Intelligent Pen: A Least Cost Search Approach to Stroke Extraction in Historical Documents

Kevin L Bauer and William A Barrett, Brigham Young University

1 Abstract

Extracting strokes from handwriting in historical documents provides high-level features for the challenging problem of handwriting recognition. Such handwriting often contains noise, faint or incomplete strokes, strokes with gaps, overlapping ascenders and descenders and competing lines when embedded in a table or form, making it unsuitable for local line following algorithms or associated binarization schemes. We introduce Intelligent Pen for piece-wise optimal stroke extraction. Extracted strokes are stitched together to provide a complete trace of the handwriting. Intelligent Pen formulates stroke extraction as a set of piece-wise optimal paths, extracted and assembled in cost order. As such, Intelligent Pen is robust to noise, gaps, faint handwriting and even competing lines and strokes. Intelligent Pen traces compare closely with the shape as well as the order in which the handwriting was written. A quantitative comparison with an ICDAR handwritten stroke data set shows Intelligent Pen traces to be within 2.58 pixels (mean difference) of the manually created strokes.

2 Introduction

One key problem in processing historical images is properly segmenting the image into words and extracting the relevant stroke information from the handwriting. By their nature, historical documents pose many challenges for stroke extraction, including faint strokes, strokes with gaps, form lines, and ascenders and descenders (Figure 1).

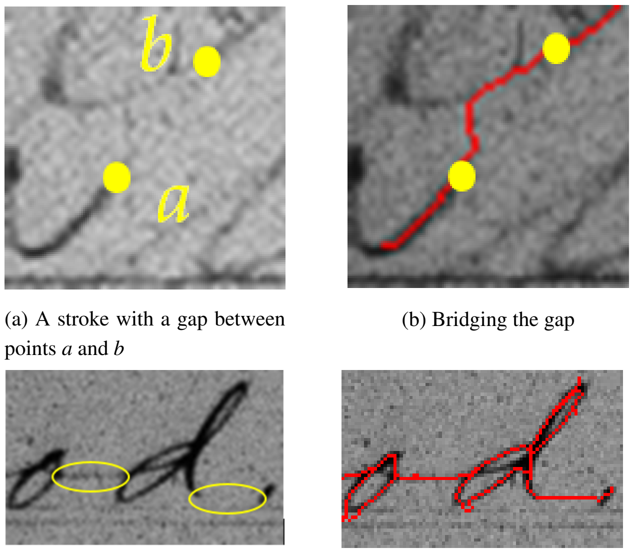
While many algorithms for stroke extraction exist, their effectiveness is limited to newer documents that are cleanly scanned with little degradation or that are born digital because of the challenges mentioned above.

3 Related Work

Typically the goal of stroke extraction algorithms is to be able to apply on-line handwriting recognition methods to static, offline handwriting images. This is usually done by using a medial axis transform or some other thinning algorithm to extract a skeleton: a pixel-wide representation that outlines the basic shape of the handwriting. Figure 2 gives an example of a skeleton obtained by line thinning. Various techniques exist for extracting word skeletons, such as diffusion maps [5] or line thinning [7, 10].

Once the skeleton is obtained, some tracing method is used to find an ordered path through the skeleton. As noted by Qiao and Yasuhara [16], skeleton tracing methods fall into two main categories: local tracing methods, which choose a direction at each junction in the skeleton based on local heuristics, and global searches, which create a graph model of the skeleton and then use search techniques to find an optimal path through the text.

The bulk of the work examined falls into these two categories, with a few exceptions. In the area of binarization, Yan



(a) A stroke with a gap between points *a* and *b*

(b) Bridging the gap

(c) Faint strokes with low SNR

(d) Intelligent Pen recovers faint or missing strokes

Figure 1: Intelligent Pen overcomes many of the pitfalls common to historical document images

and Leedham [20] employ an adaptive thresholding technique that subdivides the image into small regions and then examines features such as the stroke angle and edge strength to set the thresholding value for that region. Although their method does a good job separating handwriting pixels from the background it does not account for the presence of form lines and does not obtain the actual strokes of the handwriting.

Clawson and Barrett [4] attempt to distinguish handwriting pixels from form pixels by first registering a group of similar documents to a template image having the same form lines but with the handwriting removed. A sliding window is then compared against both the template and the source image, and areas where the document differs from the template are marked as handwriting. However this approach gives incorrect results for documents with folds, tears, or other non-handwriting areas that differ from the template.

3.1 *Skeletonization and Local Search*

Daher et al [5] obtain the skeleton using a diffusion map. They then perform an ordered trace the skeleton by first finding a start point, then using the gradient to calculate a direction and step distance. However, their goal is only to extract graphemes, not complete strokes, and their results are given in terms of the

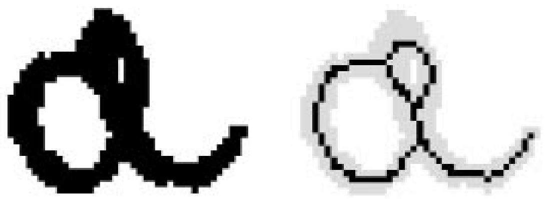


Figure 2: An example of a handwritten lower-case ‘a’ and its skeleton, obtained by line thinning. Taken from a figure in the paper by L’Homer [10].

number of graphemes extracted per page, which makes it unclear how effective their method is in obtaining complete strokes.

Pavlidis [13] uses a line adjacency graph, scanning horizontally on a binary image and finding sections of three or more adjacent dark pixels. Each of these sections becomes a node in the line adjacency graph and is connected to adjacent nodes immediately above and below it. Local heuristics are then used to identify path and junction sections to create a simplified graph. Although faster than other line thinning techniques, this method struggles with diagonals and curves and is ultimately unsuccessful at improving accuracy in handwriting recognition.

Lee and Pan [9] extract the stroke order by searching for an endpoint, then using local heuristics to trace back to the start point. They then reverse the order of this back-traced path and merge with adjacent paths to find a global ordering of the pixels.

In [3], Boccignone et al. use a line-thinning algorithm that preserves the local thickness of the line. This is done by labeling each pixel of the skeleton with its 8-distance from the nearest background pixel.

One unique variation is proposed by Lallican and Viard-Gaudin [8]. They use the grayscale image to extract edge pixels, then match each edge pixel with the one opposite it. The midpoints of the cross-sections of these pixels are used to form a pseudo-skeleton. They then extract full strokes by applying a Kalman filter to the pseudo-skeleton. Finally, they use a smoothness function to divide or combine these curves. While their method is successful in following strokes that undergo several changes in curvature it does not account for variations in pixel intensity. Their tests are confined to cleanly binarized images, making it unclear how they would perform on historical documents.

