NISTIR 5687

Method and Evaluation of Character Stroke Preservation on Handprint Recognition

Michael D. Garris (mdg@magi.ncsl.nist.gov)

National Institute of Standards and Technology
Building 225, Room A216
Gaithersburg, MD 20899

ABSTRACT

A new technique for intelligent form removal has been developed along with a new method for evaluating its impact on optical character recognition. The form removal technique automatically detects the *dominant* lines in an image and erases them while preserving as much of the overlapping character strokes as possible. This method of form removal relaxes the recognition system's dependence on rigid form design, printing, and reproduction by automatically detecting and removing some of the physical structures (lines) on the form. The line detection and removal technique operates on loosely defined zones in which no image deskewing is performed. The technique was tested on a large number of randomly-ordered handprinted lowercase alphabet fields, as these letters (especially those with descenders) frequently touch and extend through the line along which they are written. It is shown that intelligent form removal can improve lowercase recognition by as much as 3%, but this *net* increase in performance is insufficient to understand the impact on the recognition. There is expected to be trade-offs with the introduction of any new technique into a complex recognition system. A new statistical analysis was designed to evaluate the impact of intelligent line removal on optical character recognition. This evaluation method compares the statistical distributions of individual confusion pairs between two systems and automatically determines the significant improvements and the significant losses in performance. In order for system developers to continue to reduce error rates, sophisticated analyses like this become necessary to understand the real impact a modification has on recognition performance. For example, this method of evaluation should be very useful in squeezing higher performances out of voting systems. The statistical analysis presented in this paper was used to evaluate the new line removal technique and the results are reported.

1. INTRODUCTION

The National Institute of Standards and Technology (NIST) has been actively conducting research in optical character recognition (OCR) and the automated processing of handprint written on forms since 1988.[1,2,3] This includes running several conferences sponsored by the Bureau of the Census that assessed the current state of the art in OCR technology.[4,5] As a result of these conferences, and in conjunction with our own research[6,7], it has been shown that computers are now able to read isolated handprinted characters as well as (or better than) humans. This being true, why are there not numerous general purpose image reading software packages on the market, shrink-wrapped and ready for immediate sale? It is because character classification is only one small piece of an end-to-end character recognition system.[8]

One large technical hurdle still facing the OCR community includes the automatic detection of both physical and logical structures in a document. This is termed *Document Image Understanding*.[9] For the purposes of this paper, the types of documents considered will be limited to forms for which people are requested to enter responses in predetermined fields. Physical structures on a form include the location of registration marks, instructional texts, lines and rules, entry fields represented by lines or boxes, and handprinted responses. Logical structures on a form include the reading order among the instructions on the form; the order in which the fields were filled out on the form; the type of information entered in each field, is it a name, an address, a dollar amount, etc.; and the relationships between the fields, for example, should fields sum together? Document recognition systems will become more like general purpose reading machines as these capabilities continue to advance.

Specific issues facing automated forms processing include conducting intelligent form removal and robust character segmentation. Realizing the need for continued progress in these areas, a project was formed to explore new methods of form removal. The form removal technique developed and described in this paper is just one of many alternative approaches[10], as a number of companies offer their own solution. It is our experience that most of the resources in the recognition community are invested in algorithmic design and development. However, an equally important area of research consists of evaluation methods for assessing and understanding the performance of complex recognition systems. NIST has made a concerted effort to integrate its algorithmic development with designing automated methods in which these algorithms can be more accurately evaluated and their impact understood.[11,12] The contributions of this study are therefore two-fold. A new form removal technique has been designed, and as a direct result of needing to evaluate this technique, a new statistical analysis has been developed with which two recognition systems can be more precisely compared.

Many current form removal techniques rely on identifying the type of form being processed so masks and zone templates (prestored physical and logical structures) can be retrieved to aide in processing the form. These systems require strict adherence to form printing and reproduction specifications, and they require considerable effort when adding new forms or modifying existing forms in the system. Once a form is identified, the image is registered so the form coincides with the retrieved masks and zone templates. The data comprising the form (including boxes, lines, and instructions) are erased from the image and the entered responses (often handwritten) are isolated. These are exactly the steps taken by the *NIST Form-Based Handprint Recognition System*, a public domain OCR package developed by NIST for transferring technology to industry and providing a baseline of performance from which other commercially available products may be evaluated and compared.[13] Copies of this system may be received free of charge by writing the author a letter of request on company letterhead.

Unfortunately for OCR systems, people have been trained from an early age to write along a line. Their writing will frequently intersect the line with some lowercase letters naturally extending all the way through the line. If form removal is not done carefully, portions of the characters overlapping the line will be chopped up, either causing holes in the middle of character strokes or completely removing entire pieces of the character all together.

To improve the NIST public domain OCR package, a technique was developed to automatically detect and intelligently remove some of the physical structures on a form (in this case, lines) so as to preserve overlapping character strokes (which improves the quality of character segmentation). The images in this study are scanned as binary (black and white). As a result, there is often high ambiguity about what black pixels represent. At any point in the image, a black pixel may represent only line data, only character data, or the overlap of both. These ambiguities, along with the challenges they create are discussed in Section 2.

A method of using the Hough line transform for detecting *dominant* lines in an image is presented in Section 3. This section also documents a technique for intelligently erasing the detected line from the image while preserving as much as possible the shape of overlapping character strokes. The line removal is based on computing a specific type of line width statistic and keying off of certain visual cues. Results are reported in Section 4 from a test designed to evaluate the impact of this new method on OCR. The NIST public domain OCR system was modified to incorporate the new form removal technique, and the system was run on a large number of handprinted randomly-ordered lowercase alphabet fields from *NIST Special Database 19*.[14] The performance statistics showed a 3% overall increase in recognition accuracy, but these global statistics only report the *net* impact on performance.

In *most* cases, a newly developed technique will improve the system, but with what trade-offs? It was necessary to design a new statistical analysis to study the local changes in performance between the old and new versions of the recognition system. This evaluation method compares the statistical distributions of individual confusion pairs between the two systems, so both the significant improvements and the significant losses in performance are detected and reported. This method of evaluation should be very useful in squeezing higher performances out of voting systems, in which the decisions from multiple classifiers are combined to improve recognition performance.[15] The statistical analysis was used to evaluate the new form removal technique and conclusions are drawn in Section 5.

2. CHALLENGES of ROBUST FORM REMOVAL

2.1 Examples of Visual Ambiguities

Examples of black pixel ambiguity are illustrated in Figure 1. At times it becomes very difficult to distinguish which black pixels represent character data, which represent line data, and which represent the overlap of both. This ambiguity can cause confusions as shown in examples (a - e). Is the character shown in example (a) an *a* or an *n*? In example (e), the characters shown were printed by the same writer. The character on the left is a *g*, and the character on the right is a *q*. The tail of the *q* is occluded by the line on the form causing both characters to look alike. In examples (f) and (g), it is difficult to determine which black pixels are part of the descenders of the lowercase characters, and in example (h), the bottom stroke of the *s* coincides with the line.

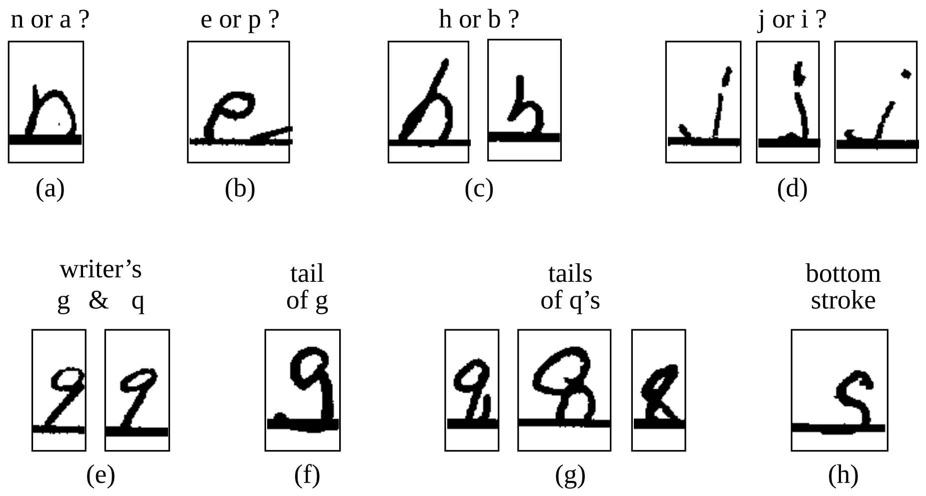


Figure 1. Examples of line and character stroke ambiguity in binary images of handprint.

Most of the examples shown in Figure 1 can be deciphered correctly by a human upon close inspection. A person resolves these ambiguities by comparing shapes from alternative classes of characters in conjunction with processing certain spatial cues within the configuration of black pixels. For example, we probably compare the difference in widths between the line and character strokes, and we compare the trajectory of the line with the shape of the characters. The techniques developed for intelligent form removal that are presented in this paper were motivated by these types of visual cues. Unfortunately, visual cues alone are not always enough to resolve the ambiguity, in which case other sources of contextual information must be applied. This information may include considering alternative classes of characters with similar shape and then resolving the alternatives through the use of a language or word model.[16] The use of these other contexts to help resolve ambiguities is left to a future study.

2.2 Automatic Form Structure Analysis

Another purpose of this study was to develop fundamental technology that contributes to the automatic analysis of form structures. A form structure includes such things as the geometrical layout of the fields on the forms, the spatial extent of each field, along with other logical and semantic relationships between the fields. The ultimate goal of this technology would permit the introduction of a new form into an automated forms processing system without requiring any adaptation or retraining of the system. Currently, most form recognition systems require the geometric layout of a form be known a priori. To accomplish this, the system first identifies the type of form being processed from a predetermined set of forms, and then the system retrieves the geometric data (masks and zone templates) corresponding to the identified form type. These systems are limited to processing forms that belong to the predetermined set, and they are completely dependent on the prestored geometric data.

Many forms processing applications have very loose controls over the quality of their forms. The same type of form may be typeset and printed by different printing contractors, on different paper stock, on different types of printing devices, and people filling out the form may even be permitted to return photocopies or facsimiles of the original form. These different factors contribute to variations in the geometric layout of the form. Rather than measure, categorize, and store each new variation of a form, it would be much more efficient to relax the dependence of the recognition system on this geometric data. Ultimately, all dependence on prestored form information would be eliminated, but this is virtually impossible since no standards yet exist for specifying how forms should be laid out and how fields on a form should be best represented for optical character recognition applications, although some studies have been conducted to help address these issues.[3,17]

In light of this, a form removal technique was designed to help reduce the dependence of a forms processing system on the geometric details of the form. The method described in the next section takes a loosely zoned binary subimage of a field and detects and removes all *dominant* horizontal lines in the subimage, while preserving the character strokes that overlap with the lines. Any field in which the writer was provided a horizontal line to enter a response can be processed by this method. The form's lines are removed and the handprint is isolated regardless of form type. Some global registration of the form is needed to produce loosely defined zones, but no precise pixel coordinates of the field's size and location are required and no prestored image mask of the form is required. In this way, a form's geometry is still needed, but the dependence on this information has been greatly relaxed.

3. INTELLIGENT LINE DETECTION AND REMOVAL

The form removal technique described in this paper removes all dominant horizontal lines from a binary image. The removal of a line is done intelligently so as to preserve any part of a handprinted character that overlaps the line. Previous techniques would simply erase the line, typically by applying a mask or template of the form. If characters overlapped the form information, this data was also erased, leaving gaping holes in the middle of character stokes (especially descenders). A postprocess was then required to fill the resulting holes. Detection of these holes can be accomplished by dilating the mask and logically ANDing it with the erased image. Candidate holes are located where the black pixels in the dilated mask overlap with the black pixels in the image. The challenge is not so much in the detection of holes, but rather in determining how the holes should be accurately filled.

Realizing the repair of character strokes would be very difficult and prone to error, a technique for intelligent line removal was developed where character strokes are preserved simultaneously in conjunction with the form removal. The technique involves two steps, the detection of dominant lines and the intelligent removal of those lines.

3.1 Line Detection Using the Hough Line Transform

The Hough transform[18] deals with the detection of specific structural and shape relationships between pixels in an image. For example, the Hough line transform[19] is an elegant template matcher that determines the number of possible lines that fit at least some of the pixel data in an image. For comparison, consider a crude line detector constructed by defining a set of matrix templates, each containing a single pixel wide black line at a different orientation. Each of these templates is windowed across the image, and at each step, the amount of coincident black pixels between the template and the image are counted and stored. The same process is repeated for each of the line templates in the set, and locations in the image that incur a high pixel coincidence are determined to contain the associated line. This type of line detection works well for many applications, however it is inefficient because every pixel in the image has to be repeatedly searched for every line in the template set.

