the application of a set of kernels for detecting edges and corners in gray scale images of boxcars.[2] The kernels define directions and shapes of contrast change in local neighborhoods. In the experiments described in Graf's paper, the kernels were very successful in detecting characters on the sides of boxcars. The success can be attributed primarily to the fact that the gray scale background of the boxcars is relatively uniform, so that contrast changes primarily occur around the characters of interest. This technique suffers when applied to binary scanned documents and forms that are comprised almost entirely of edge information. Hand print strokes, machine print text, and form structures are all ambiguously represented by collections of edges. Therefore, contrast detection kernels detect contrast-based structures at too fine a detail to be useful in extracting higher-level form structures.

The algorithm presented in this paper extracts larger-scale shape-based structures, not contrast-based structures, and accurately distinguishes hand print strokes and machine print text from form structures. This paper demonstrates how the CURL algorithm overcomes the deficiencies of localized histograms and contrast kernels by using vector correlations that reward contiguity of pixels through the use of run length values. Section 2 provides a technical description of the algorithm, Section 3 shows working examples for entry fields and form structures, Section 4 points out various implementation issues, and Section 5 demonstrates CURL's tolerance to rotational distortions.

2 Technical Description of CURL

2.1 An Overview

CURL correlates and aggregates pixels along selected trajectories in order to detect and locate shape-based structures within an image. Shape is represented by at least two edge vectors called an *edge pair*. The elements of the edge vectors address pixel positions within the input image, and these pixel addresses are defined relative to a current pixel location within the image. The edge pair is applied independently to each pixel in the image, extracting pixels along the specified trajectories. For example, one edge vector may be defined to extend horizontally 32 pixels to the right of the current pixel, and another edge may be defined to extend vertically 32 pixels below the current pixel. CURL uses this edge pair definition to detect upper-left corners of boxes. CURL is not limited to linear edges only. A point-to-point

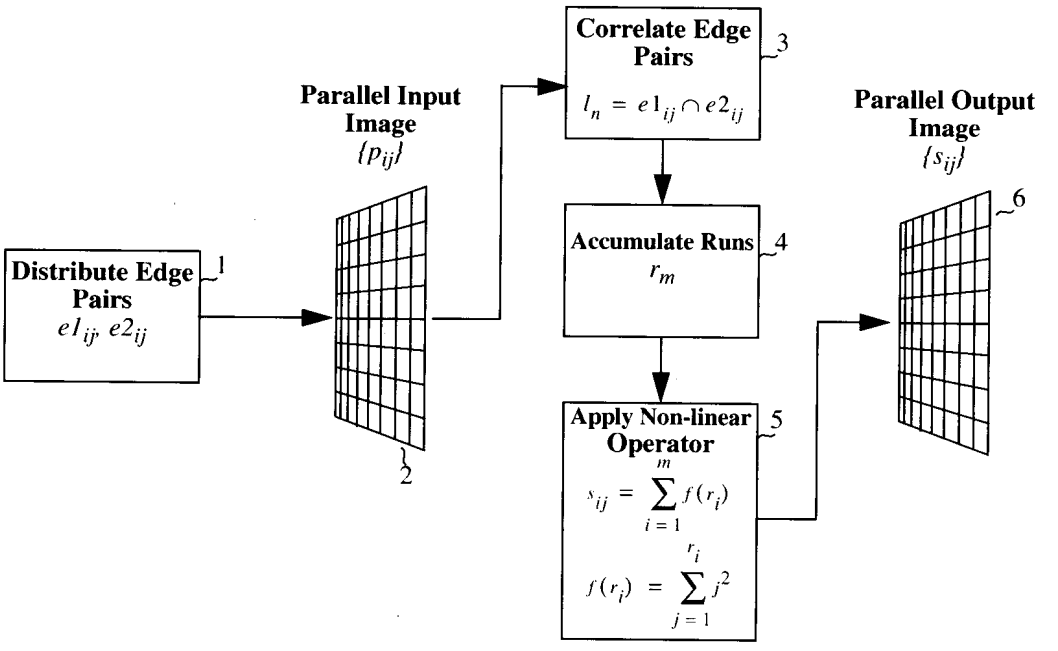


Figure 1: Flow diagram describing the CURL algorithm.

correlation can be computed between any two or more vectors representing any given shape and the points within each vector may be spaced apart from one another.

Applying an edge pair to each pixel position in the image, an intersection is computed between the two vectors of extracted pixels, forming contiguous groups of correlated pixels called runs. A non-linear operator is applied to the length of each resulting run called a *run length*. The non-linear accumulation of a run length accelerates rapidly as the duration of the contiguously correlated pixels increases. The accumulation grows very little for uncorrelated edge vectors because the runs are short. In this way, edge pairs can be defined to detect arbitrary shapes.