Doerman and Rosenfeld [6] also employ a method that uses cross-section lines connecting corresponding edge points. In their case, the handwriting is not represented as a skeleton but as a series of cross-sections. This allows easy computation of the direction of the handwriting (which should be perpendicular to the cross-section) as well as the thickness (which is the length of the cross-section). However, their method relies heavily on local heuristics, making it brittle in the presence of noise and degradation such as are found in historical documents.

3.2 Graph Search and Optimal Path Finding

Skeletons are often represented as graphs where each node is a path, junction, or endpoint section of the handwriting. These skeleton extraction techniques are then combined with graph search methods, most of which rely on the gradient and local stroke direction to determine which path to follow at each branch node.

Tan et al. [17] apply the skeletonization and graph search approach to handwritten Chinese characters. with the purpose of extracting the common components that make up each character. They test their method with four different skeleton extraction methods, with the best one able to accurately extract 90.7% of the strokes in the characters in their dataset of 341 Chinese characters. Their dataset consisted of clearly defined characters with thick strokes and no degradation issues.

Viard-Gaudin et al. [18] used a windowed scan to break the handwriting into segments, each of which is small enough to represent with a feature vector. They then used a graph search to connect the segments into strokes.

Qiao and Yasuhara propose multiple methods in [14], [15], and [16]. In [14] they use a combination of local heuristics and a global smoothness algorithm to compute the optimal path through the skeleton. In [15] they use a bi-directional search, picking a start and end point, then searching forward from the start point and backward from the end point and finally merging the two paths.

[16], [7], and [2] all treat the path traversal as an optimal Euler path problem. An Euler path is defined as a path through an undirected graph that passes through each node exactly once. In each case computational complexity becomes an issue, so the graph must either be reduced or searched hierarchically to reduce computational complexity.

The key assumption made by the algorithms discussed above is that the skeleton can be successfully extracted without loss of data or the introduction of noise, and the publications contain sparse information on what validation has been performed to this effect. To overcome these challenges a new approach is needed that is more robust to noise and works with older, degraded handwriting, and is tolerant to the existence of form lines that overlap the handwriting. Rather than create a skeleton or graph representation of the image, we propose an algorithm that would operate directly on the grayscale pixels of the original image.

The field of image segmentation provides inspiration for such an approach. In [11] Mortensen and Barrett introduce Intelligent Scissors, which uses a least-cost search algorithm to find the edges of an object of interest in an image based on a few seed points. This is done using a weighted sum of several image features.

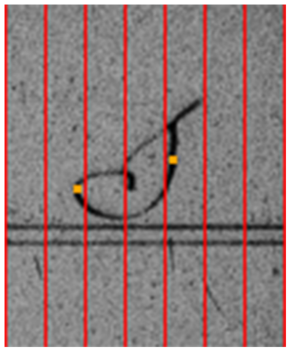
By choosing different features for the path-finding algorithm we repurpose the idea behind Intelligent Scissors to create a new algorithm for extracting strokes of handwriting, one that is piecewise optimal, operates on the original grayscale image, and is robust to noise, degradation, and other features of historical documents. Because it takes its inspiration from Intelligent Scissors, we adopt the name Intelligent Pen for our approach.

4 Intelligent Pen

Intelligent Pen performs a minimum cost path search using a cost function that is tuned for the domain of historical document processing. The goal is to create a complete trace of the handwriting for which the minimum cost paths are the ones that pass through the handwriting, ignoring noise and properly following the original path of the author’s pen, even in the presence of gaps and low-contrast segments. The process begins by expanding multiple wavefronts of minimum cumulative cost from multiple seed points, all in cost order. As wavefronts collide, paths coalesce or terminate due to comparatively high cost.

4.1 Seed Point Selection

To begin with, we need to choose seed points for the algorithm. To do this, we divide the image into several vertical strips, as in Figure 3a. We then use our least-cost equation to find the shortest path from the top of each stripe to the bottom. Figure 3b shows how the paths snap to any handwriting. For each stripe we then find the segment of the obtained path with the lowest cost, and choose the center of that low-cost segment as a seed point. Because these seed points lie in the center of a low-cost area of the path there is a high probability that they lie on a stroke of handwriting. Because additional seed points are generated automatically, as described below, we require comparatively few seed points to initiate the process.


(a) The image is divided into vertical strips, and the optimal path from top to bottom is found for each strip

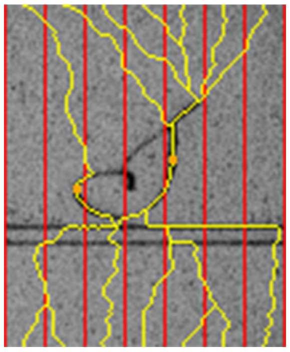

(b) The paths snap to the nearest handwriting, and the local minimum on each path is chosen as a seed point

Figure 3: Finding start points

4.2 Wavefront Expansion

To find the shortest path from one point to another we use a variation of Dijkstra's algorithm, using the cost function defined in Equation 3. To start, we initialize the cost C_{xy} for each point p in the image using the following sigmoidal equation:

$$C_{xy} = \frac{1}{(1+e^{-k(i-t)})} + F_{xy} \quad (1)$$

In Equation 1 i is the grayscale intensity of p, and t is the Otsu threshold of the image (as defined in [12]). Subtracting t from i in the equation creates the sigmoidal cost function centered at t shown in Figure 4. After some experimentation a value of k = 0.1 was shown to work well and was used for the examples in this paper. F_{xy} is a cost applied for form lines and varies for each row depending on the distance to the nearest form line.

Before running the algorithm we manually specify the y coordinates of the centers of any form lines and calculate F_{xy} for all pixels within 5 rows according to the following equation:

$$F_{xy} = \begin{cases} \frac{5-|(y-y_f)|}{5} & \text{if } (y-y_f) < 5 \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

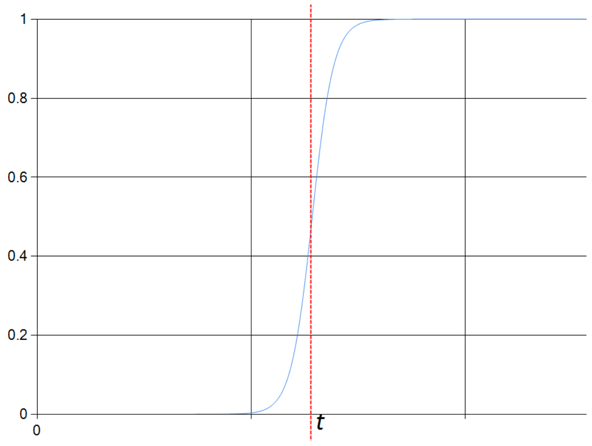


Figure 4: The cost is calculated using a sigmoidal function centered at the Otsu threshold t.

