# On-line Recognition of Handwritten Symbols

Gordon Wilfong, Frank Sinden, Laurence Ruedisueli

G. Wilfong is with Bell Laboratories of Lucent Technologies, Murray Hill, NJ 07974. Email: gtw@research.bell-labs.com

F. Sinden recently retired from AT&T Bell Laboratories, Murray Hill, NJ 07974

L. Ruedisueli is with AT&T Research, Murray Hill, NJ 07974. Email: lwr@research.att.com

**Abstract**

An on-line system for recognizing handwritten symbols from a user specified alphabet is described. When a symbol is subsequently written the system is required to recognize the symbol irrespective of the scale, orientation and position of the written symbol.

**Index Terms**

Pattern recognition, handwriting, on-line, symbol recognition, template matching

## I. Introduction

As a symbol is written on a digitizing tablet, the position of the pen tip and the force exerted by the pen tip is time-stamped and recorded at times determined by the sampling rate of the tablet. We consider the problem of recognizing such handwritten symbols based on the information provided by the tablet. This problem has received much attention [1] with the development of sufficiently high resolution tablets with high sampling rates [2].

The goal of the work described in this paper is to have an on-line system that is capable of recognizing a symbol from a user-defined alphabet irrespective of the size of the symbol as well as the orientation and position of the symbol on the tablet. Since the system is expected to recognize symbols from a user-defined alphabet, we concentrated on a user-dependent approach. Our notion of an alphabet of symbols is very general and not limited to any fixed collection such as alphanumerics or Kanji. For example, the alphabet under consideration may be a collection of cursively written words.

A number of previously reported methods for the problem of recognizing handwritten symbols have relied on recognizing features in the symbols, see for example [3]. Examples of such features are the presence or absence of ascenders or descenders, "closure" (e.g. the letters $o$ and $c$ differ since $o$ is 'closed" while $c$ is not) , or presence of dots (such as in lower case $j$'s and $i$'s). The basic idea is to build a binary decision tree where each internal node of the tree represents a feature and the edges descending from the node represent the presence or absence of the feature. Each leaf of the tree represents a symbol of the alphabet. For this method to work it is necessary to find a collection of features so that each symbol can be distinguished from the others by the presence or absence of the feature.

Such methods suffer from the problem that they are not easily extendible. That is, it may be difficult to add new symbols to the alphabet since new features may be necessary to distinguish the new symbol. In particular, any system based on this method is likely to be particular to a given alphabet whereas our goal is to develop a general system that works for any alphabet without any change to the software.

Another common method for symbol recognition is often referred to as curve matching [1]. Each symbol can be thought of as a curve (or collection of curves) in the plane and a curve is represented by a sampling of points along the curve. For each symbol in the alphabet, the system keeps a representative model curve. Curve matching techniques try to find the model curve that best fits the curve of the unknown input symbol. These methods generally use a normalization routine to minimize differences due to one or more of translation, rotation, dilation (scale), slant or other transformations. Often the normalization to remove rotational differences are either ignored, for example by insisting that symbols be written on a specified horizontal line, or are based on features of the alphabet being studied [4]. Normalization for translation is often based on the idea of translating the curve so that some simple geometric feature lies at the origin. Some examples of such features are an extreme point (for example, the leftmost point of the curve) or the center of the curve (the point whose $x-$ and $y-$coordinates are the average of the minimum and maximum $x$ values of the curve and the average of the minimum and maximum of the $y$ values of the curve respectively). The problem with these normalizations is that they are not robust with respect to rotations. One feature of a curve that can be used for translational normalization with the property that it is consistent with respect to rotations is the *center of mass of the curve* [5]. A variation on this normalization is used by the procedure discussed below.

The system described in this paper is based on curve matching. The normalization procedure and the definition of distance between two curves used in the recognition scheme result in a method that is capable of recognizing symbols written in any orientation or scale anywhere on the tablet.

## II. System Overview

The basic operation of the system is straightforward. At any point the user can be in training mode or recognition mode. To be in the training mode, the user indicates that he/she wishes to define a new symbol that the system is to be capable of recognizing. The user then writes the symbol on the tablet thus creating a model for that symbol. In the recognition mode, an unknown written symbol is compared to the models to find the model that is most similar to the unknown symbol. We discuss below the representation of the models and unknown symbols as well as the method for determining similarity.

## III. Terminology and Notation

We let $A = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ denote the current set of symbols that the user expects the system to be able to recognize. The set $A$ will be referred to as the *alphabet*.

In general, the image produced when the user writes a symbol will be called a *symbol instance*. In cases where we wish to be more specific we call a symbol instance a *model instance* if the user created the symbol instance in order to add a symbol to the alphabet or an *unknown instance* in the case where the user creates an instance to be recognized.