The Hough line transform avoids this repetitive application of successive templates by constructing a parameterized (voting) matrix representing all the possible lines that may reside in the image. The image data is searched only once, and at each black pixel in the image the question is asked, “Given this point, what are the possible lines that can pass through this point?” Each one of the possible lines is represented by an addressable cell that is incremented in the voting matrix. As black pixels are encountered in the image, cells in the voting matrix are updated accordingly. By the time a single pass through the image is complete, the voting matrix has recorded all the possible lines in the image. The more black pixels lying along a specific line, the greater the corresponding cell value in the matrix. Dominant lines in the image can therefore be systematically detected by simply finding the largest values in the matrix.

3.1.1 Solid Line Detection

This raises the question, “How are lines parameterized and used to address this voting matrix?” One possibility would be to use the slope-intercept form of a line in which one axis in the matrix would represent possible slopes and the other axis would represent corresponding y-intercepts. A problem with using the slope-intercept parameterization occurs when a line approaches a vertical orientation, in which case the slope and intercept approach infinity. To avoid discontinuities, the normal representation of a line listed in Equation (1) is used instead.

$$\rho_j = x\cos\theta_i + y\sin\theta_i \qquad (1)$$

Figure 2 (a) illustrates a line and its corresponding parameters in normal representation form, (θ_i, ρ_j). The matrix on the right shows the associated Hough line transform voting matrix. Given a black pixel in the image at position (*x, y*), θ_i is incremented between θ_{min} and θ_{max} according to a predefined step factor, and the corresponding ρ_j's are solved according to Equation (1). For each (θ_i, ρ_j) pair, the representative cell in the voting matrix is incremented by one.

A goal in this study was to detect all dominant *horizontal* lines in the image. To accomplish this, the Hough line transform detection utility was designed so θ_{min} and θ_{max} could be specified using as a single value, $\pm d$ degrees. This argument limits the range of angles in which lines are considered. However, the positive and negative values of this argument must be converted to normal representation, *-d* must be mapped to (90° - *d)* and *+d* to (*d* - 90°*)*. So in order to search for lines in the range ±5 degrees, two Hough Transform voting matrices are required, one matrix with θ_{min}= 85° and θ_{max}= 90°, the second matrix with θ_{min}= -90° and θ_{max}= -85°. An increment of 1° was used to sample across these two sets of θ ranges.

The other dimension of a Hough line transform voting matrix represents ρ, the length of the normal vector that intersects the given line and passes through the origin. The range of possible ρ's is determined by the dimensions of the image being processed. A normal vector could potentially be extended from the origin of the image to any point in the image space. Plus, the length of ρ can be positive or negative. Therefore, ρ_{min} is set to the negative of the image's diagonal length, and ρ_{max} is set to the positive diagonal length. These limits on ρ are an upper bound. Tighter ranges can be imposed by taking into account the limitations placed on θ.

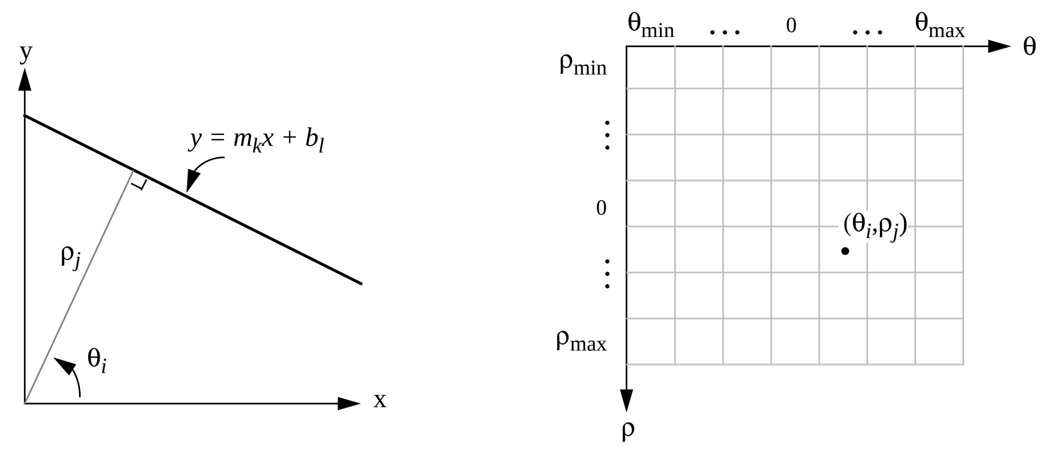


Figure 2. Line parameters in normal representation form (a) and the Hough Transform voting matrix (b).

Upon one complete pass through the image, the two voting matrices are searched to determine where the dominant horizontal lines are. The pseudocode in Figure 3 describes how a dominant line is selected. Given the voting matrix with negative θ's (NegMx) and the matrix with positive θ's (PosMx), the top-*n* (for this study n=5) candidate lines are chosen. The voting matrix is a discrete representation of the (θ_i, ρ_j) parameter space, and lines in a binary image are typically much wider than a consistent theoretical line width of one pixel. The rounding of actual θ's to discrete cell's in the matrix in conjunction with lines in the image having a pixel width greater than one contribute to small ambiguities in the voting matrix. At times a single line in the image may be closely represented by more than one cell in the voting matrices. This creates a number of different implementation challenges. First, the votes within a single cell of the matrix (in general) reflect the dominance of the line, but they do *not* reflect the pixel length of the line. Neighboring pixels will frequently map into the same (θ_i, ρ_j) cell, but the geometric position of the extra pixels contribute to the width of the line, not to the line's length. A second challenge is, once a line has been detected and erased from the image, other cells representing a very similar line should not be considered because a significant number of their pixels will already have been removed from the image.

```
Given NegMx and PosMx
Find Top 5 Candidate Lines
    Choose max from NegMX and PosMx
    Convert max ($\theta_i$, $\rho_j$) to slope and intersection point
    Compute where line intersects edges of the image
    Count black pixels along line trajectory
    Store left-most black pixel and right-most black pixel
end find
Choose line from set of candidates with max black pixels
Set corresponding ($\theta_i$, $\rho_j$) cell in appropriate Mx to zero
```

Figure 3. Pseudocode for dominant horizontal line selection.

To handle ambiguities in the voting matrices, the top-*n* combined candidates from the positive and negative θ voting matrices are considered for the detection of a dominant horizontal line. The five cells with the most votes are selected, and the slope and normal vector intersection point are computed for each candidate line from their corresponding (θ_i, ρ_j) parameters. Given a (θ_i, ρ_j) pair, the slope *m* of the line is computed as

$$m = \tan(90° + \theta) \tag{2}$$

and the normal vector intersection point (x_n, y_n) is calculated

$$x_n = \rho\cos\theta \tag{3}$$

$$y_n = \rho\sin\theta \tag{4}$$

Some of the implementation details not discussed in this paper include ensuring equivalent values of θ are consistently represented in quadrants appropriate for system-provided trigonometric library functions. The method described in this paper ensured all θ's were consistently represented by their equivalent angles in quadrants I and IV (-90° to +90°). Also not mentioned are the details on how (θ_i, ρ_j) pairs were mapped to discrete (*i*, *j*) cells in the matrices. This involves keeping records of the minimum and maximum values for each dimension of the voting matrices in conjunction with taking into account the step size used to sample the range of θ's, and offsetting values of ρ according to twice the diagonal length of the image. These implementation details and others are left as an exercise to the reader.

Using simple line geometry, the points where each candidate line intersects the edge of the image are computed, and the number of black pixels along each line trajectory are counted. In addition, line lengths are computed as the distance between the left-most and right-most black pixels along each line trajectory. In the end, the candidate line with the greatest number of black pixels is selected and the line's (θ_i, ρ_j) cell in the appropriate voting matrix is set to zero, so the cell will not be selected again. If on a subsequent dominant line selection, a (θ_i, ρ_j) cell with maximum value turns up to have no black pixels along its line trajectory, then the line must have been redundantly represented by a cell already processed. In this case, the current (θ_i, ρ_j) cell is appropriately set to zero and the search for the top-*n* candidates continues. This search terminates prematurely if the total vote *v* for the next maximum (θ_i, ρ_j) cell falls below half the image width *w*, $v < (0.5 \times w)$. If this happens, then only those candidates already found are considered for dominance selection.

Line selection continues according to the algorithm in Figure 3 as long as two dominance criteria are met. The first test is based on the length *l* of the detected line as compared to the number of black pixels *b* counted along the line trajectory. If $b < (0.75 \times l)$, then the test for dominance fails and line selection terminates. The second test is based on the length *l* of the detected line as compared to the overall width *w* of the image. If $l < (0.5 \times w)$, then the test for dominance fails and line selection terminates. The constants used in the dominance criteria have been tested over a broad range of field types, however they may require modification for other applications.

3.1.2 Dashed Line Detection

A second technique was designed for detecting dashed lines in an image. As with solid lines, the dashed line method uses the Hough line transform. The top-*n* (θ_i, ρ_j) cells in the voting matrix are used to direct the search for dominant dashed lines, but unlike solid lines (which were selected based solely on the number of black pixels in the line), dashed lines are selected by analyzing the regularity of black and white pixel runs along a given line trajectory. In general, a dashed line will exhibit a very regular pattern of black pixels followed by white pixels followed by black pixels, and so on. To measure this regularity, the length of the runs of black pixels and the length of the runs of white pixels are measured, and a standard deviation is computed for each of the two sets. These standard deviations are then combined to compute a score by which dominant dashed lines are selected.

The formula for deriving the score is based on two criteria that measure how small both the black and white pixel run length standard deviations are. The first criterion measures how *small* the standard deviations are relative to the other candidate lines. The smaller a line's standard deviations are compared to the other candidates, the more regular are the particular line's dashes and intervals between the dashes. Given the maximum standard deviation σ_M (whether black or white) from the entire list of candidate lines, smallness *s* is calculated as

$$s = \frac{(\sigma_M - \sigma_m)}{\sigma_M} \qquad (5)$$

where σ_m is the larger of the current line's black σ_b or white σ_w pixel run length standard deviations. As the current line's σ_m becomes increasingly small, *s* approaches 1.0. As σ_m becomes increasingly large and approaches σ_M, *s* approaches 0.0. The standard deviations are computed according to Equation (8).

The second criterion measures how *close* a line's standard deviations are relative to each other. Both standard deviations (white and black) must be equally small to represent a dashed line. It was observed that a line trajectory passing through handprinted text exhibits regular black run lengths because the writing has a fairly consistent stroke width, however the lengths of the white intervals between the characters fluctuates much more. To distinguish a dashed line from a line trajectory passing through handprint, both how small and how close the black and white pixel run length standard deviations are to each other must be considered. In light of this, closeness *c* is calculated as

$$c = 1.0 - \frac{|\sigma_b - \sigma_w|}{\sigma_m} \qquad (6)$$

The average of the two results in Equations (5) and (6), $a = (s + c)/ 2.0$, is used for the final score of each candidate line. The line with the maximum score is selected as the dominant horizontal dashed line. Dashed line selection continues according to the algorithm in Figure 3 (with selections based on the *a* scores) as long as two dominance criteria are met. The first test is based on the length *l* of the detected line as compared to the number of black pixels *b* counted along the line trajectory. Due to a dashed line having significantly fewer black pixels than a solid line, the dominance factor used for this test is reduced from 0.75 (used for a solid lines) to 0.1. If $b < (0.1 \times l)$, then the test for dominance fails and line selection terminates. The second test is exactly the same as the one used for solid lines. If $l < (0.5 \times w)$, then the test for dominance fails and line selection terminates, where *l* is the length of the detected line and *w* is the width of the image.

3.2 Intelligent Line Removal

Upon detection, each dominant horizontal line in the image is carefully removed so the character data in the image is preserved. This requires an ability to distinguish line data from intersecting and overlapping character data. As shown in Section 2, there can be a significant amount of ambiguity in binary images because black pixels can represent part of the line, part of the character, or both overlapping each other. Humans resolve these ambiguities amazingly well, and one way they accomplish this is through careful analysis of visual cues. The technique developed for this study tries to mimic this ability.

The method begins by processing what is known and predictable, and based on the initial processing, further refines itself. At first, the only known and relatively predictable structure in the image is the detected line. To begin from the location of plausible characters would be full of pitfalls as character locations, sizes, and shapes can vary greatly. We do, however, have a reasonably good description of the line trajectory in the image, and we know the line has a relatively uniform width. It is with this knowledge that initial assumptions are drawn and the processing begins.