In Equation 2 y_f is the center point of the nearest form line.

When expanding a point p in the wavefront, the cumulative cost N for each neighbor n of p is calculated as follows:

$$N = d_n * C_n + C_p + 1 \quad (3)$$

The distance cost d_n is 1 for four-connected neighbors of p and $\sqrt{2}$ for diagonal neighbors of p. We add a constant value of 1 to the cost so that, in the presence of pure black (zero cost) paths we give preference to shorter paths.

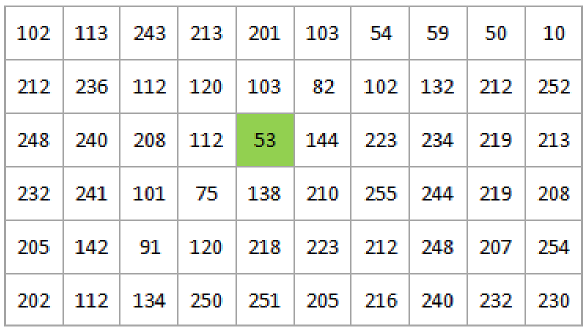
Figure 5 and Algorithm 1 give an example of how the cost wavefront expands. Having initialized our cost matrix as in Figure 5a, a start point is then expanded, and all its neighbors are added to the wavefront (yellow in Figure 5b-5d). Costs on the wavefront are maintained in sorted order and the node with the lowest cost is the next to be expanded. Any pixels neighboring this node that are not already in the graph are added, and their cost is set to be the cumulative cost of the shortest path back to the start point. This is done by adding the value from the initial cost matrix to the cumulative cost for the node being visited. Each node stores a pointer to the neighbor lying on its shortest path to the start point, so once the wavefront reaches a stopping criteria we have the shortest path to the seedpoint from anywhere in the wavefront.

It should be noted that due to Bellman's principle of optimality [19] every sub-path of an optimal path is itself an optimal path between its endpoints. We exploit this principle to stitch together piece-wise optimal paths.

Since the points with the lowest cumulative cost are expanded first, each wavefront tends to do a soft fill on the handwriting as it expands in all directions (Figure 6). The following sections describe how paths are grown and coalesced from the collision of multiple wavefronts to obtain a complete trace of the handwriting.

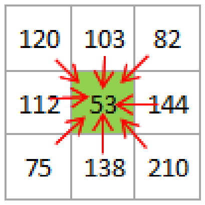
4.3 Parallel Path Expansion

After choosing start points in the fashion described above, we use each start point as the seed point for a wavefront, then allow all the wavefronts to expand simultaneously and in parallel in order of lowest cost. This parallel path expansion causes the algorithm to focus first on areas of lowest cost. If for some reason a

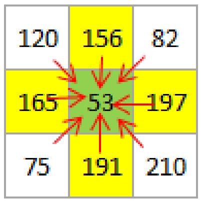


102	113	243	213	201	103	54	59	50	10
212	236	112	120	103	82	102	132	212	252
248	240	208	112	🟩 53	144	223	234	219	213
232	241	101	75	138	210	255	244	219	208
205	142	91	120	218	223	212	248	207	254
202	112	134	250	251	205	216	240	232	230

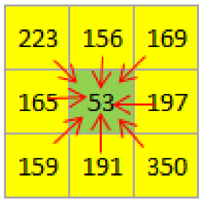
(a) Initial Cost Matrix: Costs ∼ pixel intensity. Green start point



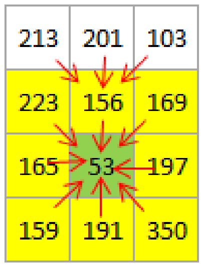
(b) Each neighbor sets a backpointer



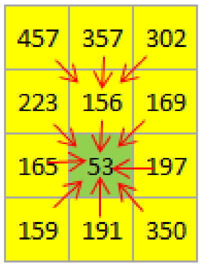
(c) Cumulative cost is calculated



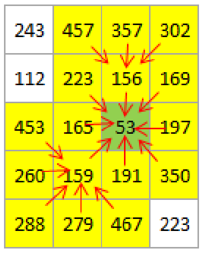
(d) Diagonal Cost given by $C_n * \sqrt{2}$



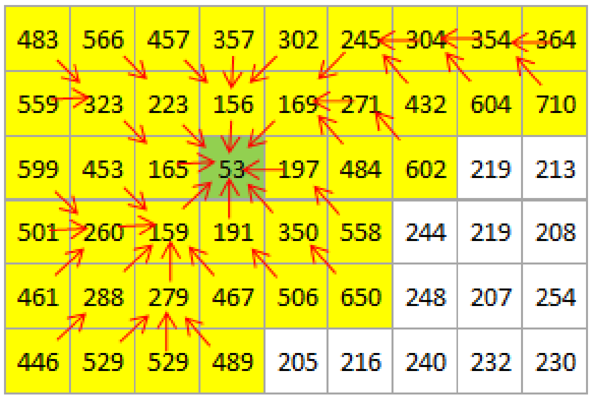
(e) Expand node with minimum cumulative cost (156)



(f) Cumulative costs updated



(g) Expand Node with next lowest cumulative cost (159, lower left)



(h) The wavefront expands along lower-cost (darker) strokes rather than outward in all directions

Figure 5: Wavefront Expansion Algorithm

Algorithm 1 Wavefront Expansion Algorithm

```
$W \leftarrow startPoints$ // W: set of all points in wavefront
$B \leftarrow startPoints$ // B: cost-ordered set of boundary points

while No stopping criteria reached do
    // Get lowest-cost, unvisited point
    // (53 In Fig 5b, 156 in 5e, and 159 in 5g)
    $p \leftarrow B[0]$
    $neighbors = getNeighbors(p)$

    for $n$ in $neighbors$ do
        if $n$ not in $W$ then
            $W.add(n)$
            $B.add(n)$
            // calculate cost based on distance (Fig. 5d)
            if $n.x \neq p.x$ and $n.y \neq p.y$ then
                $n.cost \leftarrow n.cost * \sqrt{2}$
```

seed point is in a high cost region of the image, it won't expanded as quickly as other seed points (Figure 6b).

A video of how the parallel wavefront expansion algorithm looks in real time can be found at `https://youtu.be/uSU9V-gaxls`. As shown in Figures 6c and 6d the wavefronts eventually collide and merge together, then continue to expand until reaching the end of the strokes. Note that Intelligent Pen does not follow the horizontal lines because we set those lines to be higher cost because they are not part of the handwriting (see Figure 9b).