The notation $\alpha$ will generally be used to denote a symbol instance. The creation of an instance $\alpha$ determines an *instance sequence*

$$\eta_\alpha = < z_1(\alpha), z_2(\alpha), \ldots, z_n(\alpha) >$$

for some $n > 0$, which is the ordered sequence of pen point locations, $z_i(\alpha) = (x_i(\alpha), y_i(\alpha))$, sampled during the writing of $\alpha$. Notice that $n$, the number of sampled locations, will vary with each instance. The sampled pen locations $z_i$ will be called *data points* of $\alpha$. During the creation of an instance, the user may lift the pen above the tablet at various times. We define the *strokes* of an instance to be the maximal subsequences of data points between pen lifts. Therefore we associate with each instance $\alpha$, an ordered list of indices $E_\alpha$ where the elements of $E_\alpha$ are the indices of the last data point on each stroke. For example, consider an instance $\alpha$ of the letter "T" with instance sequence $< z_1(\alpha), z_2(\alpha), \ldots, z_n(\alpha) >$. Suppose $\alpha$ consisted of two strokes: one being the vertical segment $< z_1(\alpha), z_2(\alpha), \ldots, z_j(\alpha) >$

and the other the horizontal segment $< (z_{j+1}(\alpha), z_{j+2}(\alpha), \ldots, z_n(\alpha) >$. Then $E_\alpha$ is the ordered list $j$ followed by $n$.

## IV. Representations

To determine which symbol a given instance $\alpha$ was intended to be, we wish to compare a representation of the unknown instance with a representation of each of the model instances. The method used to represent an instance of a symbol will be discussed next. Throughout the discussion let instance $\alpha$ have instance sequence $\eta_\alpha =< z_1(\alpha), z_2(\alpha), \ldots, z_n(\alpha) >$ where $z_i(\alpha) = (x_i(\alpha), y_i(\alpha))$.

There have been a number of methods suggested for normalizing an instance with respect to translation in which the basic idea is to define a feature of the instance and translate the instance so that that feature is at some fixed position (usually the origin). One such feature sometimes used is the center of mass of the instance where the definition of center of mass of the instance ([6]) is given by the pair

$$\left( \frac{1}{n} \sum_{i=1}^{n} x_i(\alpha), \frac{1}{n} \sum_{i=1}^{n} y_i(\alpha) \right).$$

This definition however is not robust if the relative number of data points arising from a particular part of a symbol varies greatly between different instances. For example, suppose we have two instances of capital "T" where the first is written at a constant speed and the second is formed by writing the horizontal stroke very slowly (thus resulting in many data points) compared to the speed at which the vertical stroke is written. Then the center of mass of the first instance will be in roughly the intuitive "center" of the symbol (that is, about half way down the vertical stroke) whereas in the second case the center of mass will be almost at the point of intersection of the two strokes. Of course, there are various methods for re-sampling that would eliminate the kind of errors represented by this example. Instead we define the *centroid* of instance $\alpha$ to be $z_\alpha = (x_\alpha, y_\alpha)$ where

$$x_\alpha = \frac{\sum_{i=1}^{n-1} d_i(x_i + x_{i+1})}{2L}$$

$$y_\alpha = \frac{\sum_{i=1}^{n-1} d_i(y_i + y_{i+1})}{2L}$$

$$d_i = \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} \text{ and}$$

$$L = \sum_{i=1}^{n-1} d_i.$$

That is, $z_\alpha$ is the center of mass of the piecewise linear curve formed by interpolating line segments between consecutive data points of $\alpha$. To perform the position normalization we translate the data points of the instance so that the centroid of the instance moves to the origin.

The basic representation of an instance $\alpha$ in the recognition scheme is the *instance vector*

$$\mathbf{v}_\alpha = (\bar{x}_1(\alpha), \bar{y}_1(\alpha), \bar{x}_2(\alpha), \bar{y}_2(\alpha), \ldots, \bar{x}_n(\alpha), \bar{y}_n(\alpha))$$

where $\bar{x}_i(\alpha) = x_i(\alpha) - x_\alpha$ and $\bar{y}_i(\alpha) = y_i(\alpha) - y_\alpha$ for $1 \le i \le n$. A useful feature of this representation is that if $\alpha$ and $\beta$ are identical instances except that they have been created at different locations on the tablet and at different orientations then the representation $\mathbf{v}_\alpha$ is just a rotation about the origin of the other instance vector $\mathbf{v}_\beta$.

Thus to create an alphabet of symbols to be recognized the user creates a model instance $\alpha_k$ of symbol $\sigma_k$ which in turn determines an instance vector $\mathbf{v}_{\alpha_k}$. These instance vectors will be called *model instance vectors* and for ease of notation we shall denote them by $\mathbf{v}_k$ rather than $\mathbf{v}_{\alpha_k}$, $1 \le k \le m$. Similarly, creating an unknown instance $\alpha$ results in an *unknown instance vector* $\mathbf{v}_\alpha$.

Given some unknown instance vector $\mathbf{v}_\alpha$ and the model instance vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m$, we wish to find the model vector $\mathbf{v}_k$ "closest" to $\mathbf{v}_\alpha$. The measure of closeness used is described in the following section.

## V. Distance

We now derive a measure of dissimilarity or *distance* between an unknown instance vector and a model instance vector. Note that in the remainder of this paper, we assume that all model instance vectors and all unknown instance vectors are vectors in $\mathcal{R}^{2n}$. That is, we are assuming that every instance (including the instances used to form the model instance vectors) results in exactly the same number $n$ of sampled pen point locations. We discuss later how this assumption can be overcome.