The first step is to get a good estimate of the width of the line in the image. The detected lines are relatively horizontal, which enables them to be represented as a sequence of vertically oriented slices (the width of one pixel) that when sandwiched all together form a line. The position and length of each one of these slices can be measured and stored by traversing the detected line trajectory, and at each successive horizontal position, computing the length of the vertical run intersecting that point on the line. The location of the top-most black pixel, the bottom-most black pixel, and the intervening distance (height) for each slice on the line are computed and stored. In theory, the slices will all be of equal height, but small fluctuations are common due to the varying quality of the source document and scanner jitter. The most pronounced fluctuations in slice heights occur when other data, such as characters, intersect the line. There is often an observable discontinuity in the heights of the slices as black pixels from the character are merged with the black pixels of the actual line. These large fluctuations occur relatively infrequently across the length of the line, so a majority of the line slices do represent the true width of the line. Operating under this assumption, the estimated width of the line is computed as the median slice height along the entire line (between the left-most and right-most black pixels along the trajectory). Figure 4 illustrates how slice heights fluctuate with the intersection of a character.

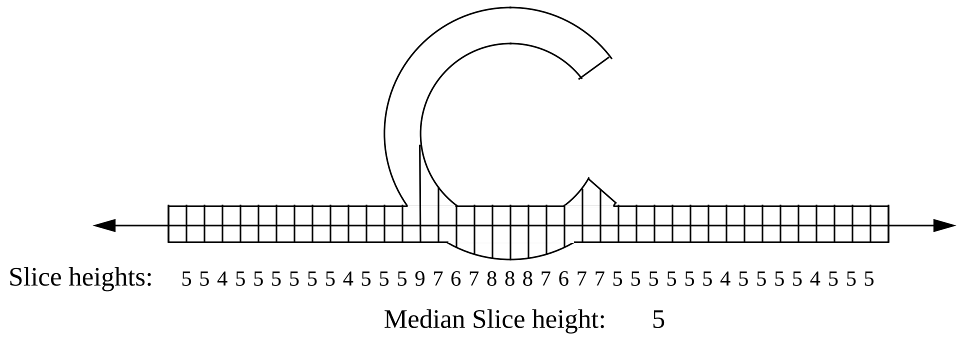

Slice heights: 5 5 4 5 5 5 5 5 5 4 5 5 5 9 7 6 7 8 8 8 7 6 7 7 5 5 5 5 5 5 4 5 5 5 5 4 5 5 5
Median Slice height: 5

Figure 4. Illustration of slice heights along a line trajectory.

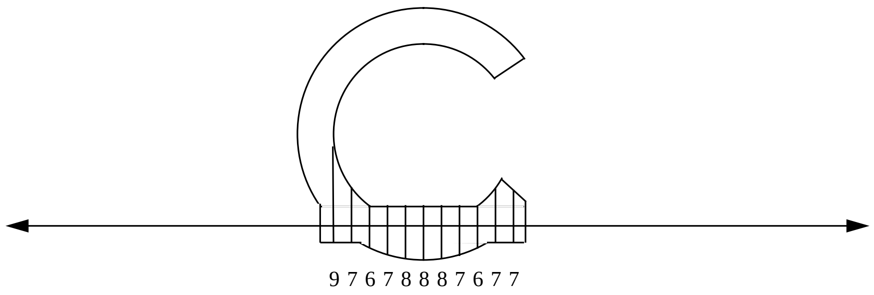


Figure 5. Result of removing slices less than or equal to the median slice height.

Once the estimate for the line width has been computed, all slices less than or equal to the median slice height in length are erased from the image. In the example shown in Figure 4, the median slice height is 5. All slices with height less than or equal to 5 are removed from the image and the result is shown in Figure 5. The remaining slices need further processing in order to separate the line data from character data.

3.2.1 Fuzzy Line Segment Removal

Removing all slices less than or equal to the median slice height performs exceptionally well when the quality of the scanned document image is very high. Unfortunately, image quality can vary significantly in many applications due to the fluctuations in the quality of source documents and the reflectance properties within the optical performance of scanner’s. It has been observed in our database collections that the width of a line, when scanned, can vary considerably across the length of the line, even when the line is clearly printed with a uniform width. At times a line may gradually thicken, and at other times, it may narrow. If these fluctuations exceed the median slice height, then entire sections of residual line data will remain. To account for these small but gradual deviations, a method for detecting and removing segments of *fuzzy* lines was developed.

```
ARGUMENTS:
A1. a vector of slice statistics including their tops, bottoms, heights,
    and erasure status
A2. the median slice height from all the slices

ALGORITHM:
for each slice in the vector slices (A1)
  Select one of the following steps that apply:
    1. If at the end of a run of unerased slices …
    2. If in the middle of a run of unerased slices AND
       the current slice’s height is too long (C1 or C2) to contain
       only line data …
    3. If the current slice’s height is small enough (C1 or C2) to
       contain only line data …
    Otherwise, continue to the next slice
end for

STEP 1:
If at the end of a run of unerased slices …
  Select one of the following steps that apply:
    1.1 If run began from the point of an erased slice, then
        erase the run of slices.
    1.2 If run began from a piece of character, then (be restrictive)
        erase the run of slices only if the length of the run is long
        enough (C3) to compute and test minimal line statistics.

STEP 2:
If in the middle of a run of unerased slices AND
  the current slice’s height is too long (C1 or C2) to contain only
  line data …
  If run began from a piece of character, then (be more restrictive)
    If the run is long enough (C4) to have reasonably accurate
       line statistics for validating the run of slices as a line AND
       there is a very small amount of fluctuation (C5) among the
       heights of the slices in the run, then
         erase the run of slices.
  If run began from the point of an erased slice, then
    don’t do anything and continue to the next slice because
    character strokes often merge continuously with the line, and
    erasing a run in this case risks erasing character data.

STEP 3:
If the current slice’s height is small enough (C1 or C2) to contain
  only line data …
  Select one of the following steps that apply:
    3.1 If not in the middle of a run of slices to be erased, then
        start a new run and initialize run’s slice statistics.
        If the current slice’s left neighbor is erased, then
          use the looser tolerance (C1) for testing the heights of
          future slices in the run.
        Otherwise
          assume the run begins from a piece of character and
          use a tighter tolerance (C2) for testing the heights of
          future slices in the run.
    3.2 If in the middle of a run of slices to be erased, then
        update the current run’s slice statistics.
        If the slices in the run fluctuate more than a fair amount
          (C6), then
          - restart the run from the current slice.
          - use a tighter tolerance (C2) for testing the heights of
            future slices in the run because the new run begins
            from a piece of character.

CONSTANTS:
C1. a looser tolerance on slice heights based on the median slice
    height (A2). C1 = max(A2 + 3, A2 * 2.0)
C2. a tighter tolerance on slice heights based on the median slice
    height (A2). C2 = max(A2 + 3, A2 * 1.5)
C3. a minimum run length required to get and test line statistics.
    C3 = 3
C4. a longer run length required to get “accurate” statistics for vali-
    dating a run to be a line. C4 = A2 * 3.0
C5. a very small amount of fluctuation among unerased slices in a
    run. C5 = 1
C6. a fair amount of fluctuation among unerased slices in a run.
    C6 = 2
```

Figure 6. Algorithm for removing *fuzzy* line segments.

The algorithm (listed in Figure 6) continues to analyze and process the unerased slices along the line trajectory. The list of remaining slices are processed left to right, with contiguous slices that exhibit line-like statistics being gathered into groups, called *runs*. There is at least one core strategy incorporated in the algorithm. This takes into account whether a run starts and ends at the point of erased slices or from slices containing character data. When a run starts or ends at a piece of character data, tighter tolerances and stricter tests are applied to determine if a relatively uniform run of slices contains only line data and therefore should be erased. In comparison, the criteria used to erase a run of relatively uniform slices that begins and ends at points of erased slices is much more relaxed because it is very unlikely the run contains any character data whatsoever. The tightening of tolerances includes changes in the limits on slice height, amount of fluctuation within a run of slices, and the length of the run. The length is an important criteria because the longer the contiguous slices maintain relative uniformity, the more likely the slices are only part of the line. The dynamic adjusting of these tolerances defines fuzzy line width boundaries that permit smooth fluctuations around the median slice height. The algorithm also attempts to preserve as much of the character data as possible.

3.2.2 Filling Character Voids

Even with all the careful adjusting of tolerances in the fuzzy line segment removal, there can still be unresolved ambiguities in the image. Using the line slice techniques described above, the chance of these ambiguities increases as the width of the character stroke becomes increasingly thin. For example, study the illustration in Figure 7, where a curved character stoke is shown overlapping a line. This particular configuration could represent the bottom of a C or an O. The width of the character stroke is smaller than the width of the line, and the regions shaded in gray represent slices already erased by the line slice removal methods (based on median slice height and fuzzy line segments). In this example, the bottom of the character stroke is completely enveloped by the intersecting line, so upon erasing line slices, the character stroke is inadvertently cut in two, leaving a void shown as the middle gray region in the figure. Two different methods were developed to detect and refill these types of voids.

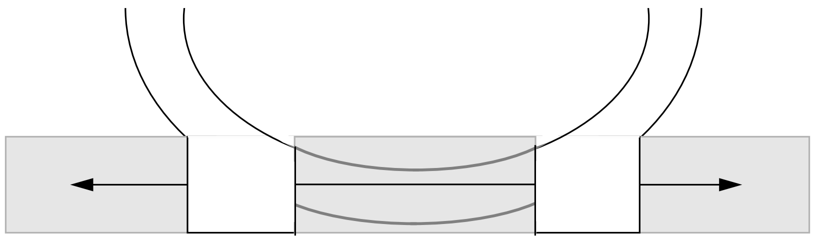


Figure 7. Part of an overlapping thin character stroke erased.

The first method takes into account the fact that the shorter the length of a void, the more likely two neighboring character pieces belong together and the void should be filled. To fill a void, the erased slices comprising the void are simply redrawn in the image. The method also looks for compatibility between the two neighboring character pieces. Each contiguous group of *erased* slices is gathered into a run, and then each run is processed independently as a potential void. The limit used to determine if a void is short enough is based on the median slice height of the line. Specifically, a limit of $l_s = \max(8, 8 \times m)$ was used, where l_s is the resulting limit and m is the median slice height. If the length of a run is short enough, then each edge of the run is tested for compatibility. Adjacent to the run's left and right edges are slices that have not been erased and are assumed to contain pieces of characters. If the heights of the left and right unerased slices are both within the limit $l_c = m + 2$, then it is likely the void contains character data, and it is filled.

The second method of filling voids is more complex. Voids caused by character curvatures overlapping the line, like the one shown in Figure 8, often exceed the first method's limit l_s. To detect and fill these long voids, it becomes necessary to analyze the shape characteristics of the neighboring character pieces. This time contiguous groups of *unerased* slices are gathered together into runs, and neighboring runs are compared to each other. First, their interior vertical edges (V_l and V_r in the figure) are tested to make sure the intervening void passes completely through the neighboring character pieces. The erased slice (V_l) adjacent to the right side of the left character piece is searched above and below for any reasonably close character data. The same is done for the erased slice (V_r) adjacent to the left side of the right character piece. Next, the bottoms of the character pieces (H_l and H_r) are tested to make sure no character data extends through the bottom of the line. If these vertical and horizontal edges are clear of character data, then it can be deduced that the character stroke comprising each of the neighboring character pieces begins above the line and extend downwards intersecting the line without protruding through the bottom of the line.

At this point it is necessary to examine the top interior contours of the two character pieces to determine if in fact a curvature exists and whether the void should be filled. In Figure 8, line projections L_l and L_r are used to approximate the top interior contours. Projection L_l is computed by tracing the left character piece's contour, starting at the top right edge (V_l), and tracing the contour 6 steps upwards and to the left. The 6 pixel steps in y are divided by the accumulated change in x, and a slope is computed and used to construct the projection. The same process is repeated for the right character piece and the projection L_r is computed. Given L_l and L_r, the two character pieces are assumed to comprise a curvature and the intervening void is filled if *all* the following criteria hold:

1. L_l is horizontal or has positive slope and L_r is vertical or has negative slope. (Remember in image space, the origin is in the top left corner and positive values of increasing y extend downward.)
2. One projection is not *perfectly* vertical while the other projection is *perfectly* horizontal. (This technique has been tailored for handprint in which case such exactness is unlikely to naturally occur within the formation of a character.)
3. The length of the intervening void must be less than the limit $l = \max(10, 5.0 \times m)$. (This avoids filling ridiculously large voids between two character pieces most likely *not* part of the same character.)
4. L_l and L_r intersect at a point within the region of the intervening void. (There must be reasonable convergence of the two projections for the two character pieces to be joined together.)

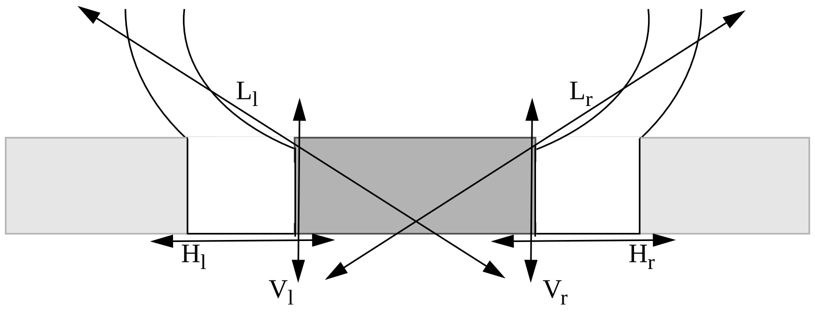


Figure 8. Detecting and filling a long void in a character curvature.