4.4 Path Consensus

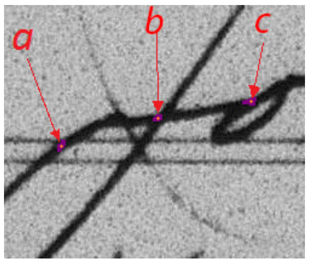
Now that the wavefronts are expanding in cost order, we need to set a stopping criteria to prevent them from completely filling the image. Therefore, the wavefronts only expand until we reach one of three stopping criteria.

1. The number of pixels in the wavefront reaches a certain predefined threshold.
2. One wavefront runs into another actively expanding wavefront (Figure 6c).
3. A wavefront collides with the edge of the image.

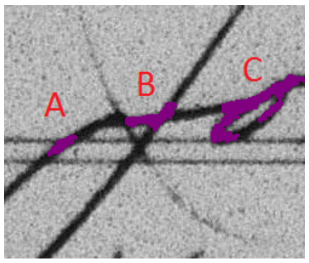
Once a stopping point is reached we use a contour following algorithm to obtain an ordered list of all the points on the outside border of the wavefront. Next, several "free points" (the green points in Figure 7a) are selected at regular intervals around this border. We then follow the backpointers from each free point back to the start point, which gives the shortest path for each free point (the yellow paths in Figure 7b). Points lying on one or more of these candidate paths are chosen as consensus paths (the red paths in Figure 7c). The red consensus points define the optimal path for this segment.

4.5 Creating New Seed Points

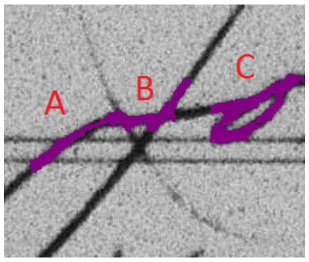
To generate new start points every consensus path is extended to the edge of the wavefront, as shown in Figure 8 and Algorithm 2. This is done by first finding the endpoint e of each consensus path (orange), then looking at each point p_i on the outside edge of the wavefront that contains e in its path back to the



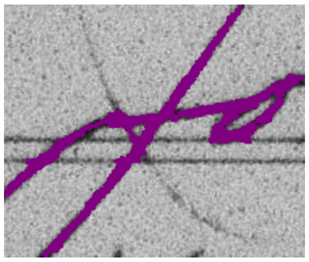
(a) Seed Points a, b, and c associated with purple wavefronts A, B, and C



(b) Wavefront C is on a lower-cost stroke and expands faster than A and B



(c) When wavefronts A and B meet, they merge and continue expanding



(d) The unified wavefront continues to expand along the lowest-cost strokes

Figure 6: Simultaneous Expansion of 3 Wavefronts A, B, and C

start point S (shown in blue in Figure 8). We then choose the p_i whose path from the start point has the lowest cost per pixel. The consensus path is therefore extended to contain all of the path from the start point to the chosen p_i (green), and p_i becomes a new start point. This process is repeated for each consensus path.

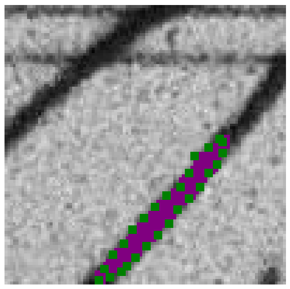
Algorithm 2 Extending a start point to the edge of the wavefront

```
Input: Start point $S$, Consensus path with endpoint $e$.
Output: Extended path with endpoint $e_2$ on wavefront border
  $P \leftarrow$ all points on edge of wavefront
  for $p_i$ in $P$ do
      Follow $p_i$'s backpointers to get shortest path to $S$.
      if $e$ is not on $p_i$'s shortest path to $S$ then
          Remove $p_i$ from $P$
  Order all $p_i$ in descending order by length of shortest path to $S$
  Choose first $p_i$ (i.e. with longest path) as new start point.
```

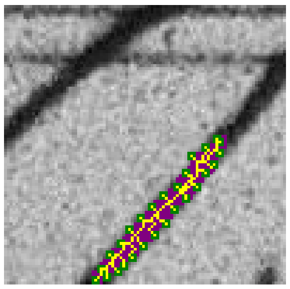
If two wavefronts collide, new start points are generated for each wavefront, though the point of intersection is discarded from the set of new start points.

Finally, every point within the wavefront is set to have infinite cost, guaranteeing that no point in the wavefront will be explored again. This is to prevent backtracking over the same section of handwriting multiple times. This can be safely done because of the piecewise-optimal nature of each path. Since each consensus path is an optimal path, we know that there are no additional useful paths to be found anywhere in the wavefront or they would already have been uncovered. Setting costs within the wavefront high also saves computation.

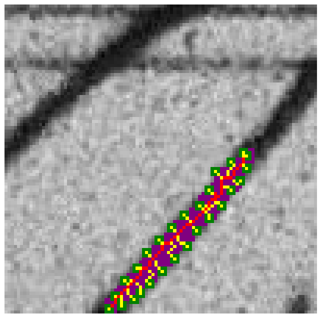
Figure 9 shows how the cost map changes over time as the wavefront expands.



(a) Free points (green) are selected at regular intervals around the wavefront's border



(b) For each of these free points the shortest path to the start point (yellow) is found



(c) Consensus points (red) appear on more than one of the yellow candidate paths and define the optimal path for this segment

Figure 7: Using path consensus to extract strokes

4.6 Stopping Path Growth

In addition, to save computational cycles and avoid exploring areas that have a low probability of handwriting we stop expanding a wavefront if it becomes roughly circular. This is due to the behavior of the wavefront expansion algorithm. When there is no handwriting near it the wavefront tends to expand in a circular pattern since all costs are roughly equal. Therefore, if a wavefront is circular it has a low probability of containing handwriting. To determine when to discard such circular wavefronts we calculate an inverse eccentricity value ecc^{-1} as the number of points in the wavefront (W) divided by the length L of the longest path within it (Equation 4).

$$ecc^{-1} = W/L \tag{4}$$

Wavefronts that are following a stroke of handwriting such as those in Figures 6 and 9 tend to be long and skinny, with a low ecc^{-1} value. On the other hand, Figure 10 shows an example of a (green) circular wavefront that was pruned because of its high ecc^{-1} value.

Since such circular wavefronts tend to occur at the end of a stroke it makes sense to prune them, and in practice this optimization improves the accuracy of Intelligent Pen, saving compute time and avoiding exploring areas of the image with no handwriting.