Figure 1 illustrates the CURL algorithm as a sequence of fundamental steps. First, a selected set of edge pairs represented by box 1 are distributed across every pixel in input image 2. The intersection in box 3 is computed for each edge pair extracted from the input image. Run lengths in box 4 are computed from each intersection, and a non-linear operator in box 5 is applied to the run lengths. Finally, each pixel in output image 6 is assigned the accumulated results from the non-linear operator for a given pair of edges.

2.2 The Algorithm

For each pixel p_{ij} in a binary image, two pixel edges $e1_{ij}$ and $e2_{ij}$ of equal length n are defined with origins relative to p_{ij}. The intersection of the two edges form a single logical vector l_n.

$$l_n = e1_{ij} \cap e2_{ij} \qquad (1)$$

The vector l_n is divided into groups of contiguous elements with values of 1, with each group separated by one or more 0's. Run lengths are accumulated for each contiguous group and stored in the list r_m, where m is the number of groups. The list of run lengths r_m is used to compute an output value s_{ij} for each edge pair belonging to the pixel p_{ij} as follows:

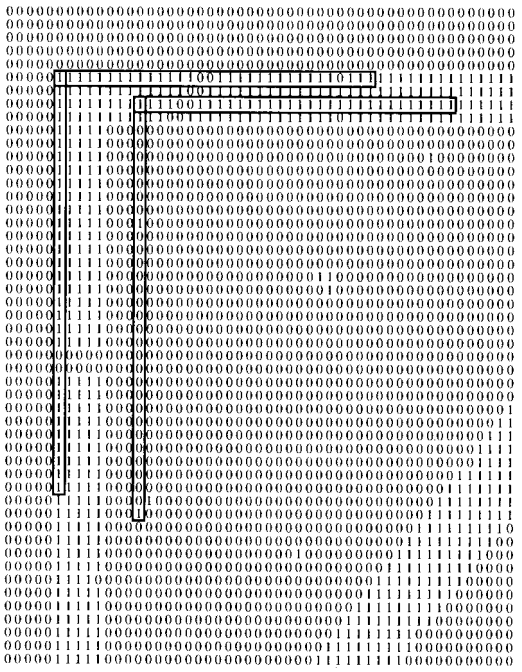


Figure 2: An example binary image containing a corner.

$$s_{ij} = \sum_{k=1}^{m} f(r_k) \qquad (2)$$

where:

$$f(r_k) = \sum_{t=1}^{r_k} t^2 \qquad (3)$$

For example, Figure 2 lists a portion of an image containing the upper-left hand corner of a box. Notice that the bottom-right hand portion of the image contains pixel information of what was entered in the box. In order to locate the corner in the image, the first edge $e1_{ij}$ is defined as the row of pixels with its origin at p_{ij} and extending to the right n pixels, where n in this example equals 32. The second edge $e2_{ij}$ is defined as the column of pixels with its origin at p_{ij} and extending downward 32 pixels. The edge pairs for two pixels, $p_{5,5}$ and $p_{13,7}$, are outlined in the figure. Both edges of $p_{5,5}$ lie on the corner, whereas only one edge for $p_{13,7}$ lies on the corner.

Figure 3 illustrates how the value $s_{5,5}$ is computed by applying CURL to the edge pair corresponding to $p_{5,5}$ outlined in Figure 2. The first two vectors in Figure 3 are the edges $e1_{5,5}$ and $e2_{5,5}$. The third vector is the result of the intersection of the two edges. The fourth line contains the resulting run lengths accumulated from each contiguous group of 1's resulting from the intersection. The fifth line in the figure lists the values computed for each run length using equation (3). The last line shows the result of the summing the values listed on the fifth line according to equation (2).

Figure 4 illustrates how the value $s_{13,7}$ is computed by applying CURL to the edge pair corresponding to $p_{13,7}$ outlined in Figure 2. The first two vectors in the figure below are the edges $e1_{13,7}$ and $e2_{13,7}$. The third vector is the result of the intersection of the two edges. The fourth line contains the resulting run lengths accumulated from each contiguous group of 1's resulting from the intersection. The fifth

$e1_{5,5}$: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1

$e2_{5,5}$: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1

l_{32} : 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1

r_4 : 14, 5, 5, 3

$f(r_k)$: 1015, 55, 55, 14

$s_{5,5}$ = 1139

Figure 3: Correlated run length example for $p_{5,5}$.

$e1_{13,7}$: 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

$e2_{13,7}$: 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

l_{32} : 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

r_3 : 2, 1, 1

$f(r_k)$: 5, 1, 1

$s_{13,7}$ = 7

Figure 4: Correlated run length example for $p_{13,7}$.

$e1_{5,5}$: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1

$e2_{5,5}$: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1

l_{32} : 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1

c_{32} : 1 2 3 4 5 6 7 8 9 10 1 1 12 13 14 0 0 1 2 3 4 5 0 0 1 2 3 4 5 0 1 2 3

$s_{5,5}$ = $c_{32} \cdot c_{32}$

$s_{5,5}$ = 1139

Figure 5: Alternative implementation of equations (2) and (3).

line in the figure lists the values computed for each run length using equation (3). The last line shows the result of the summing the values listed on the fifth line according to equation (2).