3.2.3 Corner Detection and Removal

As can be seen in the illustrations, after line slices have been erased and any character voids filled, the remaining slices contain a mixture of black pixels belonging to the line and black pixels belonging to the character. Notice the corners protruding downwards from each side of the character. These corners contain black pixels belonging mostly to the line. The next step in the line removal method attempts to locate and clip these corners off. This helps shape the character strokes that intersect the line. There are four different corner orientations, one for each corner on a rectangle. Once the logic for cutting away one corner has been derived, it can be easily adapted to remove any of the other three orientations.

In order to locate these corners, a number of categories have been defined that represent the majority of frequently occurring configurations of characters intersecting with the line. Contiguous line slices remaining in the image are grouped together and categorized into one of the eight different categories illustrated in Figure 9.

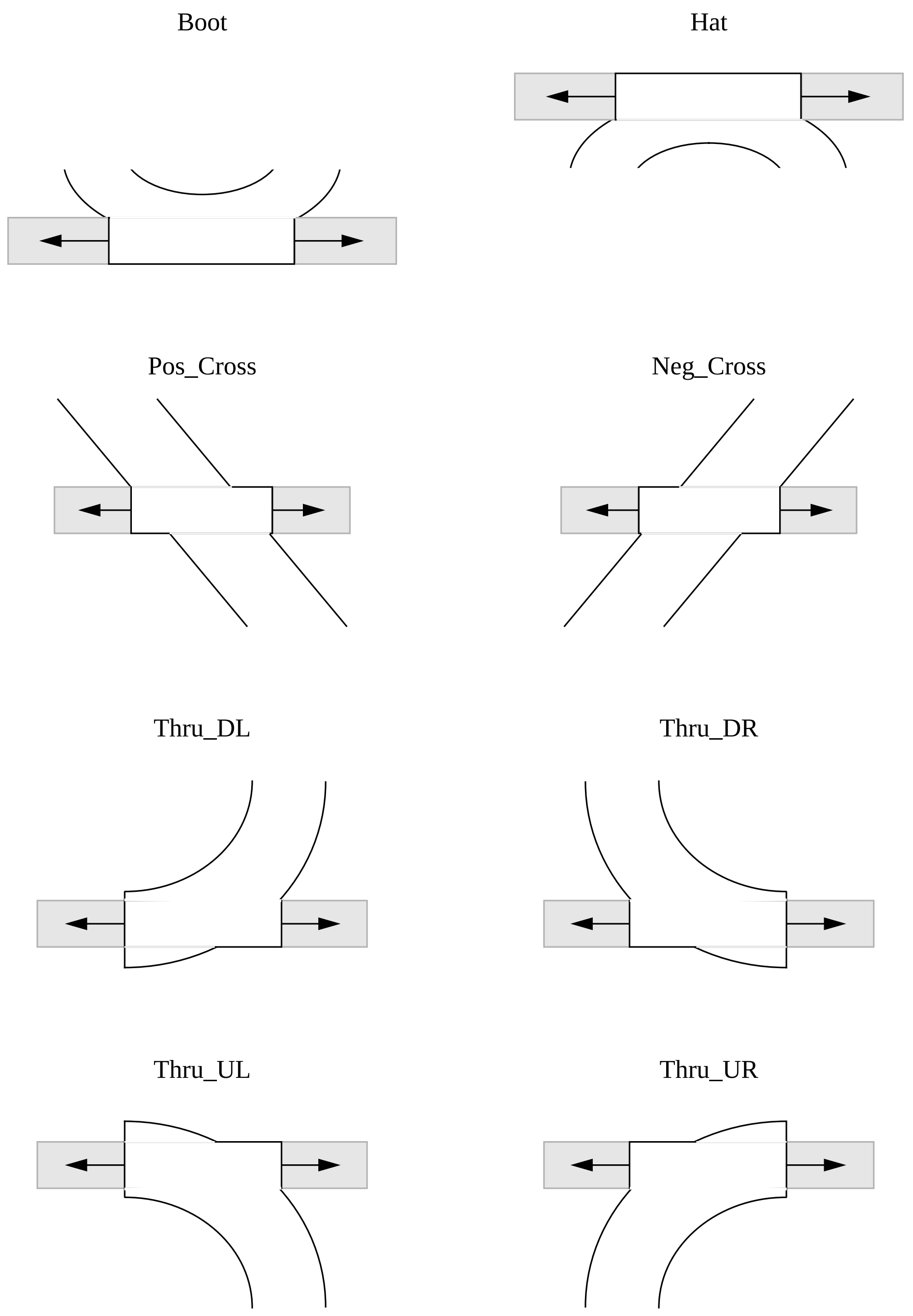


Figure 9. Eight general categories of intersecting line and character stroke configurations.

The first two categories, *boot* and *hat*, represent instances where characters such as O's or C's touch the line on the bottom or top respectively. The next two categories, *pos_cross* and *neg_cross*, represent instances where descenders of characters such as j's and y's pass completely through the line on the form. The difference between these two categories is the slope with which the character stroke crosses the line. The remaining categories represent character strokes that intersect the line, but only pass completely through on one side. In comparison, pos_cross and neg_cross categories pass completely through the line on both sides. The category *thru_dl* (standing for "through and down to the left") represents characters such as S's or J's that touch the line on the bottom. The category *thru_dr* (standing for “through and down to the right”) represents characters such as c’s or e’s that touch the line on the bottom. The category *thru_ul* (standing for “through and up to the left”) represents characters such as D’s or Z’s that touch the line on the top. Finally, the category *thru_ur* (standing for “through and up to the right”) represents characters such as E’s or F’s that touch the line on the top.

Each of the eight categories has one or more corners containing residual line data that should be clipped from the character. As mentioned above, all four corner orientations (top-left, top-right, bottom-left, and bottom-right) exist among these configuration of character strokes. A boot contains bottom-left and bottom-right corners, a hat contains top-left and top-right corners, a pos_cross contains bottom-left and top-right corners, a neg_cross contains top-left and bottom-right corners, and so on. By studying the illustrations in Figure 9, you can observe that every corner has one edge adjacent to the position of an erased slice. The position of the erased slice defines one point from which the corner should be clipped. If a second point along the other edge can be located, the corner is cut off by interpolating a line between the two points. If the second point cannot be located, the slope of the cut line is estimated from contour statistics near the first cut point. The corner is then clipped according to a line projected from the first point. A couple of examples are given for a more detailed explanation.

In the case of pos_cross and neg_cross strokes, the second edge of the corner can usually be found by searching the contour of the overlapping line and character data for an inflection point. Figure 10 illustrates the clipping of a bottom-left corner on a pos_cross. The left vertical edge of the corner is adjacent to the last slice erased from the line. The top of this erased slice marks the first point (labeled point A) from which the corner should be cut. The second cut point (labeled point B) is located by traversing the bottom horizontal edge of the corner starting at the bottom of the erased slice and searching right until the straight line runs into the edge of the character stroke. At this point, the contour will typically make a subtle bend (or inflection). This inflection point is detected by accumulating the aggregate change in the second derivatives along the contour. Given an array *y* that contains the y-coordinates of the bottoms of the unerased slices, the second derivatives can be accumulated by summing the values

$$d_i = -y_{i-1} + 2y_i - y_{i+1} \qquad (7)$$

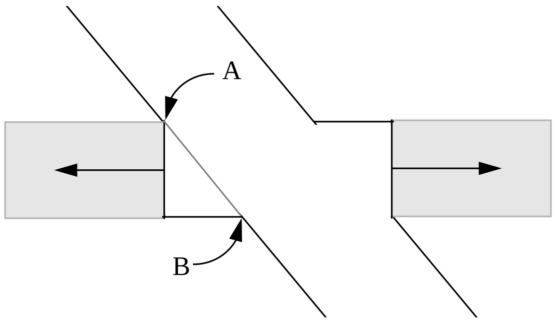


Figure 10. Cut points for the bottom-left corner on a pos_cross character stroke.

Second derivatives are used in place of first derivatives in order to relax the method’s dependence on deskewing. By examining the second derivative statistics, the image is permitted to contain some rotational distortion that will cause the y-coordinates along the bottom of the line to have slope other than zero. In our study, an inflection point was detected whenever the accumulated second derivatives exceeded one. Once the two points (A and B) are located, a line is interpolated between the two points (represented as the dashed line in the figure), and the black pixels below and to the left of the line are erased. These pixels are assumed to be part of the line, not part of the character.

At times, the line and character data merge so smoothly that no inflection point can be reliably detected. This can happen with any of the eight categories, but it almost always happens with boots and hats, and it frequently happens with the four thru_? categories. In these cases, only one of the cut points is known (the one adjacent to the last line slice erased). For example, consider the bottom-left corner on the thru_dr stroke shown in Figure 11. Cut point A is determined by the location of the last line slice erased to the left. Assume in this example the character stroke and the line data merge smoothly enough that inflection point B cannot be detected according to second derivative statistics. In instances like this, the contour pixels near point A are analyzed and used to project a line towards point B.

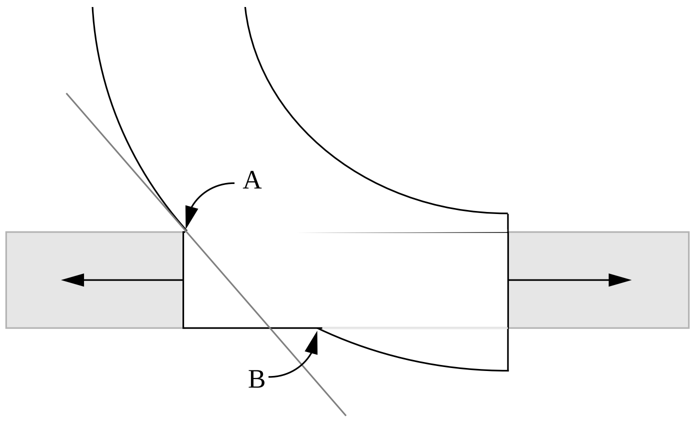


Figure 11. One-sided projection to cut off the bottom-left corner on a thru_dr character stroke.

This is done by starting at the top of the last erased line slice (one slice left of point A) and searching upwards for the first black pixel. This pixel is assumed to lie along the edge of the character stroke. Then, the same search is conducted from the top of the next slice to the left, and then the next slice to the left of that slice, and so on. If black pixels are found, then as many as 6 consecutive slices will be used in the search, producing consecutive contour points along the character stroke. The relative differences between each successive contour point's y-coordinate is calculated and the total change in y is tallied and used to compute a slope. A line is projected from point A (represented as the dashed line in the figure). Any black pixels below and to the left of the line are erased, as they are assumed to be part of the line. The projected cut line only approximates the location of the second cut point. The contour of the character would be more accurately represented by a projected curvature. Nonetheless, representing the projected character contour with a straight line, in practice, does a reasonably good job at estimating point B.

3.2.4 Identifying Character Stroke Categories

To this point, we have described eight general categories of overlapping line and character stroke configurations. Within each of these categories, we have identified a predictable number of areas represented as corners that contain black pixels comprising line data, and we have described two different techniques for detecting these corners and removing the black pixels. A step not yet discussed is how do overlapping line and character stroke configurations get categorized into one of the eight groups?

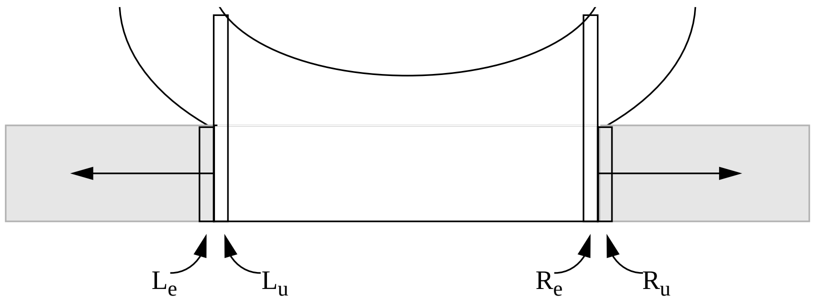


Figure 12. Slice comparisons for determining the location of character data.

Recall that each line and character stroke configuration is identified by the contiguous group of line slices not yet erased. Each configuration is categorized by measuring the slice statistics on the left and right edges of the group. The left and right pair of slices for a boot are illustrated in Figure 12. The top and bottom of the left-most unerased slice is compared with the top and bottom of its adjacent erased slice, and the top and bottom of the right-most unerased slice is compared with the top and bottom of its adjacent erased slice. Character data is determined to exist above or below the line by measuring the differences between the tops and bottoms of these pairs of slices.

