Handprinted Digit Recognition By Stroke Tracing

J.R. Parker
Laboratory for Computer Vision
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada

Abstract:

A structural method of recognizing handprinted digits is described, in which the strokes comprising the digits are decomposed into line segments. The lines are matched against stroke templates, beginning at a ‘pen down’ point and traced as far as possible. Both direction and length are important to the template match. Error rates of 3.3% are reported.

The basic idea behind *structural* pattern recognition is that objects are constructed from smaller components (features) using a set of rules[3, 4]. Characterizing an object in an image is a matter of locating the components, which at the lowest level are features, and constructing some representation for storing the relationships between them. This representation is then checked against known patterns to see if a match can be identified. Structural pattern recognition is, in fact, a sophisticated variation on template matching, one which must match relationships between objects as well as the objects themselves. The problems involved in structural pattern recognition are two: locating the components, and finding a good representation for storing the relationships between the components.

1. Shape Tracing

The vector template approach [5] essentially attempts to compare a set of standard line drawings of characters against the incoming data. A related method, which will be referred to as *shape tracing*, attempts to draw a character over top of the incoming data to see how well the drawing matches the data. This was suggested to me by my six year old daughter, who is already an expert at digit recognition. I noticed that when I asked her how she knew a particular digit was a six she traced over it with a pencil or her finger. It is possible that the pattern implicit in a six was stored by her as a pen motion.

With this in mind, an effort was made to characterize the motions involved in drawing all ten digits. Each motion begins at a start point, which is a point at which the pen would first touch the paper in making a stroke. Motion is specified by giving a direction only, or a series of directions for continuing the stroke. Directions are crudely specified as being one of eight possible: right-up (0-45 degrees), up-right (45-90), up-left (90-135), left-up (135-180), left-down (180-225), down-left (225-270), down-right (270-315) or right-down (315-0). Motion is approximated by straight lines; of course, curves exist in handprinted characters, but most computer software would plot a curve as a sequence of straight lines.

An example of how to draw a digit appears in Figure 1. The digit, in this case a ‘6’, begins in the upper right of the canvas, where the pen is put to paper. Drawing the digit proceeds as follows: left-down, down-left, down-right, right-down, up-right, up-left, left-up, and left-down. Not all sixes follow exactly this path, so the tracing of the path must have a certain flexibility. For example, the light grey line represents an alternate path that could just as easily occur, and should also be allowed in a six. The line lengths are not all crucial, but if all of the lines were the same length, the result would have been a zero instead of a six. It is clear that some effort must be made to characterize the key lines and their relative lengths.

Thus, the templates in this particular matching scheme consist of sets of pen motions, and the recognizer must attempt to measure how well each set of motions matches what is actually seen in the input image. The matching step turns out to be hard to do in software, and seven versions of the program were tested before coming up with a strategy that worked.

The first step is to identify the points where the pen first touched the paper. A bad *start point* can result in a very poor match with a perfectly good input glyph. From there, straight lines are traced from each begin point, then from the ends of those lines, and so on until all possible lines (linear features) have been located. Then the recognizer attempts to trace each possible digit over the extracted linear features, starting at each of the identified start points. Each of the three steps above is sufficiently complex to warrant a more detailed explanation.

2 Locating Start Points

When trying to identify stroke starting points it may be best to ask what the properties of such a region are in terms of local pixel measurements, since pixels are the primitives most easily accessed. When looked at this way, two solutions come to mind: the first uses the endpoints of the skeleton of the image[1, 2, 8], and the second uses a small window and looks for the characteristic ‘bump’ seen at a start point.

Using the skeleton endpoints is likely to produce spurious results. Consider, for example, the raster glyphs seen in Figure 2. The first set (a-c) have rather good skeletons, and the start points can be found quite simply by looking for 1-connected pixels in the skeleton. However, the second set (d-f) possess artifacts created by the thinning process that have resulted in an extra 1-connected pixel at the end of a short line segment. Some of these can be detected and removed simply by noting that an end pixel should not appear within a small distance of a point where two lines join - a join point is identified by the fact that it has 3 or more neighbors. This works for Figures 2d and 2e, but not for 2f. In addition to artifacts, the method can be fooled by join points having extensions from both ends.

