



An image-based, trainable symbol recognizer for hand-drawn sketches

Levent Burak Kara^{a,*}, Thomas F. Stahovich^b

^aMechanical Engineering Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA

^bMechanical Engineering Department, University of California, Riverside, Riverside, CA 92521, USA

Abstract

We describe a trainable, hand-drawn symbol recognizer based on a multi-layer recognition scheme. Symbols are internally represented as binary templates. An ensemble of four different classifiers compares and ranks definition symbols according to their similarity to the unknown symbol. The scores of the individual classifiers are aggregated to produce a combined score for each definition. The definition with the best combined score is assigned to the unknown symbol. All four classifiers use template-matching techniques to compute similarity (and dissimilarity) between symbols. Ordinarily, template-matching is sensitive to rotation, and existing solutions for rotation invariance are too expensive for interactive performance. We have developed a fast technique that uses a polar coordinate representation to achieve rotational invariance. This technique is applied prior to the multi-classifier recognition step to determine the best alignment of the unknown with each definition. One advantage of this technique is that it filters out the bulk of unlikely definitions, thereby reducing the number of definitions the multi-classifier recognition step must consider.

© 2005 Elsevier Ltd. All rights reserved.

Keywords: Pen computing; Sketch understanding; Symbol recognition; Pattern recognition; Hausdorff distance; Tanimoto coefficient; Yule coefficient; Polar transform

1. Introduction

It is common for engineers, architects, and other designers to spend a considerable amount of time laying down their initial concepts using pencil and paper. Typically, it is only after the main ideas have sufficiently matured, that all of that work is transformed into electronic media in the form of technical drawings, flow charts and mathematical models. This obvious redundancy and inefficiency has propelled the idea of sketch-

based user interfaces as a means of achieving more natural human-computer interaction.

Early attempts to create such systems were often limited by insufficient technology. New insights into human perception, as well as advances in pattern recognition, machine intelligence, computer graphics, and hardware technology, have now made it feasible to create usable systems. In fact, in many of today's mainstream computing devices, such as tablet PC's, electronic whiteboards, and personal digital assistants (PDA's), the pen is emerging as a standard tool for interaction.

Many of these new computing devices now come equipped with robust handwriting recognition utilities. However, an important aspect of sketch-based computer interaction that remains largely unsolved is the robust

*Corresponding author. Tel.: +1 412 268 8880; fax: +1 412 268 3348.

E-mail addresses: lkara@andrew.cmu.edu (L.B. Kara), stahov@engr.ucr.edu (T.F. Stahovich).

recognition of *graphical* input, such as geometric shapes, engineering symbols and glyphs. Researchers are beginning to make progress in handling such forms of input. Nevertheless, even the latest experimental systems are typically limited to basic shapes such as rectangles, triangles and circles. Furthermore, the few experimental systems that do provide shape recognition are often too restrictive because their recognizers are either special-purpose, hard-coded systems, or they require substantial amounts of training data, which makes them difficult to extend to domains with novel scripts and glyphs.

The work presented here is focused on the development of a trainable symbol recognizer that provides (1) interactive performance, (2) easy extensibility to new shapes, and (3) fast training capabilities. We have developed a novel symbol recognizer that is capable of learning new definitions from single prototype examples. Additionally, because our approach is based on a down-sampled bitmap representation (binary templates), it is particularly useful for “sketchy” input, such as drawings with heavy over stroking and erasing. Likewise, our approach employs a polar coordinate representation, which allows us to achieve rotational invariance in a computationally efficient fashion.

2. Overview

This section gives a brief overview of the main characteristics of our recognizer followed, by a description of its underlying architecture.

2.1. Template representation and its benefits in sketch understanding

Our recognizer uses an image-based recognition approach. Input symbols are internally described as down-sampled bitmap images which we call “templates.” This representation has a number of desirable characteristics. First, pen stroke segmentation—the process of decomposing the pen strokes into constituent primitives such as lines and curves—is eliminated entirely.¹ Many of the traditional recognition approaches, such as graph-based² and feature-based³

¹Parsing, the task of locating the individual symbols in the sketch, is required. That is the focus of our other ongoing research [1–3].

²In graph-based methods the basic geometric primitives obtained after segmentation are assembled into a graph structure that encodes the intrinsic attributes of the primitives and their spatial relationships.

³In feature-based methods, various aspects of the patterns are quantified and encoded in a feature space that helps distinguish between different patterns. In sketch recognition, the features are often geometric, and commonly involve the line and arc primitives obtained from segmentation.

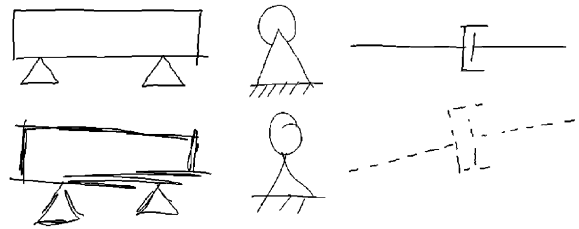


Fig. 1. Examples of symbols correctly recognized by our system (at the time of the test, the database contained 104 definition symbols). The top row shows symbols used in training, and the bottom row shows correctly recognized test symbols. Our approach can handle moderate over stroking, missing and extra pen strokes, different line styles, and variations in angular orientation.

methods, rely heavily on the segmentation process, making them vulnerable to segmentation errors. Second, our approach is suitable for recognizing “sketchy” symbols such as those shown in Fig. 1. Lastly, symbols drawn with multiple strokes or varying drawing orders do not pose difficulty. Many of the existing recognition approaches have either relied on single-stroke methods in which an entire symbol must be drawn in a single stroke [4,5] or constant drawing order methods in which two similarly shaped patterns are considered different unless the pen strokes leading to those shapes follow the same sequence [6].