As in the previous section, instance $\alpha$ is represented by the instance vector

$$\mathbf{v}_\alpha = (\bar{x}_1(\alpha), \bar{y}_1(\alpha), \ldots, \bar{x}_n(\alpha), \bar{y}_n(\alpha)).$$

Then the vector

$$\mathbf{u}_\alpha = (-\bar{y}_1(\alpha), \bar{x}_1(\alpha), \ldots, -\bar{y}_n(\alpha), \bar{x}_n(\alpha))$$

is the instance vector of the instance derived from $\alpha$ by a counterclockwise rotation of $\pi/2$ radians. Define $\mathbf{w}_\alpha : (-\pi, \pi] \times [0, \infty) \to \mathcal{R}^{2n}$ so that

$$\mathbf{w}_\alpha(\gamma_\alpha, r_\alpha) = r_\alpha(\mathbf{v}_\alpha \cos\gamma_\alpha + \mathbf{u}_\alpha \sin\gamma_\alpha). \tag{1}$$

Thus $\mathbf{w}_\alpha(\gamma_\alpha, r_\alpha)$ is the instance vector representing the instance derived from $\alpha$ by an $r_\alpha$-fold dilation and a counterclockwise rotation through $\gamma_\alpha$ radians. Let

$$S_\alpha = \{\mathbf{w}_\alpha(\gamma_\alpha, r_\alpha) : 0 \le r_\alpha < \infty \text{ and } -\pi < \gamma_\alpha \le \pi\}.$$

The instance vectors in $S_\alpha$ represent all instances derivable from $\alpha$ by translation, dilation and rotation. We will regard these instances as *equivalent.*

Similarly, for each $k$, $1 \le k \le m$, let $\mathbf{v}_k$ be the $k^{th}$ model vector (that is, the instance vector representing $\alpha_k$) and $\mathbf{u}_k$ the instance vector resulting from rotating $\alpha_k$ counterclockwise through $\pi/2$ radians. As before define

$$\mathbf{w}_k(\gamma_k, r_k) = r_k(\mathbf{v}_k \cos\gamma_k + \mathbf{u}_k \sin\gamma_k) \tag{2}$$

and $S_k$ is the set of all $\mathbf{w}_k(\gamma_k, r_k)$. We wish to define a measure of dissimilarity between the instances $\alpha$ and $\alpha_k$ that will be indifferent to translations, dilations and rotations of either one. We do this by defining a measure of distance between the *sets of equivalent vectors $S_\alpha$ and $S_k$* rather than between individual vectors.

If the various $\mathbf{w}_\alpha(\gamma_\alpha, r_\alpha)$'s and $\mathbf{w}_k(\gamma_k, r_k)$'s are interpreted as points in $2n$-space, then $S_\alpha$ and $S_k$ are subspaces of dimension two. That is to say, they are planes through the origin. To see this, consider the collection of instance vectors obtained by rotating the symbol instance without scaling. The resulting instance vectors lie on a 2-flat (that is, a plane) and define a circle about the origin in $\mathcal{R}^{2n}$. Changing the scale of the original instance vector and then rotating defines a circle of a different radius on the same plane. Thus allowing variations of scale and rotations results in instance vectors that span a plane. Unlike planes in 3-space, however, they generally have only one point in common (the origin) rather than a whole line. We define the "distance" between $S_\alpha$ and $S_k$ to be the *angle* between them.

## VI. Calculation of Distance

The angle between planes $S_\alpha$ and $S_k$ is easy to calculate.

The angle $\theta$ between two vectors $\mathbf{z}_1, \mathbf{z}_2$ in $2n$-space is derived from the definition of the scalar product:

$$\mathbf{z}_1 \cdot \mathbf{z}_2 = |\mathbf{z}_1||\mathbf{z}_2| \cos\theta \, .$$

Solving for $\theta$ yields

$$\theta = \cos^{-1}\left(\frac{\mathbf{z}_1 \cdot \mathbf{z}_2}{|\mathbf{z}_1||\mathbf{z}_2|}\right) \, .$$

For any two vectors this yields an angle between 0 and $\pi$ radians. The angle between planes $S_\alpha$ and $S_k$ is defined to be the smallest angle between any pair of vectors $\mathbf{w}_\alpha(\gamma_\alpha, r_\alpha)$ and $\mathbf{w}_k(\gamma_k, r_k)$ in $S_\alpha$ and $S_k$ respectively. Then if $\theta(\cdot,\cdot)$ is the angle between its arguments,

$$\theta(S_\alpha, S_k) = \min[\theta(\mathbf{w}_\alpha(\gamma_\alpha, r_\alpha), \mathbf{w}_k(\gamma_k, r_k))] \, ,$$

where the minimum is taken over all possible values of the parameters $r_\alpha$, $r_k$, $\gamma_\alpha$ and $\gamma_k$ (see Equations (1) and (2)). Using the scalar product formula,

$$\theta(S_\alpha, S_k) = \min\left[\cos^{-1}\left(\frac{\mathbf{w}_\alpha(\gamma_\alpha, r_\alpha) \cdot \mathbf{w}_k(\gamma_k, r_k)}{|\mathbf{w}_\alpha(\gamma_\alpha, r_\alpha)||\mathbf{w}_k(\gamma_k, r_k)|}\right)\right] \, . \tag{3}$$