Comparing ($s_{5,5}$ = 1139) to ($s_{13,7}$ = 7) demonstrates how CURL can be used to emphasize corner information within a binary image. Note that equation (3) is one of many possible choices, which may be used to measure the correlation between edge pairs. By replacing the contiguous groups of 1's in l_n with consecutive increments of the run length counter, such as shown in Figure 5, equations (2) and (3) may be implemented as the dot product of the vector c_n with itself. However, by using the approach designated by equations (2) and (3), alternative functions for $f(r_k)$ may be used. In fact, each candidate function may be implemented as a look-up table containing precomputed values based on r_k.

2.3 Edge Pair Definitions

To date, numerous edge pair definitions have been studied. They include an upper left-hand corner (c_{ul}) pair, an upper right-hand corner (c_{ur}) pair, a lower left-hand corner (c_{ll}) pair, a

$$
\begin{aligned}
c_{ul} \quad : \quad & e1_{ij} = \{p_{kl}: k = i, \ldots, i+n-1; l = j\} \\
& e2_{ij} = \{p_{kl}: k = i; l = j, \ldots, j+n-1\} \\
c_{ur} \quad : \quad & e1_{ij} = \{p_{kl}: k = i, \ldots, i-n+1; l = j\} \\
& e2_{ij} = \{p_{kl}: k = i; l = j, \ldots, j+n-1\} \\
c_{ll} \quad : \quad & e1_{ij} = \{p_{kl}: k = i, \ldots, i+n-1; l = j\} \\
& e2_{ij} = \{p_{kl}: k = i; l = j, \ldots, j-n+1\} \\
c_{lr} \quad : \quad & e1_{ij} = \{p_{kl}: k = i, \ldots, i-n+1; l = j\} \\
& e2_{ij} = \{p_{kl}: k = i; l = j, \ldots, j-n+1\} \\
l_h \quad : \quad & e1_{ij} = \{p_{kl}: k = i, \ldots, i+n-1; l = j\} \\
& e2_{ij} = \{p_{kl}: k = i, \ldots, i-n+1; l = j\} \\
l_v \quad : \quad & e1_{ij} = \{p_{kl}: k = i; l = j, \ldots, j+n-1\} \\
& e2_{ij} = \{p_{kl}: k = i; l = j, \ldots, j-n+1\}
\end{aligned}
$$

Figure 6: Edge pair definitions used by CURL.

lower right-hand corner (c_{lr}) pair, a horizontal line (l_h) pair, and a vertical line (l_v) pair. Figure 6 lists the definitions of these edge pairs using set notation. These definitions assume the origin of an image is its upper left-hand corner. The edge pair examples outlined in Figure 2 are of type c_{ul}.

Note that CURL is not limited to the set of linear edges shown in Figure 6. A point-to-point correlation can be computed between any two vectors representing any given shape and the points within each vector may be spaced apart from one another. Also, intersections may be computed on more than two edge vectors simultaneously. For example, T-shaped structures are detected with three edges, and cross-shaped structures are detected by using four edges.

3 Example Results

3.1 Entry Field Results

Based on the algorithm described above, each of the edge pairs in Figure 6 were studied on different test images. Figure 7 shows an example using the upper-left corner edge pair, c_{ul}, with the original test image on the left, the gray scale results in the middle, and the thresholded binary results on the right. Notice that the c_{ul} edge pair locates the upper left-hand corner of the box while ignoring all other pixel information in the image. Similar results are achieved when using the c_{ur}, c_{ll}, and c_{lr} edge pairs.

Figure 8 shows an example using the horizontal line edge pair, l_h, with the original image on the left, the gray scale results in the middle, and the thresholded binary results on the right. Notice that the l_h edge pair locates the horizontal sides of the box while ignoring all other pixel information in the image. Similar results are achieved when using the l_v edge pair.

Figure 9 shows an example using all the edge pairs defined in Figure 6 with the original image on the left, the gray scale results in the middle, and the thresholded binary results on the right. Notice that the sides and corners of the box are detected while ignoring all other pixel information in the image. The results from each edge pair are combined by simply accumulating the s_{ij} for each edge pair into a single output element.