4.7 Path Pruning

As we go through the steps above we keep track of each consensus path we find. We then use three conditions to prune out

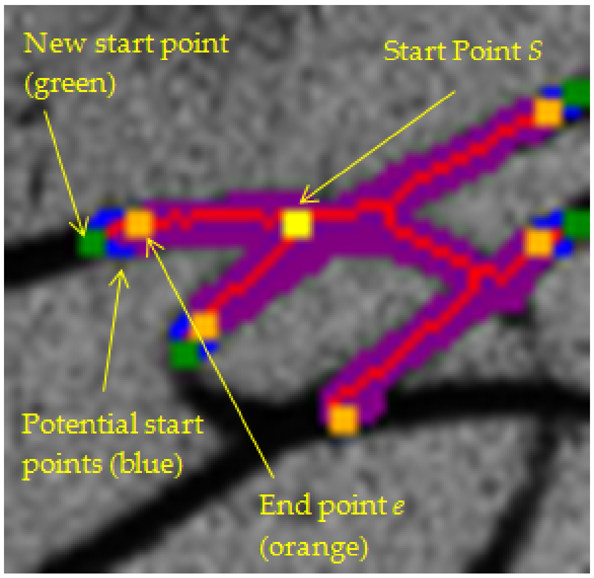
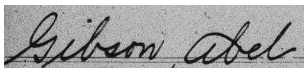
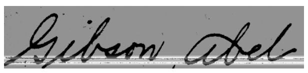


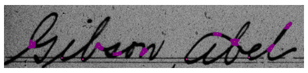
Figure 8: Consensus path end point e (orange) is extended to the edge of the wavefront by choosing the point p_i that is the furthest point from S that has e on its shortest path.



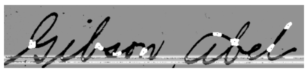
(a) Original handwriting image



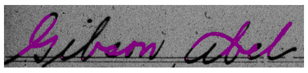
(b) Initial Cost Map. Form lines set to high cost.Note high-cost area around the form lines



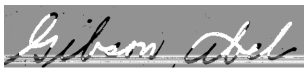
(c) The wavefront after several timesteps.



(d) Cost map with wavefront points set to infinite cost.



(e) The wavefronts after several more expansions



(f) The cost map continues to expand with the wavefronts.

Figure 9

false positives, as outlined in Algorithm 3. These conditions are the cost-per-pixel ($p_i.cpp$ in Algorithm 3), inverse eccentricity ($p_i.ecc$), and order expanded ($p_i.order$). The cost per pixel is defined as the cumulative cost of the path divided by its length. The inverse eccentricity of a path is defined as the number of pixels in the wavefront divided by the length of the path and is used to penalize circular wavefronts (see section 4.6). We use the order in which the paths were found as a third pruning criterion because the wavefronts are expanded in cost order, so the last ones to be explored are less likely to contain handwriting.

As outlined in Algorithm 3, once these three values are calculated for each path we normalize them, then use a weighted sum of the three to determine a final pruning cost. We can then apply Otsu thresholding to this set of pruning costs to remove any high-cost paths.

Figure 11 shows how Algorithm 3 is used to detect false positives. Figure 11a shows handwriting with all the strokes initially

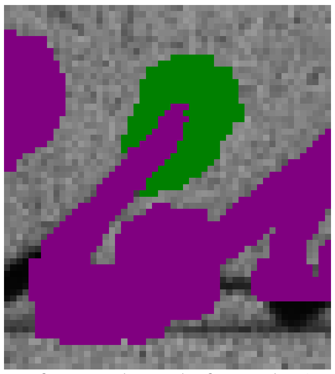


Figure 10: A wavefront at the end of a stroke (green) expands in a circle and is pruned.

Algorithm 3 Pruning High Cost Paths

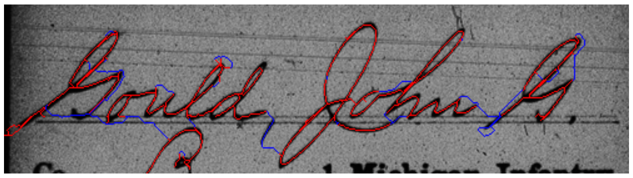
```
Given a list of consensus paths $P$, and weights $w_c, w_e, and w_o$:
$i \leftarrow 0$
for $p_i$ in $P$ do
    $p_i.cpp \leftarrow \frac{p.cost}{p.length}$ // cost per pixel of path
    $p_i.ecc \leftarrow \frac{p.wavefront.count}{p.length}$ // eccentricity of p's wavefront
    $p_i.order \leftarrow i$ // order in which paths were found
    $i \leftarrow i+1$
for $p_i$ in $P$ do
    $p_i.cpp \leftarrow normalize(p.cpp)$
    $p_i.ecc \leftarrow normalize(p.ecc)$
    $p_i.order \leftarrow normalize(p.order)$
    $p_i.pruningCost \leftarrow (w_c * p_i.cpp) + (w_e * p_i.ecc) + (w_o * p_i.order)$
$t \leftarrow getOtsuThreshold(paths)$
for $p_i$ in $P$ do
    if $p_i.pruningCost > t$ then
        $P.remove(p_i)$
```

extracted by Intelligent Pen. The highest cost paths, marked in blue, are removed by Otsu thresholding, leaving the results shown in 11b. The qualitative examples shown in this paper all used pruning weights of $w_c = 1$, $w_e = 0$, and $w_o = 0$, but we are currently exploring ways to automatically tune these weights for better results.

4.8 Additional Passes

In many cases one pass of the algorithm will miss some portion of the text due to natural breaks in the handwriting, an incomplete set of seed points, etc. However, the algorithm can be repeatedly applied, with each iteration preserving information from the previous one. Each pass uses the ending cost map from the previous pass where wavefronts associated with previously explored costs are set high as in 9. This minimizes computation on subsequent passes by focusing the search only on unexplored regions.

Figure 12 shows how we obtain start points for additional passes. After the initial pass (Figure 12b), we look for unexplored handwriting by Otsu thresholding the image using only the unexplored points as input, then finding all the connected components of unexplored pixels (Figure 12c). The final result, shown in Figure 12d, shows how the extra passes allowed us to recover some


(a) Each path is assigned a final cost based on cost per pixel, eccentricity, and the order found. High cost paths (blue) are removed using Otsu thresholding.

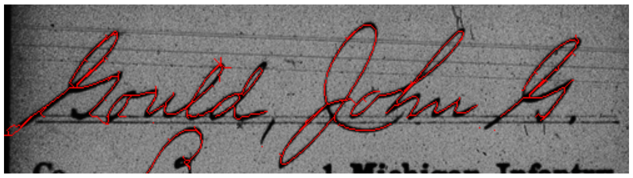

(b) Results after removing high-cost strokes.