Using Equations (1) and (2), we can write this in terms of the given vectors $\mathbf{v}_\alpha, \mathbf{u}_\alpha, \mathbf{v}_k, \mathbf{u}_k$. But before doing this, let us note some convenient properties of these vectors, easily verified by applying the definitions:

$$\mathbf{v}_\alpha \cdot \mathbf{u}_\alpha = \mathbf{v}_k \cdot \mathbf{u}_k = 0$$

$$\mathbf{v}_\alpha \cdot \mathbf{v}_k = \mathbf{u}_\alpha \cdot \mathbf{u}_k = p_k \tag{4}$$
$$\mathbf{v}_\alpha \cdot \mathbf{u}_k = -\mathbf{u}_\alpha \cdot \mathbf{v}_k = q_k$$

The new symbols $p_k$ and $q_k$ are hereby defined. Without loss of generality, we can assume that the $\mathbf{u}$'s and the $\mathbf{v}$'s are normalized to 1:

$$|\mathbf{v}_\alpha| = |\mathbf{u}_\alpha| = |\mathbf{v}_k| = |\mathbf{u}_k| = 1 \tag{5}$$

Substitution into Equation (3) of the expression for $\mathbf{w}_\alpha(\gamma_\alpha, r_\alpha)$ given in Equation (1) (as well as the corresponding expression for $\mathbf{w}_k(\gamma_k, r_k)$ given in Equation (2)) yields, with the help of Equations (4) and (5),

$$\theta(S_\alpha, S_k) = \min[\cos^{-1}(p_k \cos(\gamma_k - \gamma_\alpha) + q_k \sin(\gamma_k - \gamma_\alpha))]$$

where the minimum is now taken over the single parameter $\gamma_k - \gamma_\alpha$. Note that the dilation factors $r_\alpha, r_k$ have disappeared due to the normalization assumption. Since the arccosine is a decreasing function over the range $0, \pi$, this can be rewritten

$$\theta(S_\alpha, S_k) = \cos^{-1}(\max_\gamma [p_k \cos\gamma + q_k \sin\gamma]) \, .$$

(We have here replaced the internal variable $\gamma_k - \gamma_\alpha$ by the single symbol $\gamma$.) Recall that $p_k$ and $q_k$ are directly derived (by scalar products) from the given data. Finding the maximum by calculus, we obtain

$$\theta(S_\alpha, S_k) = \cos^{-1}\left((p_k^2 + q_k^2)^{1/2}\right) . \tag{6}$$

It is not difficult to show (by the Schwarz inequality and Equations (4) and (5)) that $p_k^2 + q_k^2 \leq 1$ so that $\theta$ is always real. In practice it is not really necessary to calculate the square root and the arccosine; it is sufficient to observe the closeness to unity of the expression

$$p_k^2 + q_k^2.$$

The definitions (Equation (4)) of $p_k$ and $q_k$ show that this expression can be written in various ways in terms of the given data. Among these are this one:

$$(\mathbf{v}_\alpha \cdot \mathbf{v}_k)^2 + (\mathbf{v}_\alpha \cdot \mathbf{u}_k)^2.$$

Finding the degree of dissimilarity between two instance vectors, then, comes down to the simple calculation of scalar products. It should be remembered, however, that the $\mathbf{u}$'s and the $\mathbf{v}$'s were assumed normalized, so in general their norms must be calculated also. From a statistical point of view, these scalar products of normalized vectors can be regarded as correlations.

Of course, it is not necessary to normalize the vectors and in that case the measure of dissimilarity is given by

$$\frac{(\mathbf{v}_\alpha \cdot \mathbf{v}_k)^2 + (\mathbf{v}_\alpha \cdot \mathbf{u}_k)^2}{(\mathbf{v}_\alpha \cdot \mathbf{v}_\alpha)(\mathbf{v}_k \cdot \mathbf{v}_k)}.$$

For each unknown instance $\alpha$ we are interested in the model instance that is most similar to $\alpha$ and so we only need consider the values

$$\frac{(\mathbf{v}_\alpha \cdot \mathbf{v}_k)^2 + (\mathbf{v}_\alpha \cdot \mathbf{u}_k)^2}{(\mathbf{v}_k \cdot \mathbf{v}_k)}$$

since $(\mathbf{v}_\alpha \cdot \mathbf{v}_\alpha)^2$ is the same for every $k$.

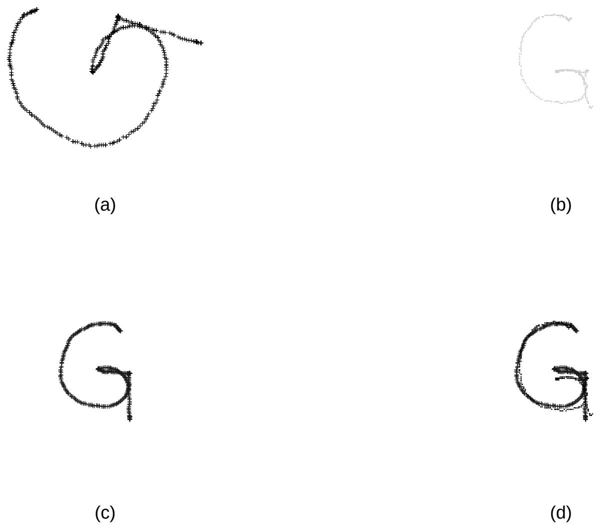


Fig. 1. Sample instance and closest model instance.