3.2 Form Results

This section demonstrates how CURL can be used to extract intrinsic form structures from an image. In Figure 10, a section of a form is

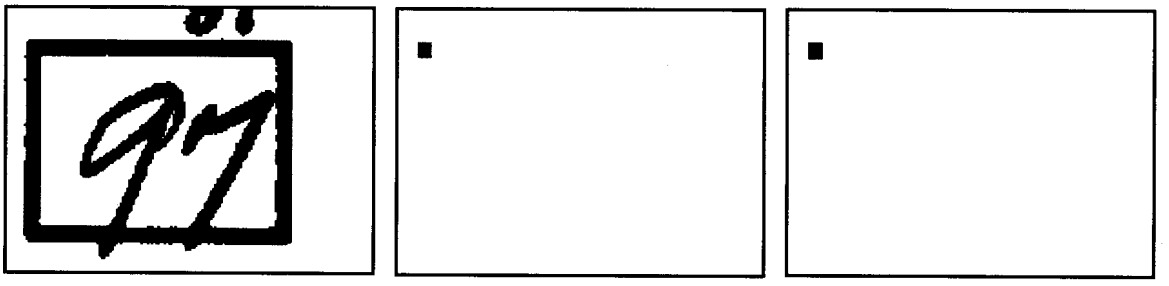


Figure 7: Results from using the c_{ul} edge pair.

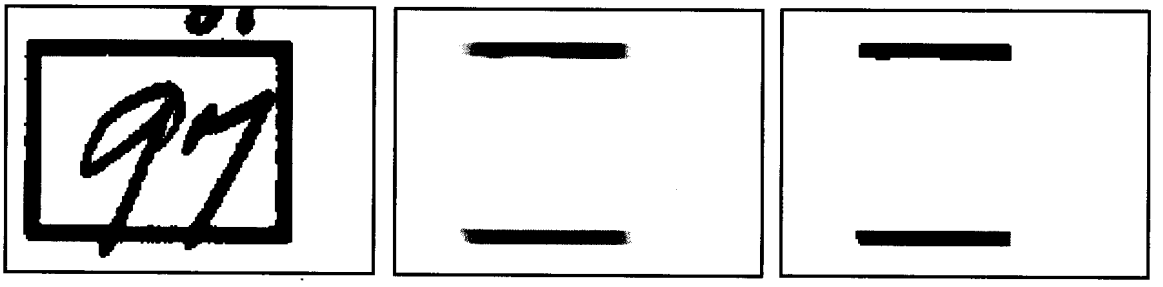


Figure 8: Results from using the l_h edge pair.

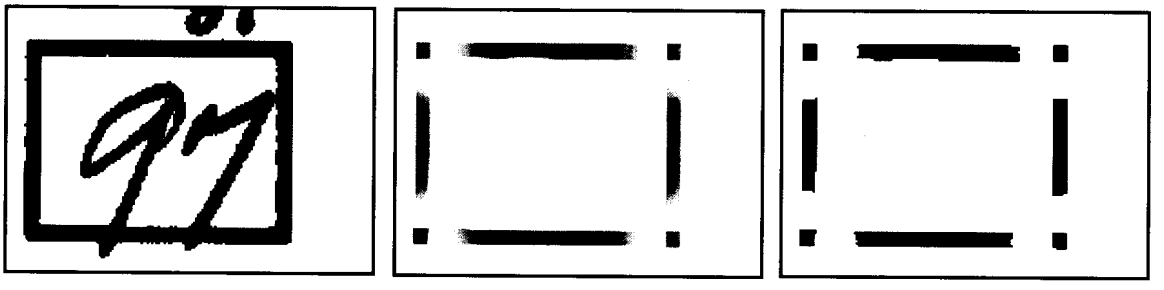
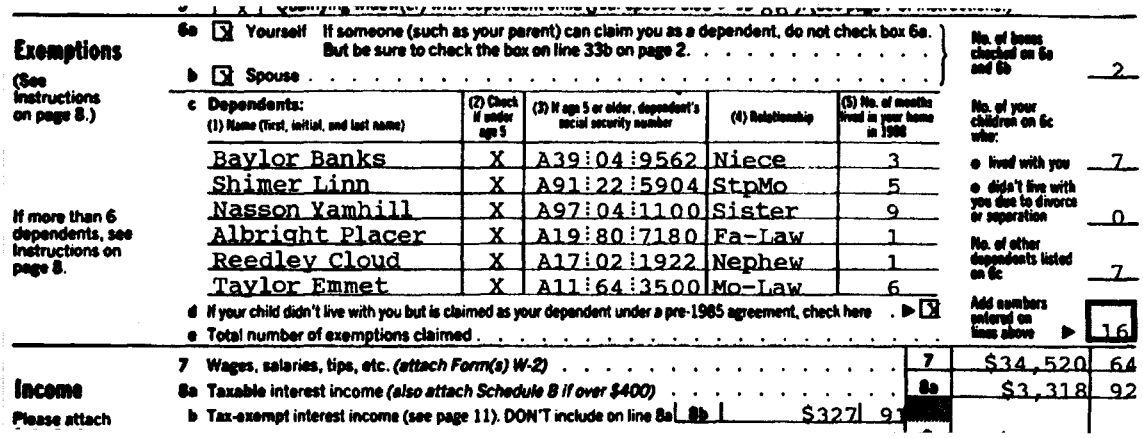