Figure 11: Pruning high-cost paths

of the missing strokes.

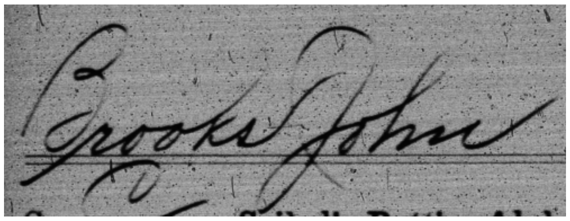
4.9 Combining and Ordering Paths

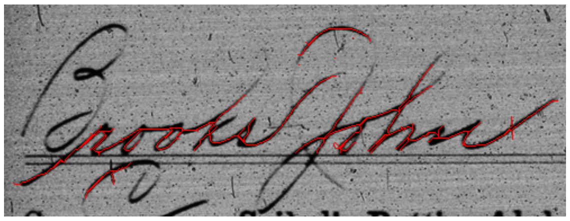
Having obtained a set of consensus paths, we then group them together into larger connected components. In an ideal situation each connected component would represent an entire cursive word. For example, in Figure 13, the name "Gibb" is all part of one connected component (shown in green), while the name "John" is part of a separate connected component (blue). In some cases extra connected components can be created for capital letters that don't connect to the word, as well as in cases where wavefronts failed to connect properly (such as the base of the 'J' in "John").

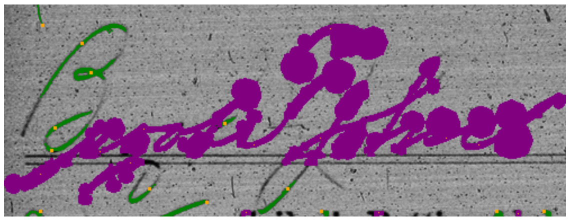
As a final step, Intelligent Pen performs a basic ordering of the pixels in each connected component. For each connected component we build an ordered set of pixels as outlined in Algorithm 4. We start by adding the leftmost pixel to a stack. While the stack is not empty we pop a pixel off the stack and add it to the ordered set, then see if there's a neighboring pixel not in the set already. If there's only one, we push it onto the stack and continue. If there are multiple neighbors not already in the stack or the ordered list then we know we're at an intersection, and we want to take the branch that continues in the same direction as the path we've been following. To do this we find the trajectory of the past 10 pixels on the path, then calculate the predicted position of the next point p_{i+1}. Then for each neighbor n_i we calculate the difference d_i between n_i and p_{i+1}. We then sort the neighbors by d_i, and push them onto the stack in that order. As a result, the neighbor that changes direction the least is processed first, giving the stroke continuity as shown in Figure 14.

5 Results

Two datasets were chosen to validate the effectiveness of the algorithm: one provided by an ICDAR contest[1], and one provided by FamilySearch. The ICDAR dataset provided a quantitative look at the algorithm's performance, while the FamilySearch Dataset provided several qualitative demonstrations of where Intelligent Pen outperforms existing algorithms.


(a) Original Image


(b) The first pass misses the top half of the 'B' and part of the 'J'


(c) Previously visited pixels (purple) are ignored on the second pass. New start points (orange) are chosen using the unvisited connected components (green)

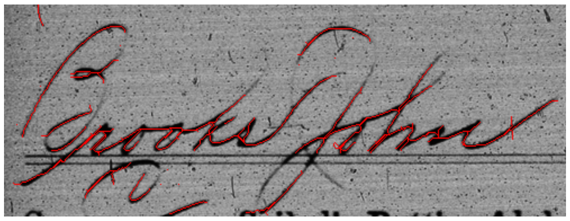

(d) The second pass recovers additional strokes that were missed on the first pass

Figure 12: Using additional passes to pick up missing strokes

5.1 Qualitative Results

The second dataset was a set of cursive names from historical documents. It was provided by FamilySearch along with the transcriptions of the names obtained by volunteer indexers. The images in this dataset all have noisy backgrounds, form lines cutting through the handwriting, ascenders and descenders. Many of the images have low contrast between the handwriting and the background, as well as gaps in the strokes and other imperfections in the writing. Figure 15 shows two pieces of handwriting from this dataset, along with the strokes obtained by Intelligent Pen after one pass. Figures 11 and 13 are also taken from the same dataset.

Intelligent Pen recovers much of the handwriting while not including the form lines. The J in Figure 11 is particularly successful, fully recovering the descender and ascender while not following the form lines or the noisy horizontal lines. In Figure 15b, Intelligent Pen correctly recovers all the strokes, but also picks up

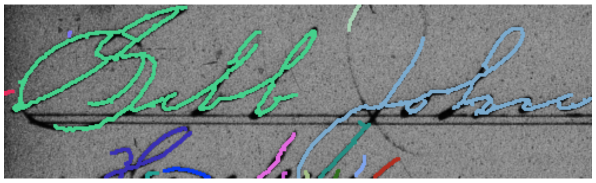


Figure 13: Once the strokes are obtained they are joined together into connected components.

Algorithm 4 Ordering Paths

```
1:  Given a list of connected points $C$, an empty list of points $O$,
    an empty stack $S$, and an integer $d$:
2:  Sort $C$ from left to right
3:  Grab first (leftmost) point in $C$ and push it onto $S$
4:  while $S$ is not empty do
5:      Pop point $p$ off of $S$
6:      Add $p$ to $O$
7:      Get all points $N$ in $C$ that neighbor $p$ and are not in $O$ or $S$
8:      if There is only one point $n_i$ in $N$ then
9:          Push $n_i$ onto $S$
10:     else
11:         Fit polynomial function $f$ to past $n$ points
12:         for $n_i$ in $N$ do
13:             Find next $n$ points on same path as $n_i$
14:             for Each of these $n$ points $p_i$ do
15:                 $d_i = p_i - f(p_i)$
16:             Calculate sum $d$ of all $d_i$
17:         Order $N$ by $d$
18:         Push each $n_i$ onto $S$
```

some false positives on the inside of the ’B’ as well as on the ’k’ and ’h’. Figure 15d Is a nearly perfect example, though it does pick up a stray piece of form line on the right edge of the image.

Figure 16 shows a few more examples. Figure 16b returns a fairly complete set of strokes, but misses a section of the ’G’. The connecting line between the ’l’s and the bottom half of the ’m’ are missing because of the high-cost from their proximity to the form line. The ’e’ and ’g’ in Figure 16d are likewise missing strokes due to pruning high-cost sections containing the form line.