Figure 1 shows an example of the recognition results. In the figure, (a) shows the data points of an unknown instance $\alpha$, (b) shows the data points of the model instance $\alpha_k$ of the symbol found by the system to be closest to the unknown sample according to the distance $\theta(S_\alpha, S_k)$, (c) shows the optimal scaling and rotation of the unknown instance to best "match" the model instance and (d) is an overlay of the unknown instance and the model instance. It should be noted that the dilation and rotation are never explicitly performed during the recognition. Also, it is easy to extract the values of the dilation factor and rotation that would achieve the desired match. In particular, the dilation factor is given by

$$\frac{\sqrt{(\mathbf{v}_\alpha \cdot \mathbf{v}_k)^2 + (\mathbf{v}_\alpha \cdot \mathbf{u}_k)^2}}{(\mathbf{v}_\alpha \cdot \mathbf{v}_\alpha)}$$

and the counterclockwise rotation $\gamma$ is such that

$$\cos\gamma = \frac{(\mathbf{v}_\alpha \cdot \mathbf{v}_k)}{\sqrt{(\mathbf{v}_\alpha \cdot \mathbf{v}_k)^2 + (\mathbf{v}_\alpha \cdot \mathbf{u}_k)^2}} \quad \text{and}$$

$$\sin\gamma = \frac{(\mathbf{v}_\alpha \cdot \mathbf{u}_k)}{\sqrt{(\mathbf{v}_\alpha \cdot \mathbf{v}_k)^2 + (\mathbf{v}_\alpha \cdot \mathbf{u}_k)^2}}.$$

For alphabets in which there are distinct symbols that are rotations of one another (such as "C" and "U" for some people) or are the same except for size (for example, "c" versus "C") these values of dilation and rotation can be used to help discriminate between these otherwise confusable pairs.

It should be noted that the measure of distance described here has been derived in other ways (for example, see [7]) for other applications such as matching images.

## VII. Sequences with different lengths

As mentioned, the analysis and computation described above assumes that each model sequence has the same number of points as the unknown instance sequence. Of course, this is almost never the case. This problem is handled as follows. We begin by assuming that the model instances and the unknown instance have the same number of strokes. In the next section we show how to get around that assumption.

Suppose the unknown instance vector is

$$\mathbf{v}_\alpha = (x_1(\alpha), y_1(\alpha), \ldots, x_n(\alpha), y_n(\alpha))$$

and the $k^{\mathrm{th}}$ model instance vector is

$$\mathbf{v}_k = (x_1(\alpha_k), y_1(\alpha_k), \ldots, x_{n_k}(\alpha_k), y_{n_k}(\alpha_k)).$$

Note that the dimension, $2n_k$, of each model instance vector could be different from $2n$, the dimension of the unknown instance vector. We derive a new model instance vector $\bar{\mathbf{v}}_k$ based on $\mathbf{v}_k$ defined by $n$ rather than $n_k$ data points. The new vector is defined by deriving (from $\mathbf{v}_k$) a point $(\hat{x}_i(\alpha_k), \hat{y}_i(\alpha_k))$ for each data point $(x_i(\alpha), y_i(\alpha))$ of the unknown instance vector. The manner in which such points are computed is discussed next.

We think of each stroke of an instance $\alpha$ as a polygonal chain with the sampled pen positions marking the vertices of the chain. That is, it is a piecewise linear curve. As a stroke is created we store the sampled pen positions $z_i(\alpha)$ and also the arc length distance $a_i(\alpha)$ of $z_i(\alpha)$ along the piecewise linear curve. At the end of the stroke, we associate with each sampled point the normalized distance of that point along the chain. That is, if the total arc length of the stroke is $L$ then we store the value $r_i(\alpha) = a_i(\alpha)/L$, together with the data point $z_i(\alpha)$.

We now show how to derive a point $(\hat{z}_i(\alpha_k), \hat{y}_i(\alpha_k))$ from the model instance vector $\mathbf{v}_k$ to be associated with the point $z_i(\alpha)$ of the unknown instance vector. Suppose data point $z_i(\alpha)$ is on the $j^{th}$ stroke of the unknown instance $\alpha$. Then we find the point whose normalized distance along the $j^{th}$ stroke of the model instance is $r_i(\alpha)$. Such a point may actually be a data point of the model or more likely a point interpolated between two data points.

More formally, for each point $z_i(\alpha)$ of the $j^{th}$ stroke of the unknown instance $\alpha$, we define a corresponding point $\hat{z}_i(\alpha_k) = (\hat{x}_i(\alpha_k), \hat{y}_i(\alpha_k))$ derived from the $j^{th}$ stroke of the model instance $\alpha_k$ as follows. Suppose $z_i(\alpha)$ has normalized distance $r_i(\alpha)$ along the stroke. If there is some data point $z_s(\alpha_k)$ of the $j^{th}$ stroke with normalized distance $r_s(\alpha_k) = r_i(\alpha)$ then set $\hat{z}_i(\alpha_k) = z_s(\alpha_k)$. Otherwise, let $z_m(\alpha_k)$ and $z_{m+1}(\alpha_k)$ be the data points on the $j^{th}$ stroke of the model instance such that $r_m(\alpha_k) < r_i(\alpha) < r_{m+1}(\alpha_k)$. Then define $\hat{z}_i(\alpha_k) = (\hat{x}_i(\alpha_k), \hat{y}_i(\alpha_k)$ where