Figure 9: Results from using all 6 edge pairs.

shown whose entry fields have been filled in with machine printed data. The form in this example is representative of an entire image database produced by NIST. *NIST Special Database 2* (SD2) is a collection of IRS 1988 1040 Package X forms completed with machine printed data generated by a computer model in order to simulate *real* tax data.[3] Figure 11 shows the results obtained by running CURL on the form image using the line and corner edge pair definitions in Figure 6. Notice that the resulting image contains the line and grid structures of the form and is completely void of any font information. This is true even though the majority of the font data entered in the fields intersects these structures. The information in Figure 11 can be used to identify the form and it can be used to locate data entry fields on the form. Notice that CURL detected the corners of the boxes for lines 6a, 6b, and 6d.

Figure 12 shows a section of a form completed with hand print. The CURL results are shown in Figure 13. The form in this example is representative of an entire image database produced by NIST. *NIST Special Database 6* (SD6) is a collection of IRS 1988 1040 Pack-



Exemptions

(See instructions on page 8.)

If more than 6 dependents, see instructions on page 8.

6a ☒ Yourself If someone (such as your parent) can claim you as a dependent, do not check box 6a. But be sure to check the box on line 33b on page 2. . . . . No. of boxes checked on 6a and 6b 2

b ☒ Spouse . . . . . .

c Dependents: (1) Name (first, initial, and last name) | (2) Check if under age 5 | (3) If age 5 or older, dependent's social security number | (4) Relationship | (5) No. of months lived in your home in 1988

(1) Name	(2)	(3)	(4)	(5)
Baylor Banks	X	A39:04:9562	Niece	3
Shimer Linn	X	A91:22:5904	StpMo	5
Nasson Yamhill	X	A97:04:1100	Sister	9
Albright Placer	X	A19:80:7180	Fa-Law	1
Reedley Cloud	X	A17:02:1922	Nephew	1
Taylor Emmet	X	A11:64:3500	Mo-Law	6

No. of your children on 6c who: • lived with you 7 • didn't live with you due to divorce or separation 0

No. of other dependents listed on 6c 7

d If your child didn't live with you but is claimed as your dependent under a pre-1985 agreement, check here . ▶ ☒

e Total number of exemptions claimed . . . . . Add numbers entered on lines above ▶ 16

Income

Please attach

7 Wages, salaries, tips, etc. (attach Form(s) W-2) 7 $34,520 64

8a Taxable interest income (also attach Schedule B if over $400) 8a $3,318 92

b Tax-exempt interest income (see page 11). DON'T include on line 8a 8b $327 91

Figure 10: Results from using the c_{ul} edge pair.

Figure 11: Results from using the l_h edge pair.

age X forms completed with hand printed data generated by a computer model in order to simulate *real* tax data.[4] This example is extreme in that the hand print is large compared to the size of the fields, the hand print is very bold, and the hand printed data intersects frequently with the structures on the form. In light of this worse-case example, CURL performs very well. Most of the hand printed data is removed with only a few spurious vertical strokes and some salt and pepper noise remaining. The noise is primarily caused by the vertical stokes in the hand printed dollar signs.

4 Implementation Issues

In this section, issues related to implementation are discussed. The CURL algorithm is modular allowing many different functions for $f(r_i)$ to be used. Note that equation (3) can also be implemented as the dot product of the two edges as shown in Figure 5, and the values for $f(r_i)$ can be precomputed and stored in a look-up table. To date, no equation has been found to perform superior to equation (3) with an edge length of 32 when applied to cluttered forms scanned at 12 dots per millimeter. Also, the algorithm has been extended to detect com-