In this figure, L_e represents the last slice erased on the left, and L_u represents the left-most slice unerased in the line-character configuration. The tops and bottoms of these two slices, when compared, determine there is character data above the line. The same is true for the right pair of slices. The slice R_e represents the last slice erased on the right, and R_u represents the right-most slice unerased in the configuration. The configuration of unerased slices can be identified using these simple features (whether character data is detected above or below the left and right edges). For example, if character data is determined to be above the line on the left (but not below), and character data is determined to be below the line on the right (but not above), then the group of unerased slices is identified to be a pos_cross. If character data is above the line on the left (but not below) and character data is above and below the line of the right, then the group of unerased slices is identified to be a thru_dr. In practice, the majority of line-character configurations will be identified as one of the eight categories illustrated in Figure 9. Those configurations that do not fall into any of the eight categories are simply left alone. Given the identity of the configuration, corners of line data can be located and removed according to the methods described above and illustrated in Figure 10 and Figure 11.

3.3 Line Removal Examples

This section presents a number of results from performing the above techniques on a variety of fields. The first group of results shown in Figure 13 are money fields extracted from IRS 1040 forms. These images were processed at 12 pixels per millimeter (300 pixels per inch) binary. Each contains two horizontal lines from the form (due to the cramped spacing of fields on these forms). The lower line marks the location of the current field, and the upper line marks the field above. In each example, the original field subimage is displayed above, with the results after intelligent line removal displayed immediately below. Notice, the handprint that intersects the lines is preserved, and the handprint protruding into the current field from above and below is also preserved. Keep in mind, the goal of this line removal technique is to automatically detect and erase as much of the horizontal line data as possible while preserving as much character stroke data as possible. These examples contain numerous instances of character strokes intersecting and overlapping line data, and as can be seen from the results, the line removal technique does a very good job at achieving its goals.

The next set of results, shown in Figure 14, are fields extracted from 1990 Decennial Census long forms. In these examples, the images were scanned at 8 pixels per millimeter (200 pixels per inch) and each field is demarcated by a dashed box. The dashed line detection method described in Section 3.1.2 was used to automatically locate the tops and bottoms of the box, and the results of removing the dashed horizontal lines are shown immediately below each original image. The dashed line detection does confuse machine print with dashed lines because the statistics of the black and white pixel runs are very regular and uniform. This confusion results in extra dashed lines being detected at the very tops and bottoms of the images (places where there is residual machine printed text), and the machine print is partially erased. Nonetheless, the horizontal dashed sides of the box are consistently located and removed with the handprint therein preserved, which is the goal of this work. Notice, although the third example in this group is rotated, the dashed lines are effectively detected and removed. This demonstrates the method’s relaxed dependence on form deskewing, which is another goal.

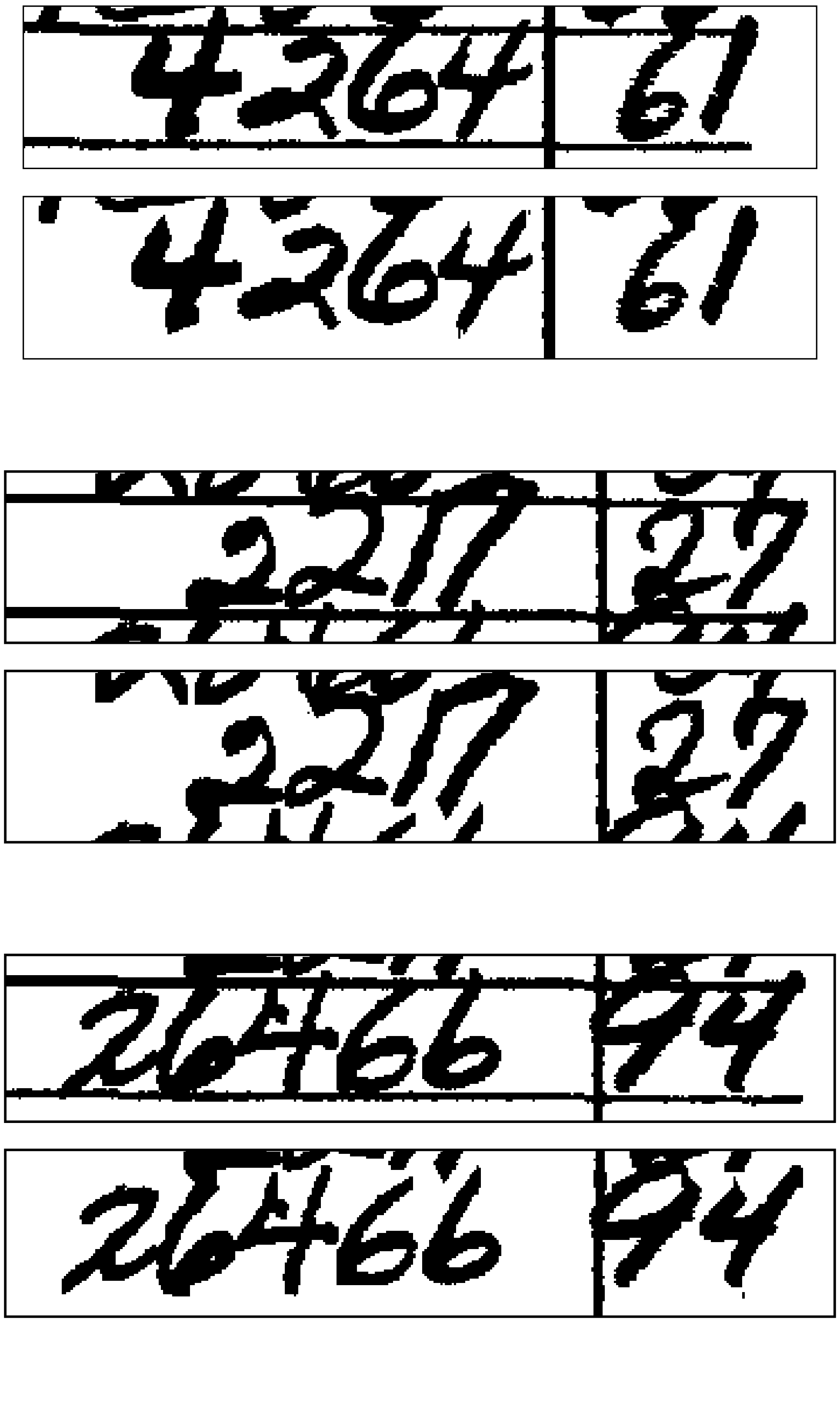


Figure 13. Horizontal lines removed from money fields off IRS 1040 tax forms.

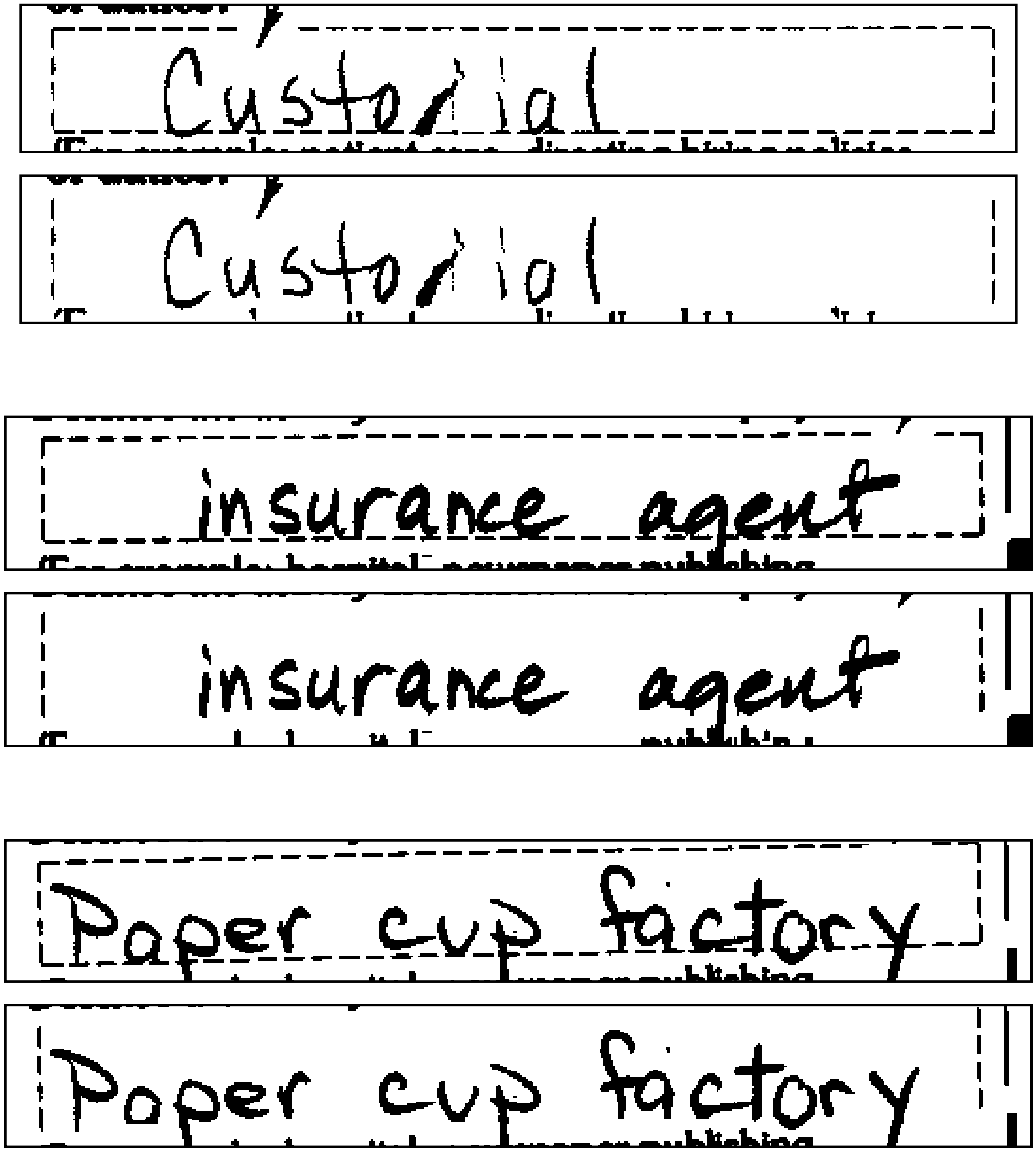


Figure 14. Dashed horizontal lines removed from occupation fields off Census forms.

The last set of examples in this section are shown in Figure 15. The first three subimages show results from the NIST method for line detection and removal. The images contain two response lines from the dependency block on an IRS 1040 form. This block is laid out as a small table in which multiple entry fields exist on multiple lines. The fields are extremely small compared to the amount of information requested to be entered. This causes many overlapping and touching characters within the same line as well as between lines, and many characters intersect and cross through the lines on the form. As a result, the likelihood of line and character stroke ambiguities (like those discussed in Section 2) is very high. This makes lines removal extremely difficult.