2.2. Learning from a single example

There has been a large body of work concerning character and digit recognition. Most systems have traditionally been built on statistical learning methods that require large amounts of training data. For instance, LeCun et al.’s [7] neural network recognizer for handwritten digits, one of the best in its class, uses a total of 60,000 patterns for training purposes. However, due to the need for large training sets, these systems are not easily extensible to new applications with novel symbols and shapes.

By contrast, our focus is to develop a portable ink recognition utility that can be used in a multitude of different applications, with a wide variety of graphical elements. Therefore, one of the principle goals of this project was to enable users to create, extend and update their own library of symbols without the need for extensive training. To this end, we designed our system to work from single prototype examples. For training, the user creates a new symbol definition by drawing a single example. With this approach, users can seamlessly train new symbols, and remove or overwrite existing ones on the fly, without having to depart the main application. An additional advantage is that, unlike many statistical and neural network approaches, the

existing symbols do not need to be retrained or adjusted upon the introduction of a new symbol.

While the primary advantage of our recognizer is its ability to work from single training examples, it is easily extended to work from multiple training examples. For this, the user simply provides multiple, different definition templates to the database for each type of symbol. During our user studies, we have observed this approach to noticeably improve the recognition accuracy at an expense of a minor increase in recognition times.

2.3. Multiple classifiers

During our studies we experimented with a variety of classification methods and found that no single method was adequate for hand-drawn shapes. However, recognition accuracy increased dramatically when classifiers were used in combination. Inspired by this observation, we designed a recognition scheme comprised of four classifiers. Our tests indicated that the combined scheme usually outperforms individual classifiers, and is always better than the worst performing classifier. In fact, we have frequently encountered cases in which the combined scheme produced the right result even though none of the classifiers ranked the true class at the top. These findings are consistent with a large body of evidence that supports the idea of multiple classifiers for recognition [8].

2.4. Achieving rotation invariance efficiently

Template matching is ordinarily sensitive to rotations. Therefore, patterns must be brought to the same orientation before template matching is applied. In many cases, this is achieved by incrementally rotating one pattern relative to the other until the best correspondence is obtained. However, this approach is too expensive for real-time applications due to the costly rotation operation. We developed a technique, based on polar coordinates, to greatly expedite this process. The technique is based on the fact that rotations in screen coordinates become translations in polar coordinates. Hence, finding the optimal rotational alignment in screen coordinates reduces to determining the shift between patterns in polar coordinates. As we shall describe later, this technique is conceptually similar to the cross-correlation operation in signal processing.

2.5. Two-step recognition

We use the results of the polar analysis not only to determine the best alignment angles but also as a tool to filter out unlikely matches before recognition. We have found that the similarity metric employed in polar coordinates gives a reasonable estimate of the match in screen coordinates. Specifically, we found that although

the analysis in polar coordinates may sometimes mistake two dissimilar patterns as being similar, it almost never misses a true match when there is one. Taking advantage of this feature, we designed a two-phase recognition scheme that first involves an elimination of the bulk of the unlikely matches in polar coordinates, followed by a detailed evaluation of the reduced set of candidates in screen coordinates.

2.6. System architecture

The recognition architecture consists of four sequential layers as shown in Fig. 2. The first step is preprocessing, where the input symbols are cropped, size normalized and quantized into templates. If the system is in training mode, the template becomes a

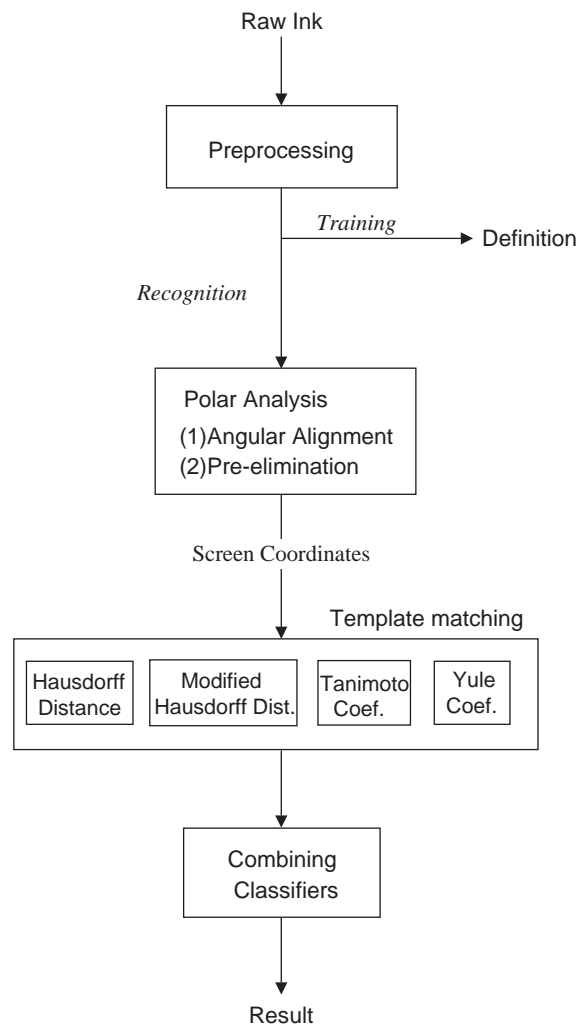


Fig. 2. Recognition architecture.

definition and is added to the database of existing definitions. If the system is in recognition mode, the template is passed to the next stage where it is matched against the definitions.

In the first step of recognition, the unknown symbol is transformed into a polar coordinate representation, which allows the program to efficiently determine which orientation of the unknown best matches a given definition. During this process, definitions that are found to be markedly dissimilar to the unknown are pruned out and the remaining ones are kept for further analysis. In the second step, recognition switches to screen coordinates where the surviving definitions are analyzed in more detail using an ensemble of four different classifiers. Each classifier outputs a list of definitions ranked according to their similarity to the unknown. In the final step of recognition, results of the individual classifiers are pooled together to produce the recognizer's final decision.