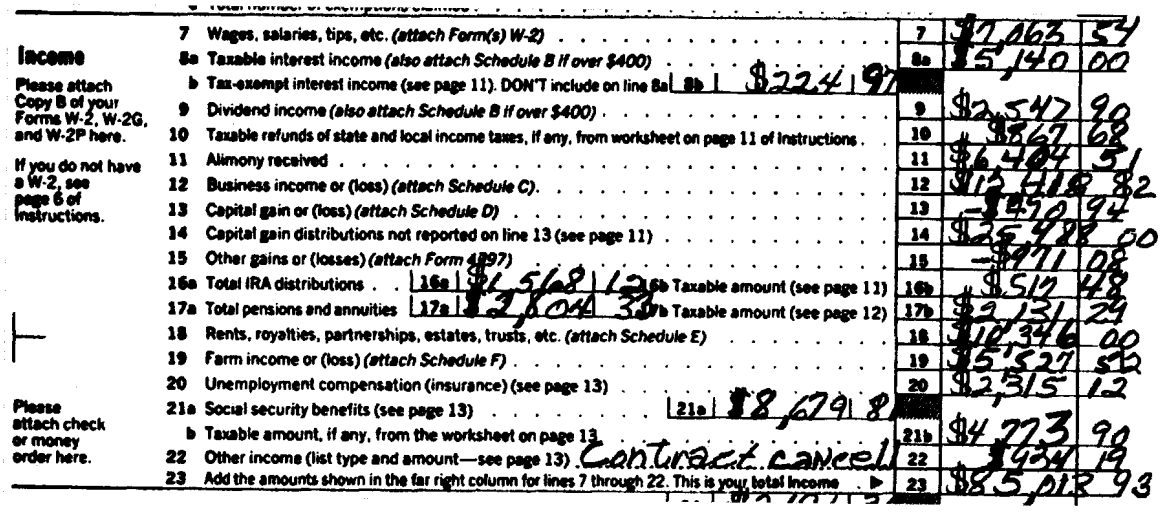


Figure 12: Results from using the c_{ul} edge pair.

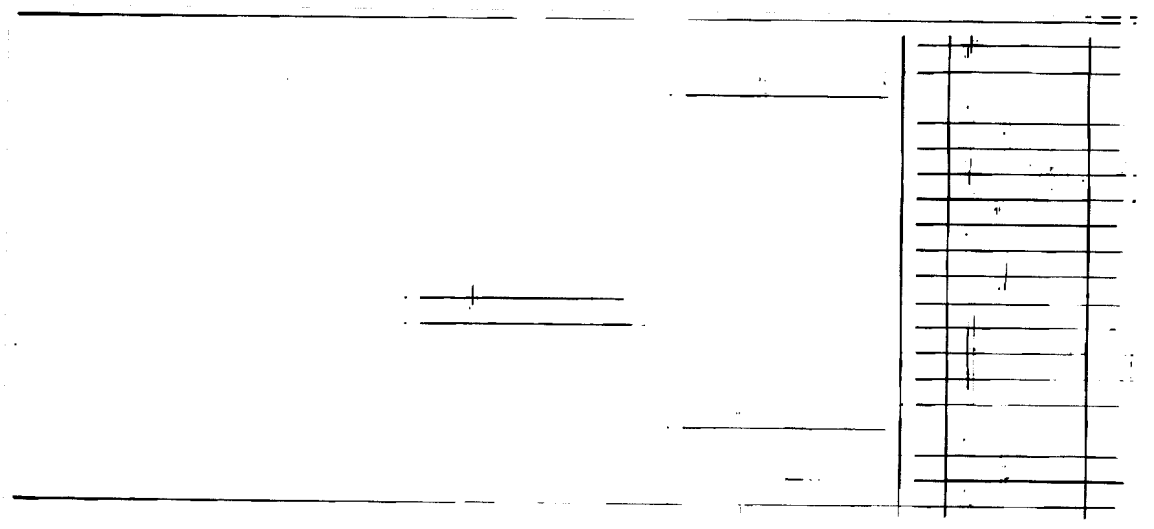


Figure 13: Results from using the l_h edge pair.

binations of line intersections including T's and crosses by computing the intersections of edge triples and edge quadruples. Edge pairs have also been developed for detecting circular shapes in an image.

This technique has several advantages over more conventional approaches. First, form structures are detected without any *a priori* knowledge of the specific form in the image. For example, no form template is required. Second, these structures are detected directly from the original image so that any distortions including translation, rotation, and scale are automatically handled. Therefore, global descewing and normalization of the image is avoided. One disadvantage of this algorithm is that a serial implementation is computationally expensive because results are computed and stored for every pixel in the input image. However, CURL is well suited for implementation in a highly parallel processing environment.

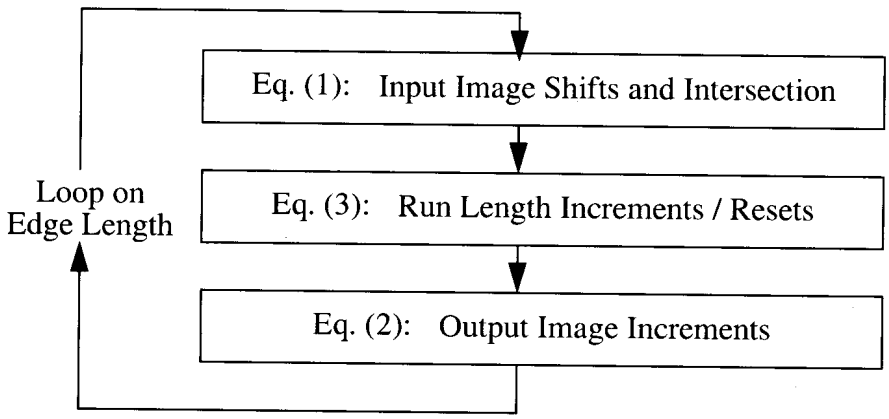


Figure 14: Parallel implementation of CURL.