5.2 Quantitative Results

The first dataset examined was taken from a contest in connection with ICDAR 2013[1]. It consisted of 605 signatures that were acquired using a Wacom Intuos4 Large digitizing tablet and a Wacom Inking pen. The signatures were written on a blank paper on top of the tablet to acquire the online information, then the paper was scanned to provide the data in an offline format. For our analysis we used the 605 signatures that were provided with numeric ground truth data (x, y coordinates).

Because the signatures were captured in a clean environment using ink on a blank paper they lack several of the challenges common to historical documents such as background noise, form lines, and degradation. However, there were several examples of low-contrast, light strokes and minor gaps in the handwriting.

The online data consisted of a series of points representing

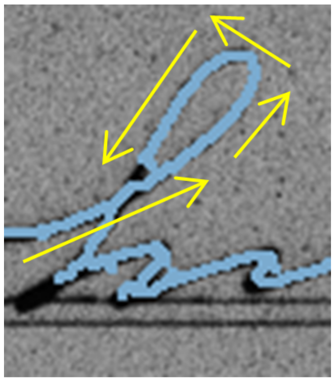


Figure 14: The strokes are ordered to correctly navigate junctions such as the loop in ‘h’

Mean Distance:	2.584
% within 1 pixel:	12.8%
% within 2 pixels:	45.6%
% within 3 pixels:	68.4%
% within 4 pixels:	79.4%
% within 5 pixels:	86.5%

Table 1: Comparison of Ground Truth for the ICDAR Dataset with strokes extracted by Intelligent Pen. All numbers are averaged over all 605 signatures

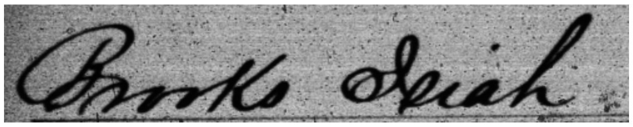
the trajector of the pen, marked as green squares in Figures 17 and 18. For each image, Intelligent Pen was used to extract a series of pixels representing the strokes of the signature, marked in red in the examples.

For each ground truth point the distance was calculated to the nearest stroke pixel identified by Intelligent Pen. Table 1 gives a summary of the results:

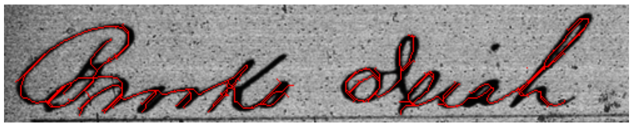
One thing we observed when evaluating these numerical results is that, perhaps due to parallax problems inherent in the capture method employed for these signatures, the online data does not necessarily match up with the scanned images of the ink signatures. For example, Figure 18 shows a representative case where the online data was a few pixels off from the handwriting. A visual examination of the results shows that the results obtained by Intelligent Pen, because they followed the handwriting, were a few pixels off from the ground truth but overlay very closely with the actual strokes. We believe that without the parallax offset, the agreement between the green points and Intelligent Pen would be much closer.

6 Future Work

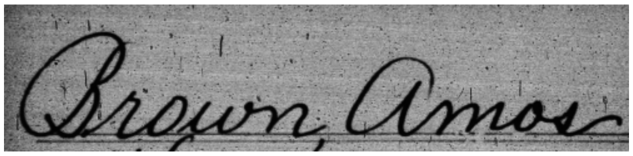
As has been shown, Intelligent Pen has great potential to recover strokes from degraded images of historical documents. One limitation of the results presented in this paper is that the form lines are manually specified. To fully automate the process a better method for initializing the cost map for form lines is needed. We are also currently exploring a new method for training the cost function that uses the cumulative histogram of several training images instead of a fixed sigmoidal function.



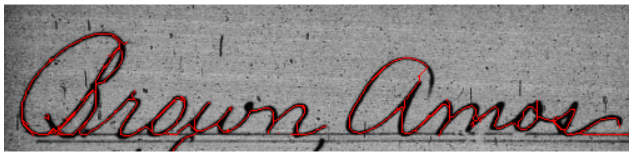
(a)



(b)



(c)



(d)

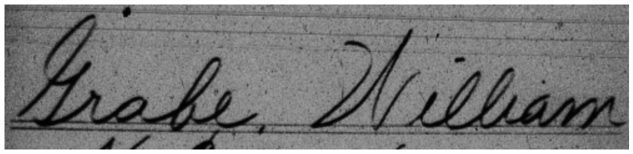
Figure 15: Examples of Intelligent Pen's performance on images in the FamilySearch dataset

7 Conclusions

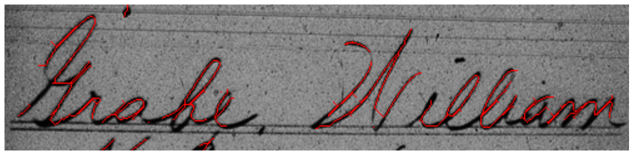
While it's not yet a perfect solution, we have shown that Intelligent Pen is capable of overcoming many of the barriers to handwriting extraction in historical document images. Its success on the ICDAR dataset shows its ability to perform well on handwriting for which it has not been trained with good quantitative comparison. Analysis of the results obtained on the FamilySearch images shows its potential for opening doors to the challenging problem of handwriting recognition in older, noisy historical documents.

8 Acknowledgments

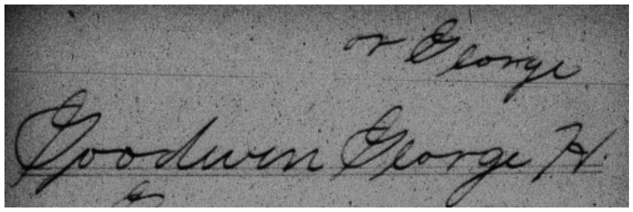
We would like to thank FamilySearch for providing a valuable dataset of handwritten proper names from historical documents.



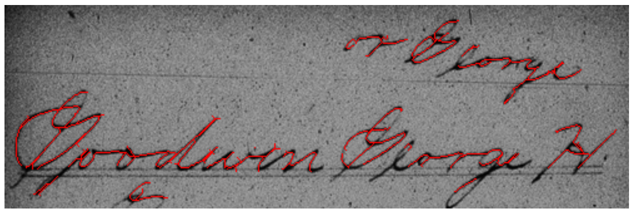
(a)



(b)



(c)



(d)

Figure 16: Some more examples of Intelligent Pen's performance on FamilySearch images.

References

[1] ICDAR 2013 - Handwriting stroke recovery from offline data. `https://www.kaggle.com/c/icdar2013-stroke-recovery-from-offline-data`. [Online; accessed 13-June-2015].