The second method for identifying start points is to use a small square window into the raster input image. The window is moved to all possible locations and the number of black to white transitions seen on the boundary of the window are counted. If a stroke endpoint is within the window then the number of such transitions should be one, since the stroke ends within the window. To make sure that the window covers enough of the stroke to be significant the center pixel in the window must also be black. This will be called the *moving boxes* algorithm for obvious reasons. A single transition implies a

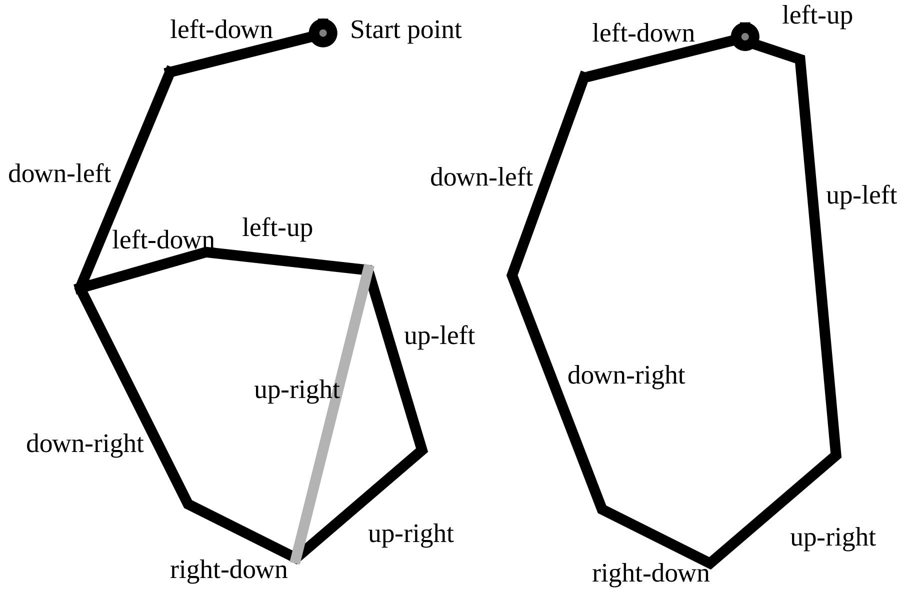


Figure 1- Drawing a six by specifying the start point and pen motions. A zero can be described by identical pen motions, but some of the lines are different lengths.

possible start point at or near the selected pixel. After all pixels are tested the largest clusters of possible start pixels are selected as actual starts; the centre or mass of each select cluster is used as the actual start point.

It is interesting to note that if only the starts detected by **both** algorithms are used the number of false starts is reduced from 6 to 2, in the case of the thinning method, and from 3 to 2 in the case of the moving boxes method. It was therefore decided to use both methods to detect start points.

3 Tracing Linear Features

Tracing lines in a thinned glyph is relatively easy[3,7], but the result is not good enough for recognition due to irregularities in the skeleton introduced mainly by defects in the outline. A tracing scheme that avoids thinning was devised, and is based on knowing the start points in advance. From each start point, the *longest line that consists of only black pixels* is located[4]. Its start point is marked as now being used, and the pixels near the line are also marked. Pixels on the boundary of the glyph are not allowed to be end points, and so the tracing of the next line resumes from an interior black pixel connected to the previous line.

Continuing from a line endpoint is done as before, by drawing the longest line on black pixels starting at that endpoint. However, pixels that are marked are counted *against* the length of the line being traced. This means that a line on black pixels is to be preferred over one containing marked ones. In addition, line endpoints will be connected to each other when possible; that is, when the connection is on black pixels. This is done even if the connection is not the longest possible black line.