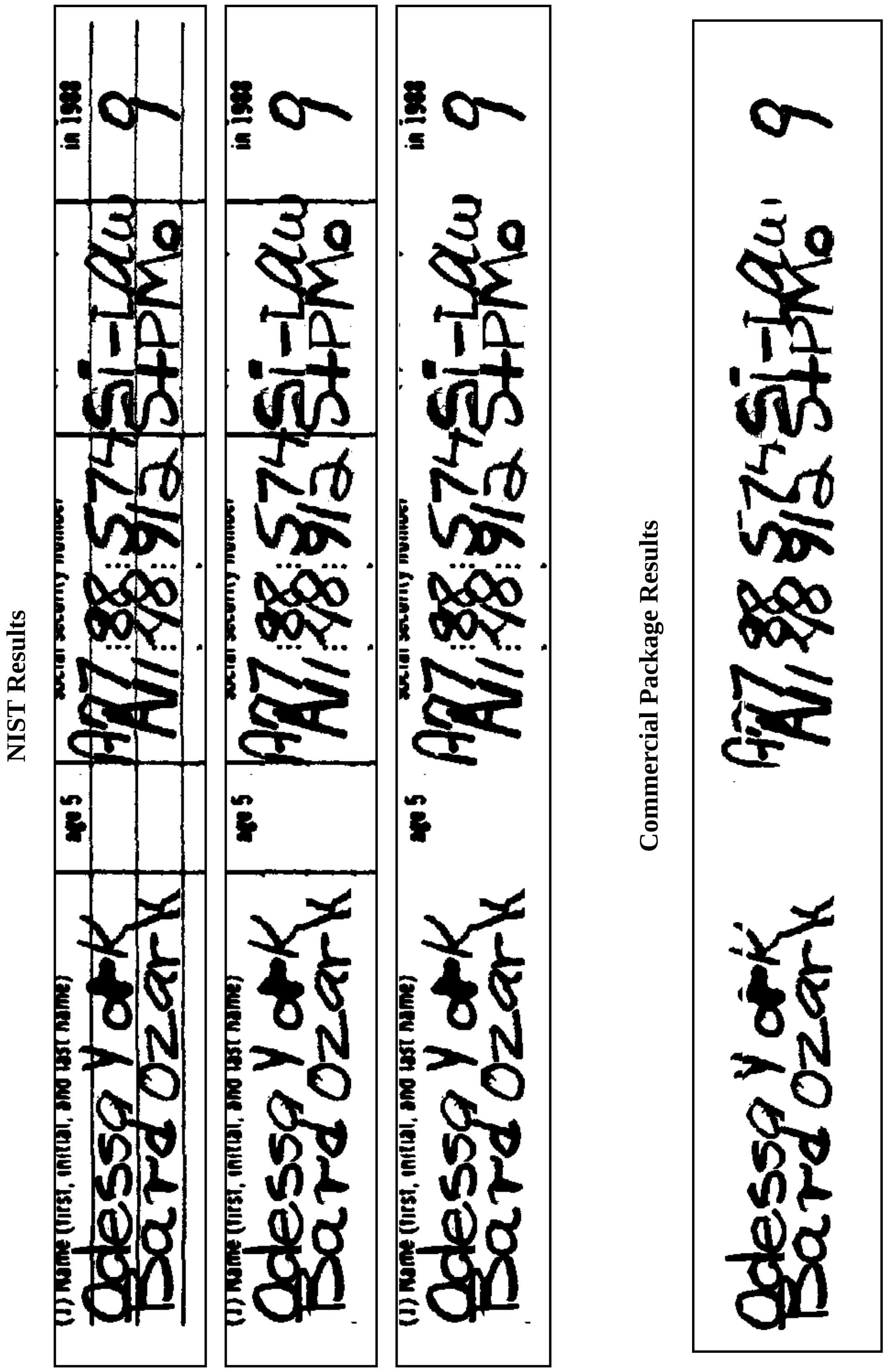

NIST Results
Commercial Package Results
(1) Name (first, initial, and last name)
age 5
social security number
in 1988

Figure 15. NIST results compared to a commercial form removal package.

The second image in this figure shows the results of horizontal line removal. The third image shows the results of conducting vertical line removal on the second image. To conduct vertical line removal, the second image was rotated 90° and the horizontal line removal technique was invoked with only two parameters changed. Line detection was restricted to 1° (previously 5°) of rotation, and the line length tolerance was set to 98% (previously 75%). The first parameter is reasonable, because the vertical lines in the field are relatively short, so minor rotational distortions will have a much shorter distance to propagate themselves. The second parameter limits the search for dominant lines to only those having at least 98% of their line comprised of black pixels. This is reasonable because the vertical lines in the image are solid and extend to the edges of the field subimage.

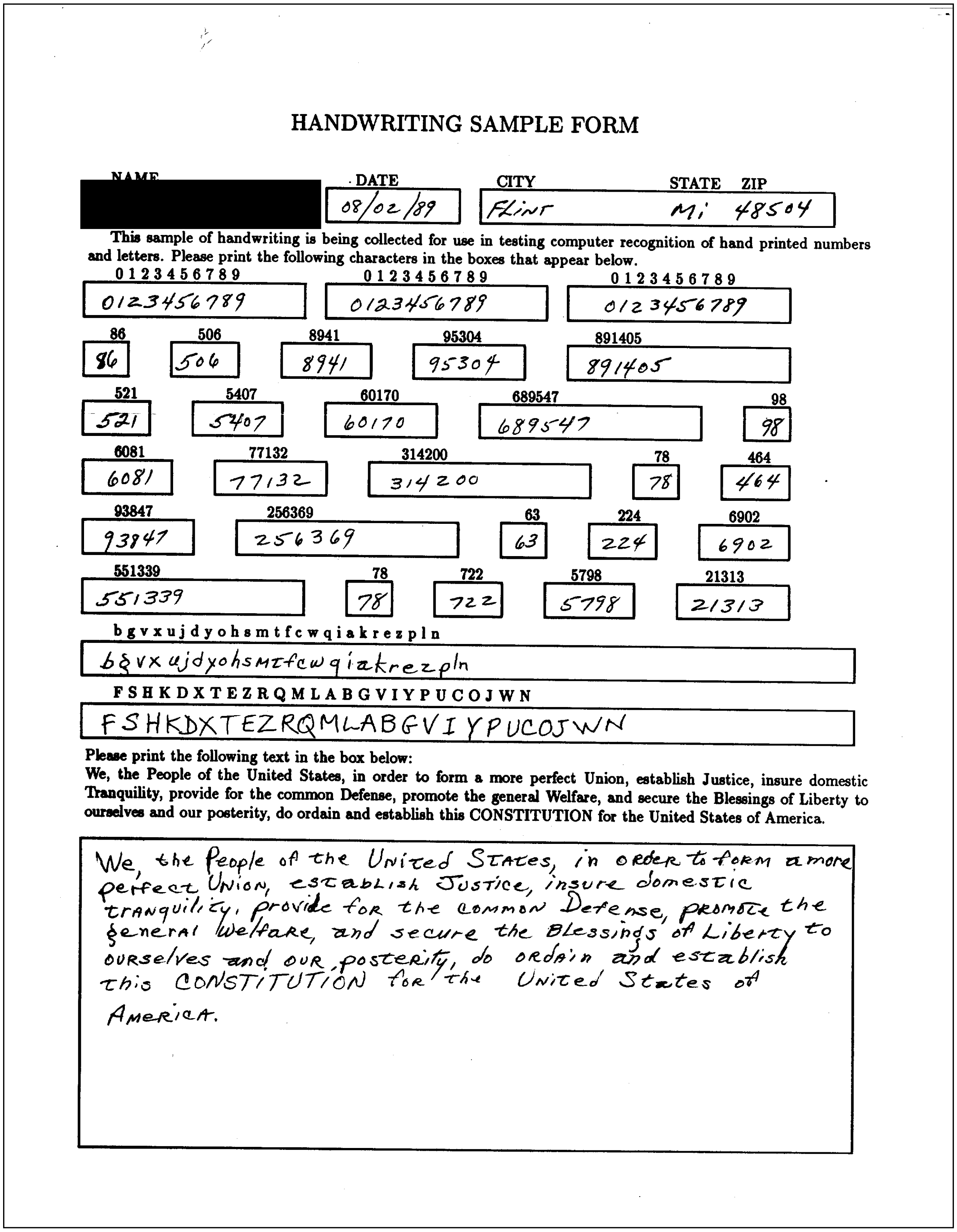
The bottom field in Figure 15 is taken from the results of a commercially available form removal package (the name of the package is not important). What is interesting is to compare the bottom image to the NIST results. Notice the frequent discontinuities in character strokes introduced by the commercial package. Keep in mind, this method of form removal had full knowledge that a 1040 form was being processed. Granted, this permitted the system to remove instructional information in addition to the line information on the form, but even though the commercial package had explicit form information available, its line removal performance is observably inferior. The NIST method performs in a superior way, even though it knew *nothing* about what form was being processed. The NIST method automatically detects all *dominant* lines in the image and intelligently erases them so character strokes are preserved. This method works amazingly well even on this very noisy and cluttered example, and the NIST method performed consistently equal to or better than the commercial system on all the fields inspected in this study.

4. IMPACT on AUTOMATED CHRARACTER RECOGNITION

Once the new method for line detection and removal was performing satisfactorily on a large number of test cases (like the ones shown in the previous section), a test was needed to evaluate the impact of the new method on the performance of an optical character recognition system. It was noted that the new method would have the greatest influence on lowercase characters with descenders, such as g, j, p, q, and y. These characters, when handprinted, frequently intersect and pass through the line along which they are written. If form removal techniques are not careful, they will clip (or disconnect) these descenders, causing inter-character ambiguities that decrease the performance of the system by inflating the number of substitutional errors. For example, a *q* with its descender clipped looks like an *a*.

NIST has gathered several large samples of handwriting and recently has published all of them in a single database called *NIST Special Database 19* (SD19)[14]. Each of the writers in this database were asked to fill in a handwriting sample form like the one shown in Figure 16. All of the 2,600 writers originally distributed with *NIST Special Database 1, 3*, and *7* have been included in SD19, with an additional 1,099 new writers. The characters handwritten on the forms have been segmented and labeled as independent images, so in all there are 3,699 forms and 814,255 labelled characters. A handwriting sample form is comprised of fields containing digits, a randomly ordered lowercase alphabet, a randomly ordered uppercase alphabet, and the Preamble to the U.S. Constitution. Having this database, the randomly ordered lowercase alphabets provide excellent testing material for evaluating the impact of intelligent line removal on optical character recognition.

In addition to the databases, NIST has a public domain software OCR package that provides the ability to integrate and test new image processing and recognition technologies without having to start from scratch in developing an entire system from the ground up. A complete description of the system is provided in the distribution documentation.[13] The OCR system uses a method of histogram projections to locate registration points within a handwriting sample form. These points are aligned to a set of reference registration points using a Linear Least Squares fit, and the image is transformed, removing any global distortions in rotation, translation, and scale. The system uses a blank form as a mask, erasing any pixels that are a part of the form (its boxes and instructions). The handwriting within each field is segmented using connected component labelling, and each component is size and slant normalized. The Karhunen Loève (KL) transform[20] is computed on each segmented character image, and a vector of KL coefficients is produced. These coefficients form a feature vector that is classified by an optimized Probabilistic Neural Network (PNN).[21,22] The recognition system stores the results of each classified character image, including the character's assigned class along with a confidence value.



HANDWRITING SAMPLE FORM

NAME	DATE	CITY	STATE	ZIP
[redacted]	08/02/89	FLint	Mi	48504

This sample of handwriting is being collected for use in testing computer recognition of hand printed numbers and letters. Please print the following characters in the boxes that appear below.

0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9
0123456789	0123456789	0123456789

86	506	8941	95304	891405
86	506	8941	95304	891405
521	5407	60170	689547	98
521	5407	60170	689547	98
6081	77132	314200	78	464
6081	77132	314200	78	464
93847	256369	63	224	6902
93847	256369	63	224	6902
551339	78	722	5798	21313
551339	78	722	5798	21313

b g v x u j d y o h s m t f c w q i a k r e z p l n

bgvxujdyohsmtfcwqiakrezpln

F S H K D X T E Z R Q M L A B G V I Y P U C O J W N

FSHKDXTEZRQMLABGVIYPUCOJWN

Please print the following text in the box below:
We, the People of the United States, in order to form a more perfect Union, establish Justice, insure domestic Tranquility, provide for the common Defense, promote the general Welfare, and secure the Blessings of Liberty to ourselves and our posterity, do ordain and establish this CONSTITUTION for the United States of America.

We, the People of the United States, in order to form a more perfect Union, establish Justice, insure domestic tranquility, provide for the common Defense, promote the general Welfare, and secure the Blessings of Liberty to ourselves and our posterity, do ordain and establish this CONSTITUTION for the United States of America.

Figure 16. A handwriting sample form from *NIST Special Database 19.*

As stated before, the original system uses a blank form as a mask to erase pixels from the input image. Any part of a character overlapping with the form is erased, causing voids and discontinuities in character strokes (especially lowercase letters with descenders). To avoid this, the public domain OCR system was modified, replacing the mask-based erasing with the new line detection and removal techniques. The lowercase field is extracted from the form (including the field's box and some residual instructional information), and the new techniques are applied. In the process of integrating the new capabilities, several issues had to be addressed. These included incorporating image tiling for the accurate detection of very long lines, developing a general method for isolating the handprint in a field once the horizontal lines are removed, and creating a new training set of labeled character images.

4.1 Tiling of Image for the Detection of Long Lines

A lowercase alphabet box on a handwriting sample form is 175 mm (not quite 7 inches) wide, that is 2,100 pixels in a 12 pixel per millimeter (300 pixels per inch) image. The length of the horizontal sides of the box used in this field are so long there is often enough distortion in the image that no perfectly straight line can overlay the entire edge of the box. This causes deficiencies in the Hough line transform, making it very difficult to detect long lines and to track line slices.

The subtle distortions in the image become significant as they propagate over longer and longer distances. In order to compensate, the lowercase field images are divided into adjacent and non-overlapping pieces (tiles), where the width of each tile is no more than 500 pixels wide. Within each tile, intelligent line detection and removal is conducted, and the handprint in the tile is isolated and extracted. In this way, the effects of image distortions are avoided as they are negligible over the shorter tile widths.

4.2 Isolation and Extraction of Handprint

Typically, there are at least two dominant horizontal lines detected in a lowercase field using the Hough line transform (the top and bottom horizontal sides of the box). So far, the techniques for detecting and removing the two lines have been discussed, but there is more work to be done in order to isolate the handprint in the image. In addition to the handprint, there is still residual instructional information in the field, and the two vertical edges of the box remain.