The algorithm has been successfully implemented in serial on a Sun Microsystem's SPARCStation 2 and in parallel on a DAP 510c[5], a massively parallel Single Instruction Multiple Data (SIMD) computer manufactured by Cambridge Parallel Processing (CPP).[1] The serial implementation sequentially computes equations (1), (2), and (3) one pixel at a time. The parallel implementation distributes these equations across multiple processors so that parts of each equation are computed for all the pixels in the image at once. Equation (1) is reorganized so that one element from each pixels's logical vector, l_n, is calculated simultaneously. This is accomplished by a set of parallel image shifts followed by a parallel intersection. The run lengths, r_m, used in equation (3) are also distributed across the pixels, p_{ij}, so that each pixel has associated with it a current run length counter that is incremented or reset in parallel based on the results of the parallel intersection from equation (1). The output pixels, s_{ij}, in equation (2) are then incremented in parallel by the square of their corresponding run length counters. In order to calculate every element of every pixel's logical vector, l_n, the above steps must be repeated n times. The parallel implementation of CURL is illustrated in Figure 14. Currently, the parallel implementation runs 40 times as fast as than the serial version.

5 Algorithm Robustness

In the examples shown in this paper, CURL utilizes correlations of linear edge pairs to detect line segments, and CURL utilizes correlations of orthogonal edge pairs to detect corners. One would expect the performance of CURL to degrade as the image of a form becomes increasingly rotated. An experiment was conducted in which the entry field image shown in Figure 7 was perturbed by increasing degrees of rotational skew and then processed by CURL. The following figures show that the algorithm can tolerate significant rotational skew. CURL successfully detects both line segments and corners between +6 and -6 degrees of rotation as shown in Figure 15 and Figure 16. At rotations greater than 6 degrees, detected line segments begin to break up. However, the box corners are consistently detected even at +12 and -12 degrees of rotation as

1. Sun SPARCStation 2 and CPP DAP 510c or equivalent commercial equipment may be identified in order to adequately specify or describe the subject matter of this work. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the equipment identified is necessarily the best available for the purpose.

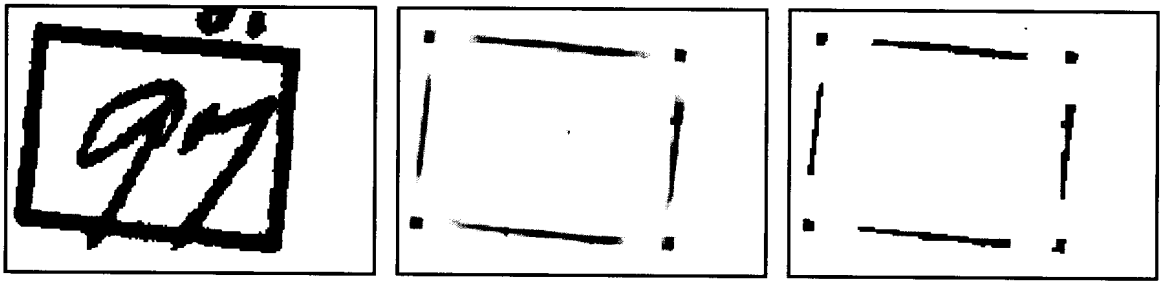


Figure 15: CURL performance when image is rotated 6 degrees clockwise.

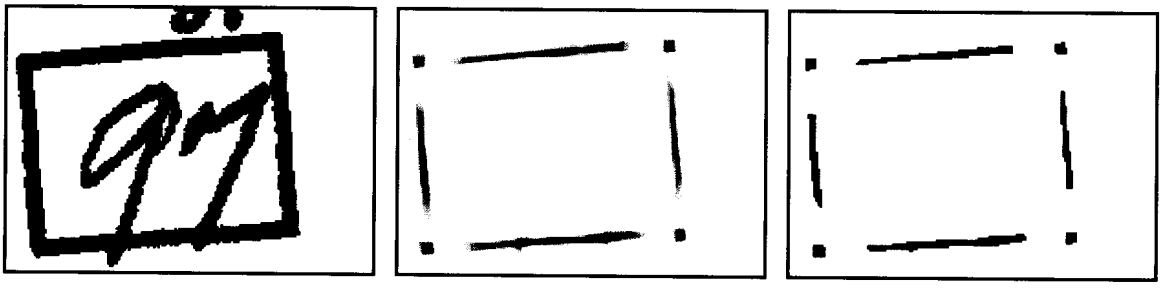


Figure 16: CURL performance when image is rotated 6 degrees counter-clockwise.