As shown in Fig. 2, the analysis in the polar coordinates precedes the analysis in the screen coordinates. However, for the sake of presentation, we have found it useful to begin the discussion with our template-representation and the four template matching techniques, since some of those concepts are necessary to set the context for the analysis in the polar coordinates. Hence, in the next few sections we shall assume that symbols are already brought into the correct orientation using the polar analysis, and we will defer the details of that until later. The remainder of the paper is organized as follows: In the next section, we explain the preprocessing of the raw data and the template representation. In Section 4, we describe the multi-classifier recognition scheme and the similarity measures used in this step. Section 5 details our method for combining classifiers. Section 6 explains our method for achieving rotation invariance using the polar transfor-

mation. Section 7 explains how the polar transformation is used as a pre-recognizer. Section 8 presents the results from an experimental study, followed by the related work and conclusions.

3. Preprocessing and representation

Symbols are drawn using a 9" × 12" WACOM Intuos2 digitizing tablet and a cordless stylus. Data points are collected as time sequenced (x, y) coordinates sampled along the stylus' trajectory. There is no restriction on the number of strokes, and symbols can be drawn anywhere on the tablet, in any size and orientation.

Input patterns are internally represented as binary bitmap images that consist of the (x, y) coordinates collected from the digitizing tablet. However, in this form, the input image usually contains too many data points, which can hinder recognition performance. To facilitate recognition, we frame and down sample the initial image into a 48×48 square grid producing a rasterized image we call a "template." This quantization significantly reduces the amount of data to consider while preserving the patterns' distinguishing characteristics. (We have experimented with other template sizes, and have found 48×48 to be a good compromise between accuracy and efficiency.) To frame the image, we first construct a bounding box aligned with the screen axes. We then expand the shortest dimension of the bounding box, without changing the location of the box's center, to produce a square. The result is that the symbol appears centered in a square frame, but does not necessarily fill the entire frame. This representation preserves the original aspect ratio so that one can distinguish between, say, a circle and an ellipse. Fig. 3 shows examples of templates.

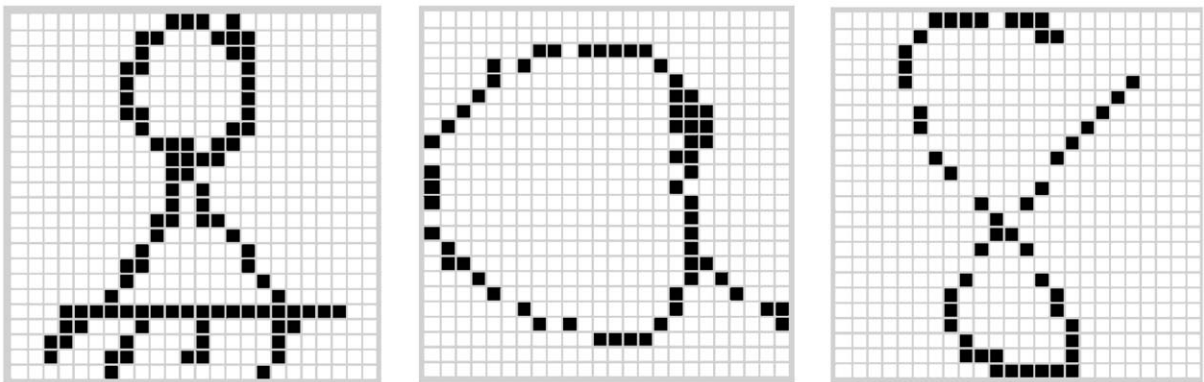


Fig. 3. Examples of symbol templates. Left: a mechanical pivot; middle: "a"; right: "8". The templates are shown on 24×24 grids to better illustrate the quantization.

4. Template matching with multiple classifiers

Template matching, in its simplest form, can be described as the process of superimposing two digital images and applying a measure of similarity. Although individual techniques differ in the way they define similarity, one property common to all template-matching methods is that they are inherently “featureless”—the template itself constitutes the input to the recognizer. Hence, there is no need, for example, to determine the set of line and arc primitives that best approximates the pattern, or the geometric relationships between them such as parallelism, intersection or containment. In their work on face recognition, Brunelli and Poggio [9] demonstrate the advantage of this simplicity over feature-based methods. Besides eliminating the need for feature extraction, template methods also provide great simplicity and flexibility in training new patterns. In our case, for instance, a new symbol can be easily added to the database by simply drawing one example of it.

While most template-based recognition systems are traditionally designed around a single similarity measure, we use four different methods to enhance recognition accuracy. The first two are based on the Hausdorff distance, which measures the dissimilarity between two point sets. Hausdorff-based methods have been successfully applied to object detection in complex scenes [10]. However, most of the applications have involved detection or recognition of “rigid” objects, such as those in photographic images or machine generated text, and only a few researchers have recently considered the use of the Hausdorff distance for hand-drawn pattern recognition: Cheung et al. [11] have used it for character recognition and Miller et al. [12] for digit recognition. In this work, we extend these studies to rotation invariant graphical symbol recognition. In particular, as described in the following sections, we apply the Hausdorff distance both in screen coordinates and in polar coordinates. Also, in Section 6, we introduce a *weighted* Hausdorff distance method that enables different parts of an image to be emphasized differently during matching, according to a measure of confidence based on prior information about the image.

Our other two recognition methods are based on the Tanimoto and Yule coefficients. Unlike the Hausdorff methods, these methods measure the similarity between patterns and output their results in the form of correlation coefficients. The Tanimoto coefficient is extensively used in chemical informatics such as drug testing, where the goal is to identify an unknown molecular structure by matching it against known structures in a database [13]. The Yule coefficient has been proposed as a robust measure for binary template matching [14]. To the best of our knowledge, the Tanimoto and Yule measures have not previously been applied to handwritten pattern recognition.

In the following paragraphs we detail these four classification methods and explain the modifications we used to better suit them to hand-drawn symbol recognition.