[2] C. Allen and A. Navarro. The recognition of roman handwritten characters using dynamic information recovery. In *Image Processing and Its Applications, 1997., Sixth International Conference on*, volume 2, pages 741–745 vol.2, Jul 1997.

[3] G. Boccignone, A. Chianese, L. P. Cordella, and A. Marcelli. Recovering dynamic information from static handwriting. *Pattern Recognition*, 26(3):409–418, Mar 1993.

[4] R. Clawson and W. Barrett. Extraction of handwriting in tabular document images. *Family History Technology Workshop at Rootstech*, 2012.

[5] H. Daher, D. Gaceb, V. Eglin, S. Bres, and N. Vincent. Ancient handwritings decomposition into graphemes and codebook generation based on graph coloring. In *Frontiers in Handwriting Recognition (ICFHR), 2010 International Conference on*, pages 119–124, Nov 2010.

[6] D. Doermann and A. Rosenfeld. Recovery of temporal information from static images of handwriting. In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR '92., 1992 IEEE Computer Society Conference on*, pages 162–168, Jun 1992.

[7] T. Huang and M. Yasuhara. A total stroke slalom method for searching for the optimal drawing order of off-line handwriting. In *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on*, volume 3, pages 2789–2794 vol.3, Oct 1995.

[8] P. Lallican and C. Viard-Gaudin. A kalman approach for stroke order recovering from off-line handwriting. In *Document Analysis and Recognition, 1997., Proceedings of the Fourth International Conference on*, volume 2, pages 519–522 vol.2, Aug 1997.

[9] S. Lee and J. Pan. Offline tracing and representation of signatures. *Systems, Man and Cybernetics, IEEE Transactions*

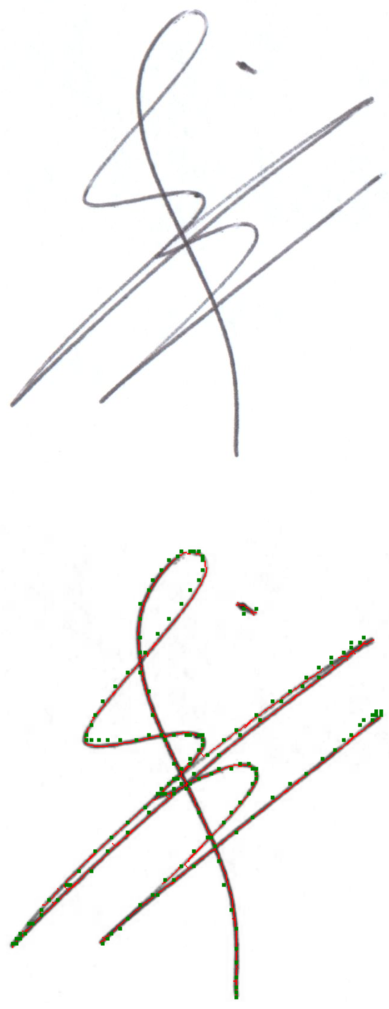


Figure 17: A signature from the ICDAR dataset, along with the corresponding ground truth data (green) and the stroke information extracted by Intelligent Pen (red)

on, 22(4):755–771, Jul 1992.

[10] E. L'Homer. Extraction of strokes in handwritten characters. *Pattern Recognition*, 33(7):1147 – 1160, 2000.

[11] E. N. Mortensen and W. A. Barrett. Interactive segmentation with intelligent scissors. *Graphical models and image processing*, 60(5):349–384, 1998.

[12] N. Otsu. A threshold selection method from gray-level histograms. *Systems, Man and Cybernetics, IEEE Transactions on*, 9(1):62–66, Jan 1979.

[13] T. Pavlidis. A vectorizer and feature extractor for document recognition. *Comput. Vision Graph. Image Process.*, 35(1):111–127, July 1986.

[14] Y. Qiao and M. Yasuhara. Recovering dynamic information from static handwritten images. In *Frontiers in Handwriting Recognition, 2004. IWFHR-9 2004. Ninth International Workshop on*, pages 118–123, Oct 2004.

[15] Y. Qiao and M. Yasuhara. Recover writing trajectory from multiple stroked image using bidirectional dynamic search.

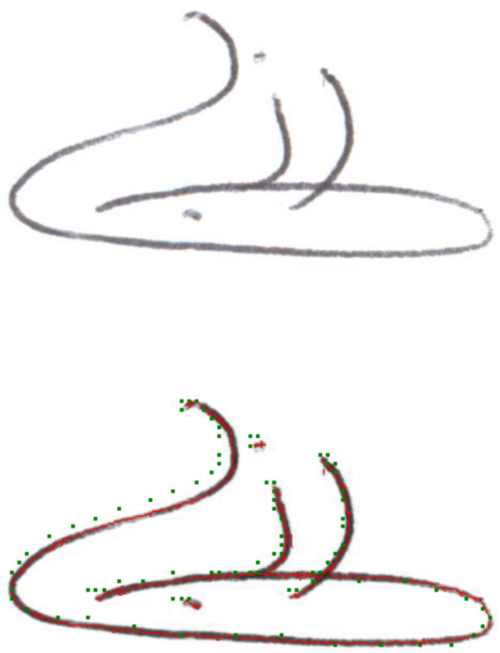


Figure 18: A signature from the ICDAR dataset in which the recorded trajectory did not line up well with the ink, introducing a common source of error into the ground truth.

In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 2, pages 970–973, 2006.

[16] Y. Qiao and M. Yasuhara. Recovering drawing order from offline handwritten image using direction context and optimal euler path. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 2, pages II–II, May 2006.

[17] J. Tan, J. Lai, W.-S. Zheng, and C. Suen. A novel approach for stroke extraction of off-line chinese handwritten characters based on optimum paths. In *Frontiers in Handwriting Recognition (ICFHR), 2012 International Conference on*, pages 786–790, Sept 2012.

[18] C. Viard-Gaudin, P.-M. Lallican, and S. Knerr. Recognition-directed recovering of temporal information from handwriting images. *Pattern Recognition Letters*, 26(16):2537 – 2548, 2005.

[19] Wikipedia. Bellman equation — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Bellman_equation#Bellman.27s_Principle_of_Optimality`, 2015. [Online; accessed 21-May-2015].

[20] C. Yan and G. Leedham. Decompose-threshold approach to handwriting extraction in degraded historical document images. In *Frontiers in Handwriting Recognition, 2004. IWFHR-9 2004. Ninth International Workshop on*, pages 239–244, Oct 2004.

9 Author Biography

Kevin Bauer received a Bachelors in Computer Science from

Brigham Young University in 2011, and is currently working on a Master's degree specializing in historical document image processing.

Dr. Barrett is a Computer Science Professor at Brigham Young University.