$$\hat{x}_i(\alpha_k) \quad = \quad x_m(\alpha_k) + \frac{(x_{m+1}(\alpha_k) - x_m(\alpha_k))(r_i(\alpha) - r_m(\alpha_k))}{(r_{m+1}(\alpha_k) - r_m(\alpha_k))} \quad \text{and}$$

$$\hat{y}_i(\alpha_k) \quad = \quad y_m(\alpha_k) + \frac{(y_{m+1}(\alpha_k) - y_m(\alpha_k))(r_i(\alpha) - r_m(\alpha_k))}{(r_{m+1}(\alpha_k) - r_m(\alpha_k))}.$$

It is simple to verify that the normalized distance of $\hat{z}_i(\alpha_k)$ along the $j^{th}$ stroke is $r_i(\alpha_i)$.

## VIII. Handling models and instances with unequal stroke counts

The definition of a stroke that we have been using, namely a maximal set of consecutive data points for which the pen is determined to be touching the tablet, is not particularly robust since making the binary decision as to whether the pen is up or down is not always clear cut. Other methods for segmenting an instance into strokes based on detecting "features" can also be unreliable. One brute-force method of getting around this problem would be to have a model for each variation in the number of strokes that a given symbol is likely to contain. In order to save the space required to store all such variations we try a different approach. Intuitively, our method is aimed at solving the problem that occurs when a user sometimes lifts the pen cleanly between "strokes" and other times drags the pen along the tablet part of the way or all of the way between the "strokes".

So far, for each instance $\alpha$ we have stored the instance vector $\mathbf{v}_\alpha$, the list of stroke end point indices $E_\alpha$ and the values $r_i(\alpha)$ (the normalized distance of the $i^{th}$ data point of $\alpha$ along the particular stroke the points belongs to.) The purpose of the values $r_i(\alpha)$ was to allow us to compare instances with unequal number of data points under the assumption that the instances had an equal number of strokes. Now we define another sequence of normalized distances to allow us to compare instances with (possibly) differing number of strokes.

Given an instance $\alpha$, we define a related instance $\alpha'$ that has one stroke but the same data points as $\alpha$. That is,

$$\mathbf{v}_{\alpha'} = \mathbf{v}_\alpha = (x_1(\alpha), y_1(\alpha), \ldots, x_n(\alpha), y_n(\alpha))$$

but $E_{\alpha'}$ has just one element, $n$. Let $z_i(\alpha') = z_i(\alpha) = (x_i(\alpha), y_i(\alpha))$. As in the previous section we define $r_i(\alpha')$ to be the normalized distance of $z_i(\alpha')$ along the one stroke of $\alpha'$. In fact, for the computation described below we will really only need to have the values $r_i(\alpha')$ for values $i$ in the list $E(\alpha)$. Therefore the only additional storage needed to represent $\alpha'$ beyond the information already stored to represent $\alpha$ is the storage for the values $r_i(\alpha')$ where $i$ is the index of the end point of a stroke of $\alpha$.

Given an unknown instance $\alpha$ with $s$ strokes and a model instance $\alpha_k$ with $t$ strokes we define the new one stroke instances $\alpha'$ and $\alpha'_k$ as above. Let $u(i)$ be the $i^{th}$ element of $E(\alpha)$, $1 \leq i \leq s$. That is, $z_{u(i)}$ is the last data point on the $i^{th}$ stroke of $\alpha$. Similarly, let $m(j)$ be the $j^{th}$ element of $E(\alpha_k)$, $1 \leq j \leq t$.

For each $u(i)$, let $N_1(u(i))$ be the $m(j)$ such that $r_{m(j)}(\alpha'_k)$ is closer to $r_{u(i)}(\alpha')$ than any other $m(v)$. Analogously define $N_2(m(j))$ to be the $u(i)$ such that $r_{u(i)}(\alpha')$ is nearer to $r_{m(j)}(\alpha'_k)$ than any other $u(v)$. Intuitively, $N_1(u(i))$ is supposed to be the index of the stroke end point of $\alpha_k$ whose normalized distance along the symbol is nearest to that of the $i^{th}$ stroke end point of $\alpha$.

We will now "partition" $\alpha'$ and $\alpha'_k$ into an equal number of strokes based on the following idea. If the endpoint of the $i^{th}$ stroke of $\alpha$ and the endpoint of the $j^{th}$ stroke of $\alpha_k$ are at similar normalized distances along $\alpha'$ and $\alpha'_k$ respectively then we assume that these endpoints are actually the endpoints of the same stroke of the symbol. Let $\hat{u}(1) < \hat{u}(2) < \ldots < \hat{u}(f)$ and $\hat{m}(1) < \hat{m}(2) < \ldots < \hat{m}(f)$ be the indices such that $N_1(\hat{u}(i)) = \hat{m}(i)$ and $N_2(\hat{m}(i)) = \hat{u}(i)$ and $|r_{\hat{u}(i)}(\alpha') - r_{\hat{m}(i)}(\alpha'_k)| < \epsilon$ for some appropriately small positive $\epsilon$. (Note that we used a value of $\epsilon = 0.1$ for our experiments.) Then we define a new unknown instance $\hat{\alpha}$ with the same data points as $\alpha$ but with $\hat{u}(1), \hat{u}(2), \ldots, \hat{u}(f)$ as indices of stroke endpoints and similarly a new model instance $\hat{\alpha}_k$ with the same data points as $\alpha_k$ but with $\hat{m}(1), \hat{m}(2), \ldots, \hat{m}(f)$ as indices of stroke endpoints. Then $\hat{\alpha}$ and $\hat{\alpha}_k$ have the same number of strokes and so they can be compared as indicated in the previous sections.