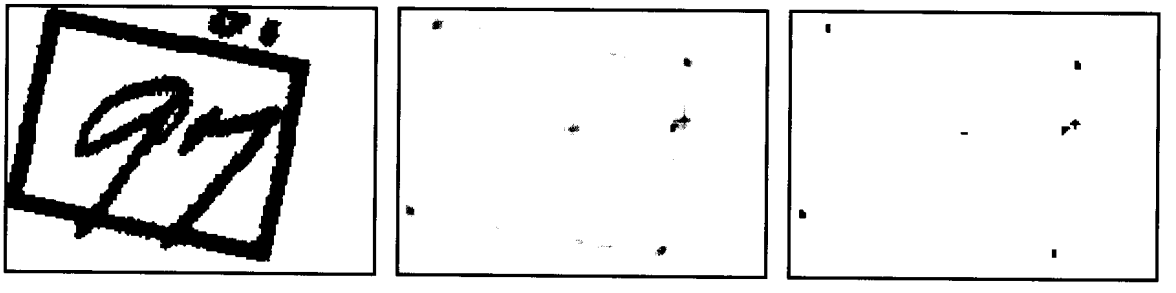


Figure 17: CURL performance when image is rotated 12 degrees clockwise.

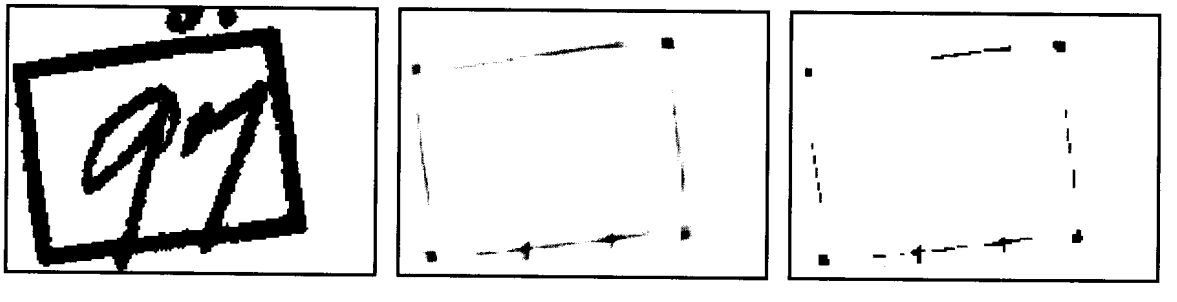


Figure 18: CURL performance when image is rotated 12 degrees counter-clockwise.

shown in Figure 17 and Figure 18. This clearly demonstrates how CURL gracefully degrades with increasing amounts of rotational skew.

6 Conclusions

An algorithm designed to detect intrinsic form structure within binary digitized documents has been presented. It has been shown that CURL automatically locates and extracts line segments and corners from images, overcoming the deficiencies of localized histograms and contrast kernels by using vector correlations which reward contiguity of pixels through the use of run length values. CURL extracts larger-scale shape-based structures, not contrast-based structures, and accurately distinguishes hand print strokes and machine print text from form structures. CURL is not limited to linear edges only. A point-to-point correlation can be computed between any two vectors representing any given shape and the points within each vector may be spaced apart from one another. Also, intersections may be computed on more than two edge vectors simultaneously, detecting T-shaped structures using three edges, and cross-shaped structures using four edges. This technique has several advantages over more conventional approaches in that form structures are detected without any *a priori* knowledge of the specific form in the image, and these structures are detected directly from the original image so that any distortions including translation, rotation, and scale are automatically handled. One disadvantage of this algorithm is that a serial implementation is computationally expensive because results are computed and stored for every pixel in the input image. However, it has been demonstrated that a parallel implementation of CURL on a massively parallel SIMD computer, CPP DAP 510c, runs 40 times as fast as than the serial version on a Sun Microsystem's SPARC-Station 2. It was also demonstrated in this paper that CURL gracefully degrades as increasing amount of rotational skew are added to the image. The algorithm detected both line segments and corners between +6 and -6 degrees of rotation, and the algorithm detected corners up to +12 and -12 degrees of rotation. In general, CURL is a shape-based feature detector. The algorithm was developed as a flexible front end to optical character recognition systems, extracting structures useful for automatically identifying form types and locating entry fields on forms.

References

[1] M. D. Garris et. al., "Massively parallel implementation of character recognition systems," in *Conference on Character Recognition and Digitizer Technologies*, Vol. 1661, SPIE (San Jose, California), 269-280, February 1992.

[2] H. P. Graf, C. Nohl, and J. Ben, "Image segmentation with networks of variable scale," *Advances in Neural Information Processing Systems*, Vol. IV, Morgan Kaufmann, (Denver Colorado), 480-487, December 1991.

[3] D. L. Dimmick, M. D. Garris, and C. L. Wilson, "Structured forms database, Technical Report Special Database 2," **SFRS,** National Institute of Standards and Technology, December 1991.

[4] D. L. Dimmick and M. D. Garris; Structured forms reference set 2, Technical Report Special Database 6," **SFRS2,** National Institute of Standards and Technology, September 1992.

[5] P. M. Flanders et. al., "Efficient high-level programming on the AMT DAP," in IEEE Proceedings: *Special Issue on Massively Parallel Computers*, 79(4):524-536, April 1991.