As lines are removed, their end points are stored in a list. The left end points are sorted by their y-coordinates, and vertical distances are computed between each neighboring pair of lines. The line pair representing the largest distance is assumed to belong to the tops and bottoms of the horizontal sides of the field's box. A quadrilateral region is defined as the intervening area between these two lines (the region is not necessarily square). A search is conducted on the pixels in this region, top-to-bottom and left-to-right. As a black pixel is found, its connected component is extracted and stored as a plausible character image, and the component is erased from the field image. The search continues for black pixels, and connected components are copied and erased, until the entire region is processed. In this application, the first and last significant components in the field are discarded as they corresponded to the vertical sides of the box. The region bounded by the two horizontal lines contains the majority (but not all) of the handprint in the field. By using connected components to lift the handprint out of the image, characters that reside in the region, but extend outside the region, are extracted in their entirety. By limiting the search to this region, residual instructional information and other spurious marks are avoided, so only handprint is extracted.

4.3 Training Set Creation

A problem arose because the public domain system's classifier was not trained on character images generated by the new line removal technique. The original system's classifier was trained on lowercase examples whose descenders and other character strokes overlapping the line were clipped. Up till now, NIST did not have the technology to conduct robust form removal, so we were faced with a type of *chicken and the egg* dilemma. The clipped characters introduce some obvious biases into the performance of the recognition. These biases include creating artificial inter-character ambiguities and causing a certain number of letters to be under-represented by good, cleanly segmented character images. In order to evaluate the *true* impact of intelligent line removal on OCR, it was necessary to retrain the recognition system on character images produced by the new line removal technique.

The 500 writers in the group labeled hsf_6 were selected from SD19 for creating a new training set of cleanly segmented character images. Each writer's lowercase alphabet was isolated, segmented, and classified, producing a hypothesized field value from the recognition system's original (biased) lowercase weights. The recognition system used the line detection and removal techniques described above, so descenders and other lowercase character strokes intersecting the lines on the form were preserved. Each field value produced by the system was compared against a *reference* string of what was actually requested to be written in the field.

Dynamic string alignment based on the Levenstein distance was used to conduct the comparison.[23] This alignment method labels each character in the system's *hypothesis* string as either correct, substituted, or inserted; and each character in the reference string as either correct, substituted, or deleted. The object of this exercise was to produce a new set of training examples with verified reference labels. To minimize the labor involved in labeling each character correctly, the system's assigned hypothesis labels were used, except where the character was identified as a substitution by the alignment utility, in which case the substitutions were replaced by their corresponding reference label. Using the reference label for characters aligned as substitutions helped automatically repair some of the character assignments prior to human verification, thus reducing the amount of manual labor spent repairing incorrectly classified characters.

To begin human verification, all the characters recognized correctly, along with all the reassigned substitutions, were grouped into one large set of characters and sorted by character class (a through z). The remaining character images (aligned as insertions) were grouped into a second set of characters and sorted by class. The two sets of characters were used to bootstrap a character verification scheme called *machine-assisted reference classification*[24].

The first group of characters were displayed by class in large blocks shown simultaneously on a computer display. In this way, those characters misclassified could be easily detected and marked because they stand our against the background of all the other characters belonging to the correct class. The assigned labels of those characters not marked were accepted and set aside as verified while the marked characters where added to the second group of characters (those aligned as insertions). This second group of characters was displayed by class, but this time, one character at a time. When viewed, the human operator was given the choice to accept, reject, or manually correct the assigned class. All the characters accepted or corrected were again collected into a group and displayed simultaneously in large blocks by class. The marked characters from this session were collected into a second group and displayed one at a time. This process of checking and correcting continued several cycles until all the characters either had their assigned labels verified and accepted or the character image was rejected (due to bad segmentation).

In all 12,500 characters were verified and accepted. The entire verification process took one person less than 8 hours to complete. This machine assisted labelling technique efficiently provided a new training set of cleanly segmented characters, and the NIST public domain OCR package (augmented with the new line detection and removal techniques) was retrained. The new training set was used to compute a new covariance matrix and new eigenvector functions, which in turn, were used to compute new KL feature vectors. For this experiment, the 64 top-ranked eigenvectors were used to compute 64-element feature vectors. The new prototype feature vectors were then used on-line to compute discriminant functions in the optimized PNN classifier.

4.4 Global Scoring Results

A test set consisting of the 2,100 forms in hsf_0, 1, 2, and 3 (also included in SD19) was chosen. The augmented and retrained recognition system processed the lowercase field on each form and the results were stored to disk in a format compatible with the NIST recognition system scoring package.[12,25] Every lowercase field was assigned a reference string, and the scoring package aligned the reference strings to the system's hypothesized classifications. Based on these alignments, global performance statistics were automatically calculated and reported.

The same exercise was repeated for an older version of the NIST recognition system. In this case, the automatic line detection and intelligent line removal was not used. This old version used the estimates of rotational distortion from the form registration process to conduct skewed histogram projections to locate the edges of the boxes on a handwriting sample form. A subimage of the lowercase field was extracted, including the box and residual instructional information. Rather than conduct histogram projections parallel to the raster scan lines in the image, adjacent lines (the width of one pixel) were projected along a trajectory defined by the estimated amount of rotation in the image. The two maximum peaks in the skewed histogram (corresponding to the two horizontal sides of the box) were selected, and any pixel data that overlapped or extended beyond the sides of the box was erased. The old version of the recognition system clipped any lowercase character that intersected or extended outside the box. All the other components (connected component character segmentation, size and slant normalization, KL feature extraction, and optimized PNN classification) were the same as those used in the new system. It should be noted that the old system was trained on prototype feature vectors computed from segmented character images generated by the skewed histogram technique.

To assess the overall impact of the new form removal technique, the OCR results from the old and new systems were computed and then scored using the NIST scoring package. Global performance statistics were automatically calculated and reported. The old system achieved a character output accuracy of 76.9% out of a possible 54,340 lowercase letters; the test actually consisted of 2,090 lowercase fields. The new system achieved a character output accuracy of 80.1% on the same characters. Using the new line detection and removal techniques on the lowercase alphabet fields improved the recognition about 3%. This is reasonable, as there are only 5 out of the possible 26 letters in the lowercase alphabet that have significant descenders, and from observation, we know these 5 letters are not always written such that they overlap the line on the form. If on average one of the 5 letters touches the line during the printing of the alphabet, then this would account for 3.8% (1/26) of the letters. We also know this would be an upper bound (given our assumptions), because not all of the touching characters would cause an incorrect classification. So, a 3% improvement in recognition is *reasonable*.

4.5 Statistical Analysis of Confusion Matrices

The 3% increase in recognition accuracy as a result of using intelligent line removal is in fact an improvement, but this global performance statistic tells us very little about *how* the recognition was really impacted. Global performance measures are helpful, but in practice their usefulness is limited. If nothing else, some measure of statistical confidence should be associated with the global measures. UNLV[26] has incorporated confidence limits into their scoring models and NIST is working to incorporate them into the NIST scoring package as well. But, even this information is limited. There needs to be a way of computing *local* performance statistics on both the gains and the losses in recognition accuracy. For example in this study, we knew the improvements to the recognition system should have their greatest impact on lowercase letters (particularly, those with descenders), and we just happened to have a database of lowercase characters on which to test. We have shown a 3% improvement in overall recognition accuracy, but we do not know if changes to the recognition system inadvertently caused some recognition problems. If so, what are these degradations? Is there something that can be done to minimize them so accuracy continues to improve towards some upper bound?

Any experienced system developer realizes any so-called *improvement*, when integrated into a complex recognition system, is likely to introduce new sources of error. It is all too familiar: a component is re-engineered and tediously tested in isolation on a relatively small (yet manageable) test set. The new component seems to be functioning as designed and is now placed into the end-to-end system. All activity in the room stops, the system developer holds his breath, and yes … it works! The performance of the system improves, and yet in the back of his head he wonders, “But what have I traded off (if anything) for the improvement in performance?” Lacking a means to answer this question, he is quick to accept the global performance increase, and he enjoys his brief moment of serendipity. Sound familiar? It does to the author.

A technique for determining the significant difference in local performance statistics between two OCR systems has been developed. The method analyzes the confusion matrices generated by the two recognition systems. A confusion matrix stores the number of times individual class substitutions occur during a test. Each cell in the matrix contains the number of times a letter (for example g) was incorrectly classified as another letter (for example q). The matrix represents all possible confusion pairs, with one axis representing all the characters as they should be recognized and the other axis representing all the characters as they were actually recognized. Traditionally, the diagonal of the confusion matrix contains all the characters correctly recognized, but we are only interested in the errors for this analysis, so the diagonal elements are set to zero.

To analyze confusion pairs between two different systems, the test set is partitioned into *n* equal subsets. The substitution errors incurred by each system on each test subset are collected into separate confusion matrices. This produces *n* confusion matrices for the first system, and a set of *n* confusion matrices for the second system. The confusion matrices from each system provide *n* samples for each possible confusion pair. The *n* samples are used to compute distribution statistics (a mean and a standard deviation). For each confusion pair's mean μ, the standard deviation σ is computed as

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\mu)^2} \tag{8}$$

where *n* is the number of partitions (in this case 10) and *x* represents the 10 samples for the current confusion pair. A mean and standard deviation pair is computed for each cell in the confusion matrix for the first system (μ_1, σ_1) and for each cell in the confusion matrix for the second system (μ_2, σ_2). As a result, a matrix of mean values and a matrix of standard deviations are computed for each of the two systems.

Given the distribution statistics of each corresponding confusion pair between the two systems, a Student's *t* test can be used to determine how similar the distributions are to each other.[27] The difference between two distributions is measured as

$$t = \frac{\mu_1 - \mu_2}{\sqrt{\frac{\sigma_1^2}{n} + \frac{\sigma_2^2}{n}}} \tag{9}$$

which is in units of root mean square standard error. Given the normalized distance *t*, a probability ρ is derived either numerically[28] or via table look-up[27] for each cell in the confusion matrix. This is the probability that $|t|$ could be at least this large by chance, so the smaller the value of ρ, the less likely the two distributions are the same. Low values of ρ signify the difference between the two systems for a particular confusion pair is significant. The table in Figure 17 was produced by thresholding ρ at 2%, in which case we are 98% sure the difference did *not* happen by random chance. The confusion pairs with ρ less than 2% were sorted on their corresponding value of *t* and reported in that order in table. Notice there are only 41 confusion pairs listed in the table. If wanted, the confidence threshold can be lowered in order to examine more confusion pairs. The full confusion matrix contains 676 (26×26) pairs, and (disregarding the correct recognitions along the diagonal) only 6.3% (41/650) of the possible confusion pairs are determined to be significant in this analysis. Without a statistical method like this, it would be very difficult to manually select this small subset of confusion pairs.

The first two columns in the table report each confusion pair determined to be statistically different. The first character, labeled (R)ight, is the reference character the system should have recognized. The second character, labeled (W)rong, is the hypothesis character the system incorrectly assigned to the letter. For example, the first line in the table is reporting statistics on the system incorrectly classifying p's as o's. The next two columns, labeled SYSTEM 1, contain the performance statistics from the old recognition system. The first of these columns lists the average number of errors μ_1 incurred for the corresponding confusion pair across the 10 test partitions. The second column lists the standard deviations σ_1 associated with these errors. The second system's error statistics are listed in the next two columns labeled SYSTEM 2. These are the errors from using the new line detection and removal techniques.