$N_1(u(1)) = m(1)$ and $N_2(m(1)) = u(1)$ $N_1(u(3)) = m(2)$ and $N_2(m(2)) = u(3)$

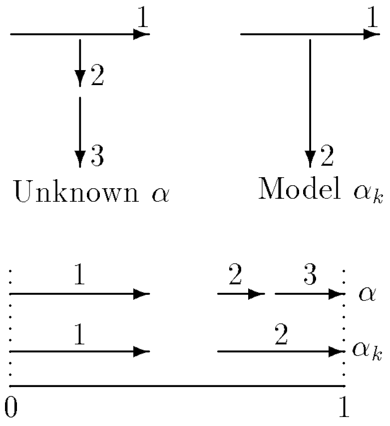


Fig. 2. Symbol T written with 3 strokes by mistake.

As an example, consider the case illustrated in Figure 2. In this example, the model instance for the letter "T" was written using two strokes, namely a horizontal left-to-right stroke followed by a pen lift and then a top-to-bottom vertical stroke. An unknown instance of the letter "T" is written exactly as the model instance was written except that due to the user accidentally lifting the pen during the vertical stroke the instance has three rather than two strokes. The figure shows graphically the relative distance of the pen down strokes when each instance is considered to be a one stroke instance (with pen up strokes interpolated linearly.) In the case shown note that $N_1(u(1)) = m(1)$ and $N_2(m(1)) = u(1)$ and also $N_1(u(3)) = m(2)$ and $N_2(m(2)) = u(3)$. However $N_1(u(2)) = m(2)$ but $N_2(m(2)) \neq u(2)$ and so the end of the second stroke of the unknown instance is ignored. That is, the second and third stroke of the instance are merged into a single stroke. Therefore we consider both instances to be instances with two strokes.

## IX. Implementation and Experiments

The recognition method described in this paper has been implemented in C [8] on various platforms including the NCR 3125 NotePad pen based computer running Microsoft® Windows™ for Pen Computing, and various UNIX® based machines using a graphics input tablet providing about 200 samples per second with a resolution of 0.1 mm [2]. As pointed out in Section VI, the amount of rotation needed to achieve the match implicitly defined by the dissimilarity measure used is easily computed. In the experiments mentioned below, we computed this value and used it to reject any possible matches where the amount of rotation exceeded a limit of about 49°.

We report the results of two experiments. In the first experiment, eight users were asked to write the capital letters A–Z and then to repeat. One set of the letters was used as the set of models and the other was run through the recognizer. The results were that there were 4 errors out of the $8 \times 26 = 208$ recognition tasks giving a 98% correct recognition rate. Of the eight users, five had 100% recognition rates, two others had one error and one had two errors.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *arc* | *copy* | *dotted* | *expand* | *in* | *lines* | *quit* | *size* |
| *arrow* | *cut* | *draw* | *fill* | *insert* | *move* | *rectangle* | *spline* |
| *break* | *dashed* | *ellipse* | *font* | *join* | *out* | *rotate* | *text* |
| *circle* | *delete* | *erase* | *group* | *line* | *polygon* | *scale* | *zoom* |

Fig. 3. List of cursively written words.

The second experiment used 32 cursively written words as the alphabet. The data used in this experiment was provided by the authors of [9]. The list of words is shown in Figure 3. Each of 19 users wrote each word 15 times. For each user, one sample of each of the 32 words was used as a model. The remaining 14 copies of each word were processed by the recognizer. The recognizer correctly recognized 95.5% of the words. Not all errors were analyzed in detail but a sampling seemed to indicate that about 15% of the errors were confusion of similar word pairs such as *out* vs. *cut* and *line* vs. *lines*. By far the largest cause for errors were due to differences in the way a user would cross *t*'s or dot *i*'s or *j*'s. That is, sometimes a user would vary the order in which these features (dots and crosses) were added or sometimes neglected to add them at all. Another source of errors was due to a user going back after writing a word to "touch-up" a bit of the word (for example, adding a small stroke to close up the top of a lower case *g*.) An additional trial was run with this same data but where five instances of each word were used as models (as opposed to one of each) and the remaining 10 processed by the recognizer. In this case the recognition rate was 98.3%.

## X. Discussion and Conclusions

An on-line method for recognizing handwritten symbols has been described. This method assumes the user always writes a given symbol in the same stroke order and direction. The experiments indicated that the majority of the errors with this system were due to additional or missing strokes in different instances of the same symbol (for example, in the cursive case, sometimes crossing *t*'s or dotting *i*'s and *j*'s and sometimes not) or due to different stroke order. Methods for handling these problems are currently under consideration.