4.1. Hausdorff distance

The Hausdorff distance between two point sets A and B is defined as

$$H(A, B) = \max(h(A, B), h(B, A)),$$

where

$$h(A, B) = \max_{a \in A} \left(\min_{b \in B} \|a - b\| \right)$$

$\|a - b\|$ represents a measure of distance (e.g., the Euclidean distance) between two points a and b . $h(A, B)$ is referred to as the directed Hausdorff distance from A to B and corresponds to the maximum of all the distances one can measure from each point in A to the closest point in B . The intuitive idea is that if $h(A, B) = d$, then every point in set A is at most distance d away from some point in B . $h(B, A)$ is the directed distance from B to A and is computed in a similar way. Note that in general $h(A, B) \neq h(B, A)$. The Hausdorff distance is defined as the maximum of the two directed distances.

In its original form, the Hausdorff distance is too sensitive to outliers. The *partial* Hausdorff distance proposed by Rucklidge [10] eliminates this problem by ranking the points in A according to their distances to points in B in descending order, and assigning the distance of the k th ranked point as $h^k(A, B)$. The partial Hausdorff distance from A to B is thus given by

$$h^k(A, B) = \max_{a \in A} \left(\min_{b \in B} \|a - b\| \right)^k$$

The partial Hausdorff distance, in effect, softens the distance measure by discarding points that are maximally far away from the counterpart point set. The results reported in the following sections are based on a rank of 6%, i.e., in the calculation of the directed distances, the most distant 6% of the points are ignored. We determined this cutoff value empirically based on the user experience with our system.

Whether it is based on the maximum or the k th ranked directed distance, calculation of $h(A, B)$ involves computing, for each point in A , the distance to the nearest point in B . This process can be greatly expedited by using what is called the distance transform. The main idea is to pre-compute all necessary distances *only once* during the training phase, allowing any distance of interest to be obtained via simple indexing during recognition. In our system, we have found the distance transform to accelerate the computation of the Hausdorff distance by a few orders of magnitude. A brief explanation of the distance transform and its utility in

template matching can be found in Section 4.5, following the discussions of the similarity measures.

4.2. Modified Hausdorff distance

Modified Hausdorff distance (MHD) [15] replaces the *max* operator in the directed distance calculation by the average of the distances:

$$h_{mod}(A, B) = \frac{1}{N_a} \sum_{a \in A} \min_{b \in B} \|a - b\|.$$

where N_a is the number of points in A . The MHD is then defined as the maximum of the two directed average distances:

$$MHD(A, B) = \max(h_{mod}(A, B), h_{mod}(B, A)).$$

Although $h_{mod}(A, B)$ may appear similar to $h^k(A, B)$ with $k = 50\%$, the difference is that the former corresponds to the mean directed distance while the latter corresponds to the median. Dubuisson and Jain argue that for object matching purposes, the average directed distance is more reliable than the partial directed distance mainly because as the noise level increases, the former degrades gracefully whereas the latter exhibits a pass/no-pass behavior.

4.3. Tanimoto coefficient

The Tanimoto coefficient between two binary images A and B is defined as:

$$T(A, B) = \frac{n_{ab}}{n_a + n_b - n_{ab}},$$

where n_a is the total number of black pixels in A , n_b is the total number of black pixels in B , and n_{ab} is the number of overlapping black pixels.

Intuitively, $T(A, B)$ specifies the number of matching points in A and B , normalized by the union of the two point sets. By definition, $T(A, B)$ yields values between 1.0 (maximum similarity) and 0.0 (minimum similarity). In the form given above, the similarity between two images is based solely on the matching black points. However, for images that contain mostly black pixels, the discrimination power of $T(A, B)$ may vanish. In such situations, coincidence of white pixels can be used as a measure of similarity:

$$T^C(A, B) = \frac{n_{00}}{n_a + n_b - 2n_{ab} + n_{00}},$$

where n_{00} is the number of matching white pixels. The denominator is the number of pixels that are white in at least one of the images. $T^C(A, B)$ is called the Tanimoto coefficient complement. It represents the number of matching white pixels normalized by the union of the white pixels from the two images. The two expressions can be combined to form the Tanimoto similarity coefficient [13]:

$$T_{sc}(A, B) = \alpha T(A, B) + (1 - \alpha) T^C(A, B),$$

where α is a weighting factor between 0.0 and 1.0. Ideally, if the number of black pixels in an image is small compared to the number of white pixels, the similarity decision should be based on matching black pixels. In this case, $T(A, B)$ should be emphasized by means of a large α . In the converse case, similarity should be based on matching white pixels, which means $T^C(A, B)$ should be emphasized by means of a small α .

This effect can be achieved by linking α to the relative number of black pixels as follows:

$$\alpha = 0.75 - 0.25 \left(\frac{n_a + n_b}{2n} \right),$$

where n is the image size in pixels. The term in parentheses is the total number of black pixels divided by the total number of pixels in the two images. The form of this relationship is adapted from [13] such that α is small when the number of black pixels is high and vice versa. We selected the two constants in the equation so that α is generally high, in the range [0.5, 0.75] to be precise. This bias favors $T(A, B)$ over $T^C(A, B)$. The choice is justified by the fact that hand-drawn symbols usually consist of thin lines (unless excessive over-tracing is done) producing rasterized images that contain fewer black pixels than white. Hence, for our applications, the Tanimoto coefficient should be controlled more by $T(A, B)$ than by $T^C(A, B)$.

Similarity measures that are based exclusively on the number of overlapping pixels, such as the Tanimoto coefficient, often suffer from slight misalignments of the rasterized images. We have found this problem to be particularly severe for hand-drawn patterns where rasterized images of ostensibly similar shapes are almost always disparate, either due to differences in shape, or more subtly, due to differences in drawing dynamics. The latter commonly occurs as a result of irregular drawing speed, often manifesting itself as unevenly sampled digital ink. Hence, for two shapes drawn at different speeds, the resulting rasterized images will likely exhibit differences. In order to absorb such variations during matching, we use a thresholded matching criterion that considers two pixels to be overlapping if they are separated by a distance less than 1/15th of the image's diagonal length. For a 48×48 image grid, this translates into 4.5 pixels, i.e., two points are considered to be overlapping if the distance between them is less than 4.5 pixels.