PAIR		SYSTEM 1		SYSTEM 2		MEAN Δ	STUDENT'S *t*	
R	W	μ_1	σ_1	μ_2	σ_2	$\mu_1 - \mu_2$	t	$\rho \times 10^2$
p	o	12.7	1.4	1.8	0.5	10.9	23.7	0.0
g	a	25.6	4.1	6.2	1.6	19.4	13.9	0.0
q	a	33.8	4.3	12.6	3.4	21.2	12.3	0.0
j	l	27.6	3.9	11.3	2.9	16.3	10.6	0.0
y	v	27.6	5.1	7.6	3.7	20.0	10.0	0.0
q	o	8.3	2.0	2.5	1.0	5.8	8.2	0.0
y	u	10.3	2.5	3.2	1.2	7.1	8.2	0.0
b	h	9.7	2.4	3.3	1.4	6.4	7.4	0.0
j	i	17.0	2.6	9.9	2.9	7.1	5.7	0.0
p	n	4.8	2.2	1.1	0.5	3.7	5.2	0.0
p	b	3.1	1.4	0.8	0.3	2.3	5.0	0.0
q	u	1.1	0.5	0.4	0.0	0.7	4.7	0.1
g	n	2.3	1.4	0.5	0.0	1.8	3.9	0.3
w	b	0.6	0.3	0.2	0.0	0.4	3.8	0.4
g	o	6.6	2.2	3.6	1.3	3.0	3.7	0.2
j	k	0.7	0.5	0.2	0.0	0.5	3.4	0.8
y	m	0.6	0.5	0.1	0.0	0.5	3.4	0.8
o	n	3.8	1.2	2.0	1.2	1.8	3.2	0.5
p	d	1.9	1.4	0.6	0.0	1.3	2.9	1.7
e	p	4.0	1.5	2.5	0.7	1.5	2.9	1.3
p	f	2.6	0.9	1.8	0.0	0.8	2.9	1.8
j	e	0.4	0.3	0.1	0.0	0.3	2.9	1.9
p	h	0.3	0.3	0.0	0.0	0.3	2.9	1.9
j	c	0.5	0.3	0.2	0.0	0.3	2.9	1.9
y	f	0.4	0.3	0.1	0.0	0.3	2.9	1.9
c	j	0.4	0.3	0.1	0.0	0.3	2.9	1.9
q	d	1.2	0.6	0.6	0.3	0.6	2.9	1.3
q	w	0.6	0.3	0.3	0.0	0.3	2.9	1.9
j	n	0.7	0.0	0.4	0.3	0.3	2.9	1.9
f	s	1.4	0.5	0.9	0.3	0.5	2.7	1.4
g	u	0.4	0.3	0.9	0.5	-0.5	-2.7	1.4
g	m	0.2	0.0	0.5	0.3	-0.3	-2.9	1.9
q	x	0.3	0.0	0.6	0.3	-0.3	-2.9	1.9
w	x	0.4	0.0	0.7	0.3	-0.3	-2.9	1.9
q	g	18.9	5.2	25.1	4.3	-6.2	-2.9	1.0
y	q	1.7	0.9	2.9	0.9	-1.2	-3.0	0.7
y	g	2.7	1.4	5.4	2.2	-2.7	-3.3	0.5
d	t	0.2	0.0	0.6	0.3	-0.4	-3.8	0.4
b	o	0.9	0.5	1.6	0.3	-0.7	-3.8	0.1
j	v	2.4	0.8	3.9	0.9	-1.5	-4.0	0.1
r	g	0.2	0.0	0.7	0.3	-0.5	-4.7	0.1

Figure 17. Statistical analysis reporting all significant (98% confident) changes in confusion errors between the old system that chopped off characters and the new system that removed lines while preserving character stokes.

The column in the table, labeled MEAN Δ, is the difference in the mean accumulated errors between the two systems. With p's classified as o's, there were on average 10.9 fewer errors made by the second recognition system. Keep in mind there were 2,090 lowercase alphabets in the test, and these fields were divided into 10 equal partitions. As a result, there were 209 examples of each character in each partition, so 10.9 fewer p's called o's is an average decrease of 5.2% (10.9 / 209). The largest significant improvement was in q's called a's, where there was a 10.1% decrease in this type of error.

The last two columns in the table list the results of the Student's *t* test. The first column lists the normalized distance *t* between the first and second systems' distribution of errors for the corresponding confusion pair. The second column lists the corresponding value of ρ, the probability that the measured difference between the two distributions occurred by chance.

The table is divided vertically in two parts. The top portion lists all those confusion pairs in which the new system improved, making fewer errors. This is represented in positive mean Δ's and in positive values of *t*. The bottom portion of the table lists all those confusion pairs in which the new system did worse than the old system. In this case, the mean Δ's and *t* values are negative. These represent the significant trade-offs of introducing the new form removal method into the recognition system. Despite the losses in performance, there are considerably more improvements with the new line detection and removal techniques, and their net impact on performance greatly outweighs the losses (3%).

A closer look at the table shows the statistics are well behaved. Looking at the top and bottom of the column of ρ values, one can observe that at the ends, ρ is very low so the confidence is very high that the difference between the two systems is significant. As we move to the middle of the table, the absolute values of *t* decrease and the values of ρ increase. This should occur because, as a general rule, the closer two distributions are to each other the more likely they come from the same underlying distribution. It is interesting to also note that, in general, the mean Δ's follow this same trend. Although, there are some exceptions giving support to the fact that, without some measure of statistical significance, a single measured sample may be misleading.

In light of these observations, the confusion pairs at each end of the table are statistically most significant. Using the intelligent line detection and removal techniques, the most significant improvements occur with confusion pairs (p, o), (g, a), (q, a), (j, l), (y, v), (q, o), and (y, u). The first (reference) character in these pairs all have descenders, and if you remove the descenders you are left with a partial character that closely resembles the second character in each pair. As we expected, the intelligent removal of lines does significantly improve recognition, especially lowercase characters with descenders. The statistical analysis of the confusion matrices was used to assess the impact of intelligent form removal on off-line handwriting recognition, and by the predictable nature of the experiment, the OCR test results have served to validate the statistical method.

On the other end of the table, significant decreases in performance have occurred with (r, g), (j, v), and up the list a bit further (q, g). These are cases where introducing the new method of line removal actually increased confusion among specific pairs of characters. The new line removal technique attempts to preserve as much of the handwritten character as possible. In so doing, some of the lowercase characters now have considerably longer descenders, which in the old system were clipped. The consistent chopping off of descenders, while bad for classification in general, actually avoids certain inter-character ambiguities. By preserving the descenders, these naturally occurring ambiguities are reintroduced into the recognition problem and errors among these confusable characters increase. For example, j's that were once cut off and highly confusable as l's now have their descenders preserved, so at times their tails curve up sufficiently to be confusable with handprinted v's.

Some of the other confusion pairs, having a negative impact on the new recognition system, are harder to explain. It is our experience that the size normalization used in the preprocessing of the character image can cause unexpected, yet consistent, patterns. For example, as the descenders on handprinted g's become longer and longer, the dominance of the top loop on the letter, after size normalization, becomes smaller and smaller to the point that it closes. At this point, the normalized character image does become confusable with some handprinted r's. The point is there was no way to predict what the impact would be when reintroducing these naturally occurring ambiguities into the system. Through this statistical analysis, the significant system degradations (in addition to the improvements) have been automatically identified.

5. CONCLUSIONS

A new method for form removal has been presented and analyzed. The Hough line transform was embellished to automatically detect all the dominant lines in an image, and a new method (based on visual cues along the line trajectory) was presented for intelligently removing the lines from the image. Lines are removed while simultaneously preserving any overlapping character strokes. Results from this new form removal were shown to be superior to a commercially available product, and these techniques were shown to improve the recognition of lowercase alphabets by 3%.

A statistical method was also presented that analyzes the local performance statistics between two recognition systems. A large testing set is partitioned into subsets and confusion matrices are compiled from the results on each partition for each recognition system. Distribution statistics (mean and standard deviation) are computed across the partition results for each confusion pair in the matrices. A Student's *t* test is then used to determine if the corresponding confusion pair statistics between the two systems is statistically different. This analysis is very useful because both the improvements and the losses in recognition performance are determined, whereas global performance statistics only report the *net* change in performance. As system developers continue to pursue lower and lower error rates, more sophisticated analyses (like the one presented in this paper) become necessary to understand the impact a modification really has on the recognition performance of a complex OCR system. For example, this method of evaluation should be very useful in squeezing higher performances out of voting systems, in which the decisions from multiple classifiers are combined to improve recognition performance.

The statistical analysis of the confusion matrices was used to assess the impact of intelligent form removal on OCR, and by the predictable nature of the experiment, the OCR test results have served to validate the statistical method.

6. REFERENCES

1. C. L. Wilson, R. A. Wilkinson, and M. D. Garris, “Self-Organizing Neural Network Character Recognition on a Massively Parallel Computer,” *International Joint Conference on Neural Networks*, Vol. II, PP. 325-329, San Diego, 1988
2. M. D. Garris, C. L. Wilson, J. L. Blue, G. T. Candela, P. Grother, S. Janet, and R. A. Wilkinson, “Massively Parallel Implementation of Character Recognition Systems,” In *Conference on Character Recognition and Digitizer Technologies,* Vol. 1661, pp. 269-280, SPIE, San Jose, February 1992.
3. M. D. Garris and D. L. Dimmick, “Evaluating Form Designs for Optical Character Recognition,” Technical Report NISTIR 5364, National Institute of Standards and Technology, February 1994.
4. R. A. Wilkinson, J. Geist, S. Janet, P. J. Grother, C. J. C. Burges, R. Creecy, B. Hammond, J. J. Hull, N. W. Larsen, T. P. Vogl, C. L. Wilson, “The First Census Optical Character Recognition Systems Conference,” Technical Report NISTIR 5452, National Institute of Standards and Technology, May 1994.
5. J. Geist, R. A. Wilkinson, S. Janet, P. J. Grother, B. Hammond, N. W. Larsen, R. M. Klear, M. J. Matsko, C. J. C. Burges, R. Creecy, J. J. Hull, T. P. Vogl, C. L. Wilson, “The Second Census Optical Character Recognition Systems Conference,” Technical Report NISTIR 5452, National Institute of Standards and Technology, May 1994.
6. C. L. Wilson, “Evaluation of Character Recognition Systems,” In *Neural Networks for Signal Processing III,* pp. 485-496, IEEE, New York, September 1993.
7. P. J. Grother and G. T. Candela, “Comparison of Handprinted Digit Classifiers,” Technical Report NISTIR 5209, National Institute of Standards and Technology, June 1993.
8. G. Nagy, “At the Frontiers of OCR,” *Proceedings of the IEEE*, Vol. 80, No. 7, pp. 1093-1100, July 1992.
9. R. M. Haralick, “Document Image Understanding: Geometric and Logical Layout,” Proceeding of the *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 385-390, Seattle, June 1994.
10. R. Kasturi, L. O'Gormann, “Document Image Analysis: A Bibliography,” *Machine Vision and Applications*, Vol. 5, pp. 231-243, Springer-Verlag, New York, 1992.

11. M. D. Garris, “Methods for Evaluating the Performance of Systems Intended to Recognize Characters from Image Data Scanned from Forms,” Technical Report NISTIR 5129, National Institute of Standards and Technology, February 1993.
12. M. D. Garris and S. A. Janet, “Scoring Package Release 1.0, *NIST Special Software 1*,” Technical Report NISTIR 4950 and CD-ROM, National Institute of Standards and Technology, October 1992.
13. M. D. Garris, J. L. Blue, G. T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C. L. Wilson, “NIST Form-Based Handprint Recognition System,” Technical Report NISTIR 5469 and CD-ROM, National Institute of Standards and Technology, July 1994.
14. P. J. Grother, “Handprinted Forms and Character Database, *NIST Special Database 19*,” Technical Report and CD-ROM, National Institute of Standards and Technology, March 1995.
15. J. Geist, “A Comparison of Two ICR Engines, (1995, available from Cardiff Software Inc., Carsbad, CA).
16. M. D. Garris, “Unconstrained Handprint Recognition Using a Limited Lexicon,” In Proceedings on *Document Recognition*, Vol. 2181, pp. 36-46, SPIE, February 1994.
17. M. D. Garris and D. L. Dimmick, “Form Design for High Accuracy Optical Character Recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, to be published.
18. P. V. C. Hough, “Methods and Means for Recognizing Complex Patterns,” U.S. Patent 3,069,654, 1962.
19. R. C. Gonzalez and P. Wintz, *Digital Image Processing*, Second Edition, pp. 130-134, Addison Wesley, Reading, Massachusetts, 1987.
20. P. J. Grother, “Karhunen Loève Feature Extraction for Neural Handwritten Character Recognition,” In *Proceedings: Applications of Artificial Neural Networks III*, Vol. 1709, pp. 155-166, SPIE, Orlando, April 1992.
21. D. F. Specht, “Probabilistic Neural Networks,” *Neural Networks*, Vol. 3(1), pp. 109-119, 1990.
22. J. L. Bentley, “Multidimensional Binary Search Trees Used for Associative Searching,” *Communications of the ACM*, Vol. 18, pp. 509-517, 1975.
23. H. G. Zwakenberg, “Inexact Alphanumeric Comparison,” *The C Users Journal*, pages 127-131. May 1991.
24. M. D. Garris, “Design, Collection, and Analysis of Handwriting Sample Image Databases,” *Encyclopedia of Computer Science and Technology,* Vol. 31, pp. 189-213, Marcel Dekker, New York, 1994.
25. M. D. Garris, “NIST Scoring Package Cross-Reference for use with NIST Internal Reports 4950 and 5129.” Technical Report NISTIR 5249, National Institute of Standards and Technology, August 1993.
26. T. A. Nartker, A. D. Bagdanov and J. Kanai editors, “1995 Annual Research Report,” UNLV Information Science Research Institute, 1995.
27. G. W. Snedecor and W. G. Cochran, *Statistical Methods*, 8th Edition, pp. 53-57, Iowa State University Press, Ames Iowa, 1989.
28. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes, The Art of Scientific Computing (FORTRAN Version),* pp. 466-467, Cambridge University Press, Cambridge, 1989.