Since our system was designed for arbitrary user-defined alphabets the system is generally thought of as a user-dependent system. However one could imagine "pre-loading" the system with all conceivable variations on the symbols of a particular alphabet to see how the system performs as a user-independent recognizer for that alphabet. Such experiments are currently being developed.

Choosing the best recognizer from the currently available systems is very much application dependent since it is generally the case that each recognizer has strengths and weaknesses. For a number of given alphabets there are various systems available.

For example, if the application is recognizing symbols written in boxes or recognizing handwritten documents then one should expect the writing to be fairly consistent in orientation and possibly scaling. In this case, a system such as that described in [10] may be appropriate. Tappert's method [10] is based on a dynamic programming procedure called "Elastic Matching" to find the best matching between the points of two instances. The approach has the property that it can handle some non-linear distortions that methods based on linear transformations (such as our system) cannot. This method can be useful if the two instances have been written in roughly the same orientation and size (or have been normalized.) Since this method (without normalization) depends on the "nearness" (in a Euclidean distance sense) between corresponding points of the two instances, it is not directly applicable when large differences in orientation and/or scaling are possible. Also, there is somewhat of a computational expense to pay for performing the dynamic programming procedure.

A method based on complete affine transformations (rather than the transformations restricted to translation, rotation and scaling as described in this paper) was described in [11]. This method finds, for each point $r_i$ of a model instance, an affine transformation to minimize the weighted sum of the distances between the resulting transformed model instance and the unknown instance. The weighting factors are such that points in the neighborhood of $r_i$ are much more influential in the resulting transformation than those points far away. Thus the transformations are referred to as Local Affine Transformations (LAT's). Each point of the model instance is transformed according to its corresponding LAT and this procedure can then be iterated. This method can be very powerful but can also be computationally somewhat expensive due to the fact that a transformation is computed for each point (rather than a single global transformation) and also the iterative nature of the full algorithm. Of course, one must be careful when using such a powerful transformation as the affine transformation to make sure that multiple symbols in the alphabet are not equivalent under the transformation.

Our system is chooses a compromise between allowing no transformations and permitting complete affine transformations. For the user-dependent applications with user-defined alphabets that we were interested in, this seemed to be a reasonable choice. For example, consider an application that calls for the user to write isolated symbols perhaps to activate commands. Then the writing the user performs will not necessarily be consistent in orientation or scale since there need be no lines to write on or boxes to write in and the orientation will be determined largely by the arbitrary orientation of the graphics tablet as placed on the user's desk. Of course, such a system will confuse symbols that differ by scale (for example, many upper versus lower case letters) or those that differ by rotation. Since we were interested in user-defined alphabets, it was helpful to restrict the allowable transformation to one that permitted the user to easily see that two symbols would be equivalent under the transformation and therefore two such symbols should not be included in their alphabets. This is fairly easy to do using only rotations and scaling but allowing full linear transformations makes the problem not so intuitive. Also, as discussed earlier our method provides (for no significant additional computation) a measure of the scaling and rotating necessary to perform the optimal transformation. These measures can aid in distinguishing such symbols.

In conclusion, the system described in this paper appears to be useful for applications with user-defined alphabets and where the size and orientation of symbols is likely to be variable. The system also has the advantage of being computationally efficient.

*Acknowledgments*

We wish to thank R. A. Boie and E. R. Wagner for the digitizing tablet and pen used for collecting data. We also thank J. Hu, M. K. Brown and W. Turin for allowing us to use their data set of cursively written words.

## References

[1] C. C. Tappert, C. Y. Suen, and T. Wakahara, "The state of the art in on-line handwriting recognition", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 8, pp. 787–808, 1990.

[2] G. E. Blonder, R. A. Boie, L. W. Ruedisueli, E. R. Wagner, and Y. H. Wong, "Capacitive moment sensing for electronic paper", AT&T Bell Laboratories Internal Document, 1991.

[3] L. S. Frishkopf and L. D. Harmon, "Machine reading of cursive script", in *Information Theory (4th London Symposium)*, C. Cherry, Ed. 1961, pp. 300–316, Butterworths.

[4] M. K. Brown and S. Ganapathy, "Preprocessing techniques for cursive script word recognition", *Pattern Recognition*, vol. 16, no. 5, pp. 447–458, 1983.

[5] K. Odaka, H. Arakawa, and I. Masuda, "On-line recognition of handwritten characters by approximating each stroke with several points", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 12, no. 6, pp. 898–903, 1982.

[6] T. Fujisaki, T. Chefalas, J. Kim, C. Tappert, and C. Wolf, "On-line run-on character recognizer: Design and performance", *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 5, no. 1&2, pp. 123–137, 1991.

[7] K. Zikan, *The Theory and Applications of Algebraic Metric Spaces*, PhD thesis, Stanford University, 1989.

[8] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, Englewoods Cliffs,NJ, 1978.

[9] J. Hu, M. K. Brown, and W. Turin, "Handwriting recognition with hidden markov models and grammatical constraints", AT&T Bell Laboratories Internal Document, 1993.

[10] C. C. Tappert, "Cursive script recognition by elastic matching", *IBM Journal of Research and Development*, vol. 26, pp. 765–771, 1982.

[11] T. Wakahara, “On-line cursive script recognition using local affine transformation”, in *Proc. 9th Int. Conf. Pattern Recognition*, 1988, pp. 1133–1137.