4.4. Yule coefficient

The Yule coefficient, also known as the coefficient of colligation, is defined as:

$$Y(A, B) = \frac{n_{ab}n_{00} - (n_a - n_{ab})(n_b - n_{ab})}{n_{ab}n_{00} + (n_a - n_{ab})(n_b - n_{ab})},$$

where the term $(n_a - n_{ab})$ corresponds to the number of black pixels in A that do not have a match in B .

Similarly, $(n_b - n_{ab})$ is the number of black pixels in B that do not have a match in A .

$Y(A, B)$ produces values between 1.0 (maximum similarity) and -1.0 (minimum similarity). Unlike the original form of the Tanimoto coefficient, the Yule coefficient simultaneously accounts for the matching black and white pixels via the terms n_{ab} and n_{00} . However, like the Tanimoto coefficient, it is sensitive to slight misalignments between patterns. We therefore employ a thresholded matching criterion similar to the one we use with the Tanimoto method.

Tubbs [14] originally employed this measure for generic, noise-free binary-template-matching problems. By using a threshold, we have made the technique useful when there is considerable noise, as is the case with hand-drawn shapes.

4.5. Distance transform

A distance transform can be simply described as a nearest neighbor function. It is a morphological operation that converts a binary bitmap image into an image in which each pixel encodes its distance (we use the Euclidean distance) to the nearest black pixel in the same image. The resulting image is called a distance map. During matching, distance maps serve as look-up tables for the closest distances. In the Hausdorff distance, for instance, the directed distance $h(A, B)$ is found by superimposing template A on the distance transform map of B . Minimum distances are simply read from B 's distance map using points in A as query indices. $h(A, B)$ then becomes the maximum (or the k th max.) of the queried distances. $h(B, A)$ is computed in a similar way except points in B are used as indices to query A 's distance map.

In our system, distance maps are constructed along with the symbol templates during preprocessing. Algorithms for the computation of distance transforms can be broadly classified as exact methods and approximate methods. We experimented with both types and found the exact methods to be more suitable for our purposes despite some loss in efficiency. Nevertheless, because the distance transform of each definition symbol is computed only once, immediately after the user presents the symbol, the difference in performance is not noticeable. Furthermore, the only distance transform that needs to be calculated during recognition is the unknown's, since the transforms of the definitions are already available.

5. Combining classifiers

Our recognizer compares the unknown symbol to each of the definitions using the four classifiers explained above. The next step in recognition is to identify the true

class of the unknown by synthesizing the results of the component classifiers. However, the outputs of the classifiers are not compatible in their original forms because: (1) The first two classifiers are measures of *dissimilarity* while the last two are measures of *similarity*, and (2) the classifiers have dissimilar ranges.

To establish a congruent ranking scheme, we first transform the Tanimoto and Yule similarity coefficients into distance measures by reversing (negating) their values. This process brings the Tanimoto and Yule coefficients in parallel with the Hausdorff measures in the sense that the numerical scores of all classifiers now increase with increasing dissimilarity. Next, to eliminate the range differences among classifiers, we normalize the values of all four classifiers to the range 0–1 using a linear transformation function. For each classifier, the transformation maps the distance scores to the range [0,1] while preserving the relative order established by that classifier. Finally, having standardized the outputs of the four classifiers, we combine the results using a method similar to the sum rule introduced by Kittler et al. [16]. For each definition symbol, we compute a combined normalized distance by summing the normalized distances obtained from the constituent classifiers. The unknown pattern is then assigned to the class having the minimum combined normalized distance. Kittler's experimental and theoretical analysis prove the sum rule to be superior to other rules such as the product, max, min and median rules. Their findings are particularly important as their rules are generalizations of many existing methods.

6. Handling rotations

Template-matching techniques are sensitive to orientation. Therefore, for rotation invariant recognition, it is necessary to first rotate the patterns into the same orientation. Often this is accomplished by incrementally rotating one pattern relative to the other until the best alignment is achieved. However, this is overwhelmingly expensive for real-time applications due to the costly rotation operation. We have developed an efficient technique, based on the polar coordinate transformation, to greatly facilitate this process. The main principle is that rotations in Cartesian coordinates become translations in polar coordinates. Hence, by identifying the linear offset between two patterns in polar coordinates, we can determine the angle by which the patterns differ in the x - y plane.⁴ This approach forms the basis of our solution.

⁴This process is conceptually analogous to the cross-correlation operation used in signal processing. Cross-correlation determines if a signal resembles a time-shifted version of another one. It does so by incrementally sliding one signal along

Others have used a polar transform for rotation invariant shape matching. Most of that work, however, has been focused on rigid objects and gray-level images [17,18]. In our work, we have extended the polar transform to the domain of imprecise, hand-drawn patterns.

6.1. Polar transform

The polar coordinates of a point in the x - y plane are given by the point's radial distance, r , from the origin and the angle, θ , between that radius and the x -axis. The well-known relations are

$$r = \sqrt{(x - x_o)^2 + (y - y_o)^2}$$

and

$$\theta = \tan^{-1} \left(\frac{y - y_o}{x - x_o} \right)$$

where (x_o, y_o) is the origin.

A symbol originally drawn in the screen coordinates (x - y plane) is transformed into polar coordinates by applying this formulae to each of the points. Fig. 4a illustrates a typical transformation. As shown in Fig. 4b, when a pattern is rotated in the x - y plane, the corresponding polar image slides parallel to the θ -axis by the same angular displacement.