The detailed method for tracing linear features from a raster digit image is as follows:

1. Determine the bounding box for the image, and get a width estimate for the strokes.
2. Locate the stroke start points, as described in section 2
3. Dilate the image by 2 pixels, and erode by 1 pixel. Now mark boundary pixels so that they can be easily identified.
4. From each start point, determine the longest line that consists of black pixels only, and save it as a stroke.
5. For each line found in step 4, mark the pixels near the extracted line as used. Also mark the start points as used.
6. For each unmarked stroke endpoint, determine whether a black line can be drawn to another unused endpoint. If so, save the resulting linear feature, mark the pixels near the extracted line as used, and mark all relevant endpoints as used.
7. For each unmarked stroke endpoint, determine the longest line that consists of black pixels that starts at that endpoint. Save the resulting linear feature, mark the near the extracted line as used, and mark all relevant endpoints as used.
8. Repeat from step 6 until either no unused endpoints remain or no black pixels remain.

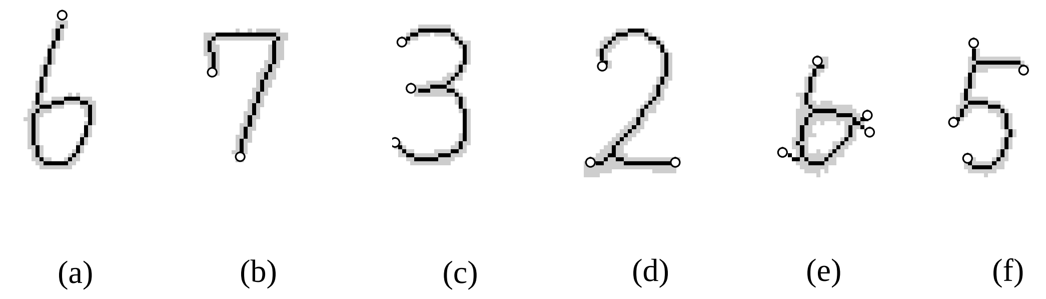
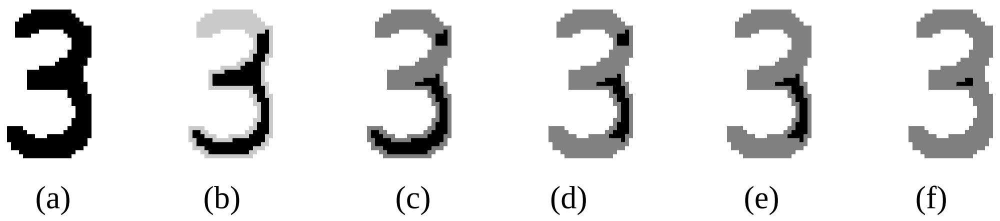


Figure 2 - Locating start points by finding 1-connected pixels in the skeleton. The cases in (a)-(c) work fine. (d) shows a ‘necking’ effect, creating a start in the lower left. (e) shows thinning artifacts caused by a rough outline. (f) suffers from line extension on the two left-most starts.



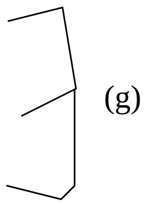


Figure 3 - (a) The original raster glyph. (b) After tracing the longest black line from the upper left start point. (c) After tracing a line from the middle start point. (d) After tracing a line from the lower start point. (e) After connecting the end points of the lines found in b and c. (f) After growing a line from the end of the middle line. (g) The extracted linear features, including a small connection between the lower and middle portions of the glyph which contained no black pixels.

The difficult part of the algorithm above is actually the marking of the used pixels. The linear feature is a thin line, consisting of relatively few pixels. However, if the pixels neighboring the line are not properly marked then the longest line from an end pixel may very well be another line back to a pixel very near the original start. The answer was to create a rectangle oriented along the direction of the linear feature and having the same length. The width was varied until the rectangle covered the boundary of the digit in that local region as well as possible. Then all pixels that were within that polygon (the rectangle) were marked, as were any pixels around the start points, and any small, isolated black regions caught between the rectangle and the boundary. One example of the feature tracing process can be seen in Figure 4, and it can be seen from this that the pixel marking process seems to be successful.