In the form given above, the polar transformation is sensitive to size. When a pattern is scaled in the x - y plane, the corresponding polar image stretches along the r -axis. To eliminate such variance, we first normalize the r -axis using the "ink length" of the symbol. We define ink length as the total distance the pen tip travels on the writing surface. The main reason for using the ink length as opposed to, say, the diagonal or perimeter of the bounding box, is that ink length is invariant to the orientation of the pattern while the bounding box properties are not. With this normalization, the values along the r -axis correspond to non-dimensional scale factors. No adjustment to the θ -axis is necessary as uniform scaling does not affect angular positions.

For polar transforms, the choice of the origin has a significant impact on the resulting image. Although a consistent selection of the origin (e.g., always the top-left corner of the drawing tablet) would theoretically seem appropriate, there are two important practical factors to consider. First, it is desirable to set the origin inside the image, preferably close to the image center, so that the θ range can be utilized to its full extent. If the origin is far

away from the image, for instance at one of the screen corners, the polar image will subtend only a narrow angular window, compressing many important details. Second, identical shapes should have identical origins so that their polar transforms are identical. A seemingly suitable, and in fact common choice for the origin is thus the centroid of the original pixels. With such a formulation, because the centroid is simply the average of the *sampled* points, the centroid has a tendency to drift towards regions containing dense pixel clusters. In our domain, however, variation in the pen speed often causes large variations in the pixel density. For example, the significant reduction of the pen speed at the beginnings and endings of strokes causes pixels to be denser in these regions compared to the rest of the stroke. We have found this phenomena to abnormally and unpredictably alter the centroid location, causing polar transforms of similar shapes to be different.

One way to prevent this is to resample the input data to obtain a sequence of points uniformly spaced in arc length as opposed to time [8]. These methods are mostly based on interpolation algorithms. We achieve a similar effect in a more direct fashion by computing the weighted centroid of the *line segments* that join pairs of consecutive points. Each segment is assigned a weight proportional to its length, and the new centroid becomes the mean of the weighted segments:

$$x_c = \frac{\sum_{i=1}^S x_i l_i}{\sum_{i=1}^S l_i} \quad \text{and} \quad y_c = \frac{\sum_{i=1}^S y_i l_i}{\sum_{i=1}^S l_i},$$

where S is the total number of segments and l_i is the length of the i th segment. Each segment is treated as a point-mass concentrated at the segment's center, denoted (x_i, y_i) . This approach attenuates the effect of short segments and therefore prevents dense pixel clusters from shifting the centroid. As a result, the centroid location becomes more stable over different examples of a pattern.

Without loss of generality, we shall take advantage of the 2π -periodicity of the θ -axis and position all polar images in a window extending from $-\pi$ to $+\pi$. This limits the amount of search necessary to find the best alignment of two polar images. Fig. 5 illustrates the idea. The top figure shows the initial polar coordinate representation of the rotated "P" from Fig. 4. Mapping the angles to the range $-\pi$ to $+\pi$ produces the result shown in the bottom of Fig. 5. Effectively, the points to the right of the $+\pi$ boundary were moved to the right of the $-\pi$ boundary.

6.2. Finding the optimal alignment using polar transform

To find the angular offset between two polar images, we use a slide-and-compare algorithm in which one

(footnote continued)

the other while taking their dot product at every step. The end result is a correlation value indicating the similarity of the two signals and the temporal delay between them. In our case, finding the optimal angle is equivalent to determining the delay.

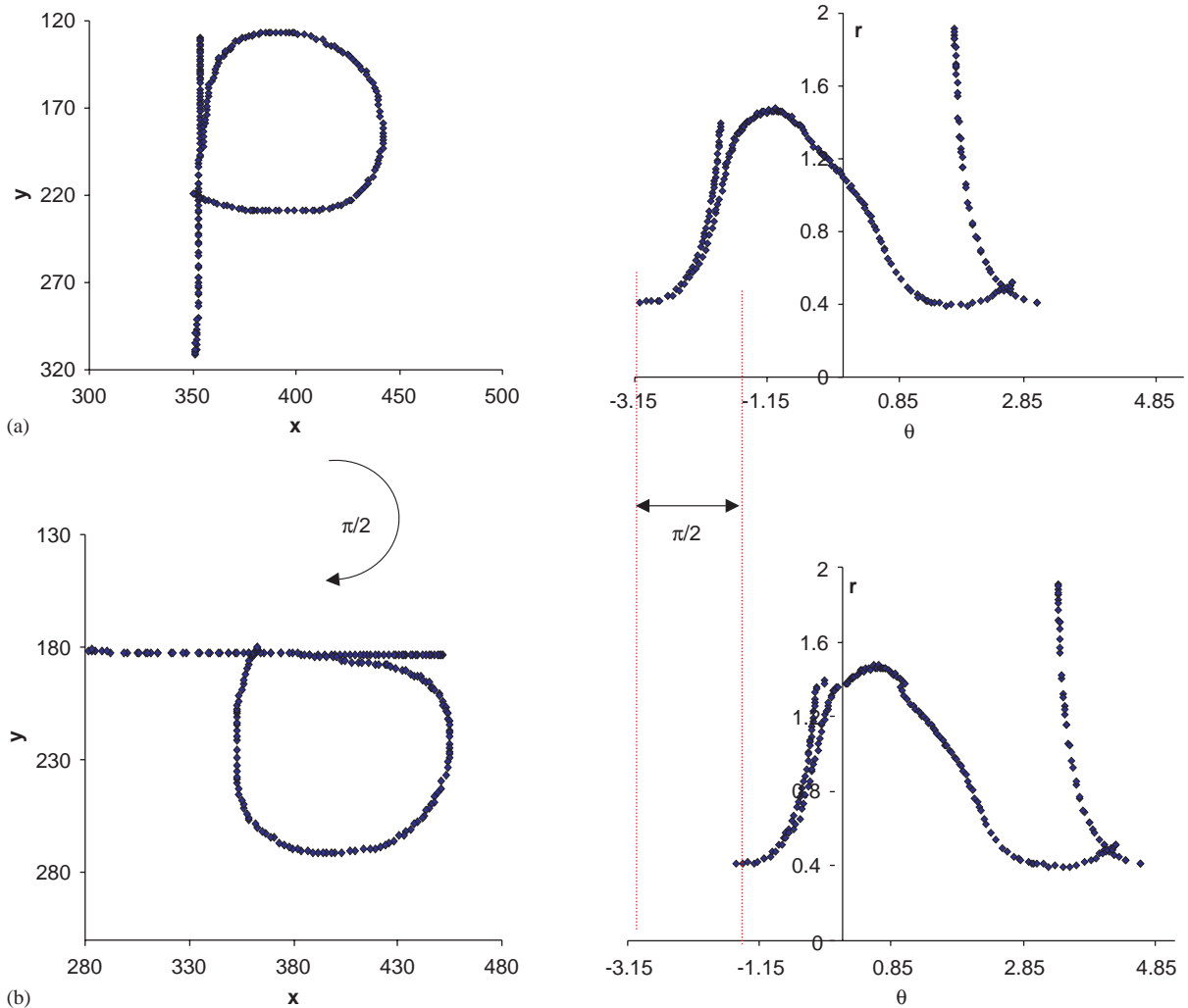


Fig. 4. (a) Left: letter “P” in screen coordinates; right: in polar coordinates. (b) When the letter is rotated in the x - y plane, the corresponding polar transform shifts parallel to the θ -axis.

image is incrementally displaced along the θ -axis. At each displacement, the two images are compared to determine how well they match. The displacement that results in the best match indicates how much rotation is needed to best align the original images. Because the polar images are in fact 2D binary patterns, we can use the template-matching techniques from Section 4 to match the polar images. In particular, we use the MHD as it is slightly more efficient than the regular Hausdorff distance (directed distances need not be sorted) and it performs slightly better than the Tanimoto and Yule coefficients in polar coordinates.

To use the MHD with polar images, we quantize the images into 48×48 templates just as we do for screen images (Section 3). Here again, we also use distance transforms to accelerate the computation of the MHD.

Although the idea remains the same, the periodicity of the θ -axis introduces a new subtlety in the distance calculations. Unlike in the x - y plane, pixels at the far right and left sides of a polar image (i.e., those close to the $-\pi$ and $+\pi$ boundaries) are in fact proximate as noted in Fig. 5. For example, a pixel just to the left of $+\pi$ will have a distance of 1 to a pixel that lies on the $-\pi$ boundary at the same r value. This periodicity is taken into account when the distance transform is computed. Because this subtlety is considered when computing distance, the distance transforms in polar coordinates need be computed only once, as distances do not change with shifts in the polar plane.

One difficulty with the polar transform is that data near the centroid of the original image is sensitive to the precise location of the centroid. Consider Fig. 6 that

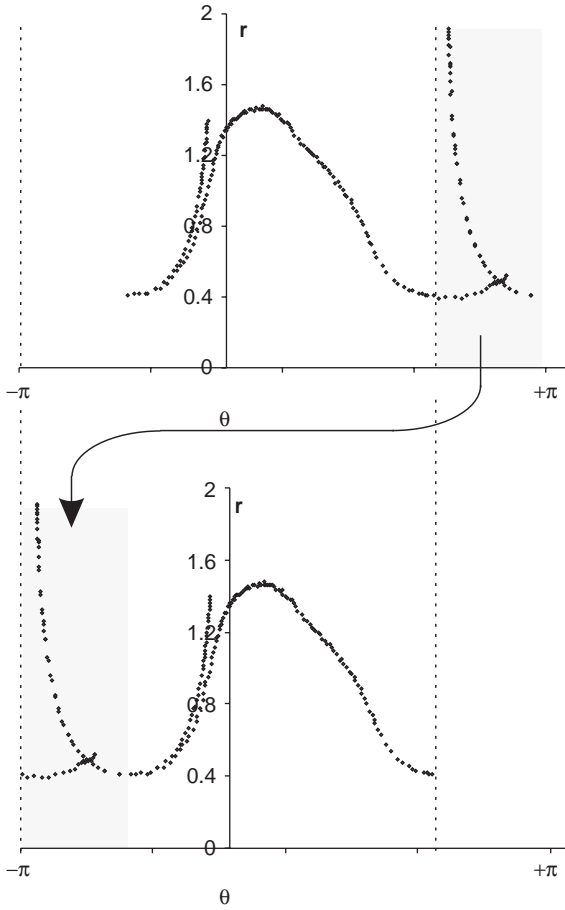


Fig. 5. Top: initial polar image of the rotated “P” from Fig. 4. Bottom: same image mapped to the range $-\pi$ to $+\pi$. In effect, the portions overstepping the $+\pi$ boundary are moved near the $-\pi$ boundary. The two images are equivalent.

shows two handwritten “T”s and their polar transforms. In the top image the tail curves slightly to the left while in the bottom image it curves slightly to the right. This difference causes the image centers to fall on the opposite sides of the tail, which, in turn leads to significant dissimilarity in the polar transforms for small r values. Naturally, the MHD is adversely affected by these variations.

To alleviate this problem, we introduce a weighting function $w(r)$ that attenuates the influence of pixels near the centroid of the screen image. Using this function, the directed MHD, previously introduced in Section 4.2, becomes:

$$h_{mod_weighted}(A, B) = \frac{1}{N_a} \sum_{a \in A} w(a_r) \min_{b \in B} \|a - b\|,$$

where a_r represents the radial coordinate of point a in the quantized polar image A . The directed distance from

B to A , $h_{mod_weighted}(B, A)$, is calculated similarly, and the maximum of the two directed distances is the MHD between A and B . Our weighting function has the form:

$$w(r) = r^{0.10}$$

which is shown graphically in Fig. 7. The exponent in the function has been determined experimentally for best performance. As shown, the function asymptotes near 1 for large values of r , and falls off rapidly for small values of r . By assigning smaller weights to the pixels near the image center, this function allows the Hausdorff distance between the polar images to be governed by the pixels that reside farther from the origin, hence reducing the sensitivity to the precise location of the centroid of the screen image.