The procedure followed for zeros and eights is only slightly different, due to the fact that neither digit generally has a start point. If no start point is found then one is created by slicing through the glyph vertically along its own axis and using the black pixel in the centre of the first stroke encountered.

4 Drawing The Digits

At the point where the recognizer attempts to draw digits, the data has been distilled down to a few line segments and start points. The templates are stored as code, at least in the initial version of the program; that is, the procedure that attempts to recognize a six has the template represented as a sequence of procedure calls, each designed to trace a part of the digit over the linear features. Training this recognizer is therefore a bit difficult. Strokes are traced from a specific start point, and each digit has a different collection of legal starts. This fact can be used in the classification; a character having four start points is unlikely to be a zero, for example. The number of features can also be used in this way. The only digit allowed to have a single feature is a ‘1’.

The tracing sequence is fixed as a C-code description for each of the digits. One way to describe the templates is to display the lines allowed by each template, and label each line d0, d1, d2 ... d_n as seen in Figure 4. The lines associated with each label are now clearly seen, as are the reasons for the constraints placed on the various values of d_i.

Eights presented unique difficulties in tracing. There was a larger than expected variety in the shape of the eights seen in the sample data set, and that combined with the fact that an eight has a larger number of linear features than any other digit made them impossible to trace both unambiguously and reliably. The solution was, arguably, a ‘hack’, but nonetheless a reliable one. An eight will be a character that more or less agrees with a zero, but has a black region in the middle. This simple process correctly classifies 95% of the ‘8’ digits.

The classifier described above was used to classify a set of 1000 digit images and gave the following results:

Table 1: Shape Tracing - Digit Recognition Rates

	0	1	2	3	4	5	6	7	8	9
Correct	100%	94%	92%	99%	90%	94%	100%	88%	99%	98%
Reject	0	0	5%	1%	4%	3%	0	0	0	0

This gives an overall error rate of 3.3%. The low recognition rate for seven digits might be improved on in a number of ways, the simplest of which is to use a ratio R of the height of a character to its width. From the confusion matrix in Figure 5 it can be seen that seven digits are mistaken for ones in a large number of test cases. By using R>2.5 to further classify a digit as a one, the recognition rate for seven can be increased to 94%, and using holes increases the rate to 97%.

5 Conclusions

A method for recognizing digits by tracing their shapes has been described. The algorithm yields near human recognition rates on isolated handprinted digits, and is an integral component of a multiple/

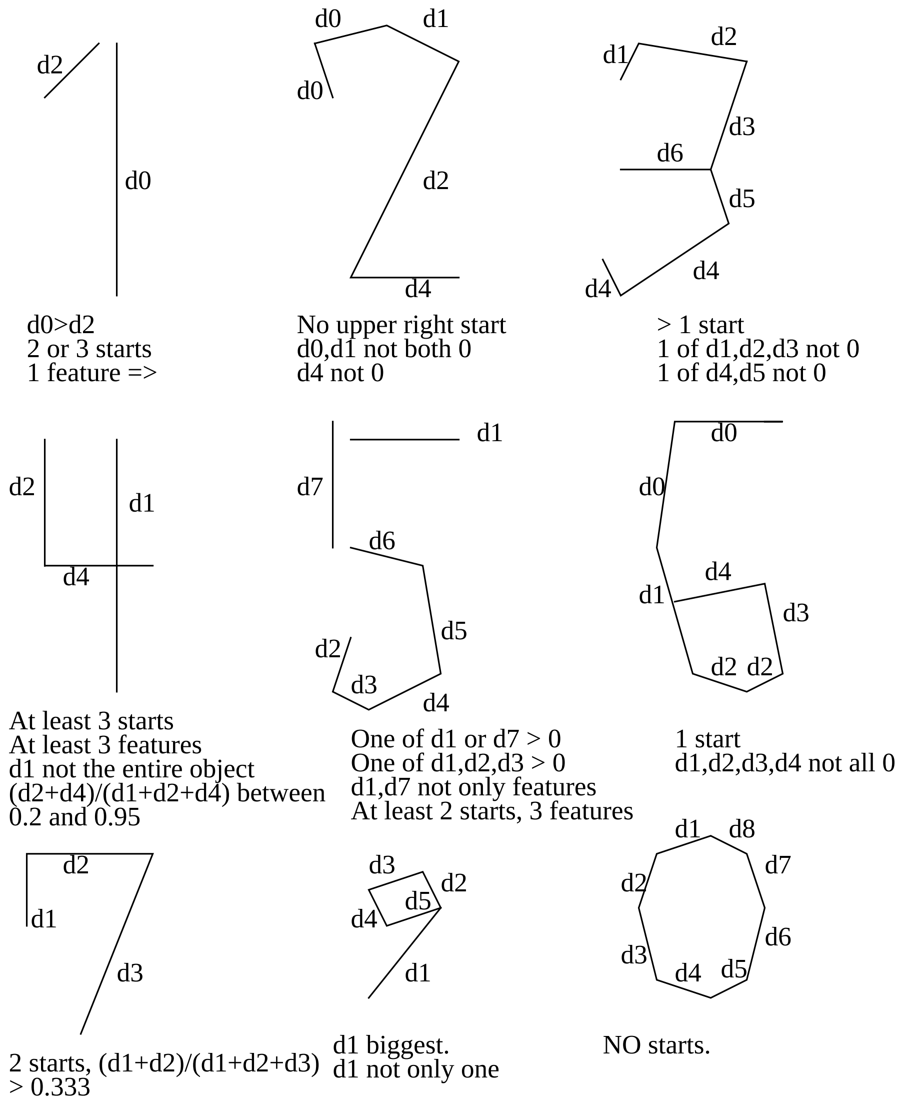


Figure 4 - Tracing patterns for digit recognition.

parallel handprinted digit recognition system having an overall measured error rate of 0.1% [6]. The technique is now in the process of being applied to the classification of symbols in engineering drawings and in printed music documents.

6 References

1. Holt, C.M. et al, An Improved Parallel Thinning Algorithm, CACM Vol 30 No. 2, 1987. pp 156-160.

2. Kwok, P.C.K., A Thinning Algorithm by Contour Generation, Communications of the ACM, Vol. 31 No. 11, November, 1988. Pp 1314-1324.

3. Lam, L. and Suen, C.Y., Structural Classification and Relaxation Matching of Totally Unconstrained Handwritten Zip-Code Numbers, Pattern Recognition, Vol. 21 No. 1. pp. 19-31, 1988.

4. Mori, S., Yamamoto, K., and Yasuda, M., Research on Machine Recognition of Handprinted Characters, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-6 No. 4, July 1984. pp 386-405.

5. Parker, J.R., Vector Templates and Handprinted Character Recognition, Proc. 12th IAPR Conference on Pattern Recognition, Jerusalem, Israel. Oct 9-13, 1994.

6. Parker, J.R., Recognition of Hand Printed Digits Using Multiple/Parallel Methods, Third Golden West International Conference on Intelligent Systems, Las Vegas, June 6-9/94.

7. Suen, C.Y., et. al, Computer Recognition of Unconstrained Handwritten Numerals, Proc. IEEE, Vol. 80 No. 7, July 1992.

8. Zhang, Y.Y. and Suen, C.Y., A Fast Parallel Algorithm for Thinning Digital Patterns, CACM Vol. 27 No. 3, 1984. pp 236-239.

1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.94	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.05
0.00	0.02	0.92	0.01	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.03	0.01	0.00	0.90	0.02	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.01	0.01	0.94	0.01	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
0.00	0.08	0.00	0.00	0.00	0.00	0.00	0.88	0.00	0.04
0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00
0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.98

Figure 5 - Confusion matrix for the stroke tracing algorithm