7. Polar Transform as a pre-recognizer

The polar analysis allows us to find the angular difference between two patterns in an efficient way. Once the angle is determined, the patterns can be aligned properly in the x - y plane by a single rotation, and compared using the template-matching techniques in Section 4. (The rotation is performed before quantizing the screen image to produce a template.) However, before applying these techniques, we can use the matching information from the polar coordinate representation to filter out many of the unlikely definitions.

We have found that the degree of match between two polar images provides a reasonable estimate of the match of the original screen images. In fact, if it were not for the imprecision of the polar transform for small r values, the entire recognition process could be performed exclusively in the polar plane. The match in polar coordinates discounts data near the centroid of the screen image, which can result in false positive matches (i.e., declaring a close match between two patterns when they are in fact dissimilar), but rarely results in false negative matches. Thus, the polar analysis can be used as a pre-recognition step to eliminate unlikely definitions. In practice, we have found that the correct definition for an unknown is among the definitions ranked in the top 10% by the polar coordinate matching. Thus, we discard 90% of the definitions before considering the match in screen coordinates.

This approach is conceptually similar to cascading presented in [8], where a simple classifier is used to reduce the number of classes before a more complex classifier with a more expensive classification rule is applied. In our case, however, the polar transform not only serves as a pre-elimination step but also as a means to efficiently achieve rotation invariance. We have found this dual functionality of the polar transform to be invaluable for achieving real-time performance on an otherwise computationally demanding task.

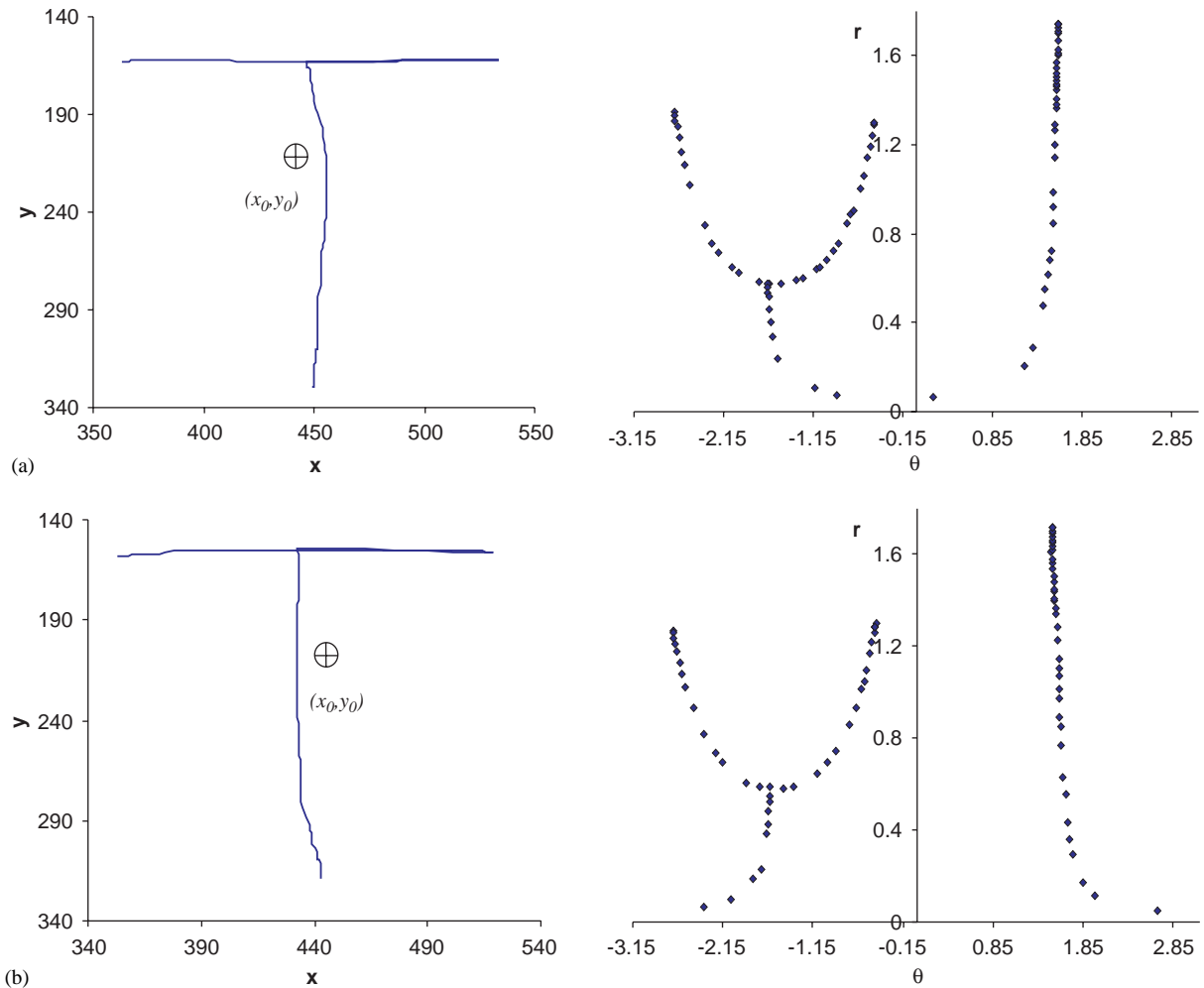


Fig. 6. The θ coordinate of the polar transform is sensitive to points near the image center. (a) Letter “T” and its polar transform. (b) Nearly the same letter except for the curl of the tail. This difference causes noticeable difference in the polar transform at small values of r .

8. User studies

To obtain an objective assessment of our approach, and to have a point of comparison with other systems, we conducted a formal user study consisting of two separate experiments. In the first experiment we used a set of 20 graphic symbols. In the second, we used digit recognition as our test bed.

Because the participants in our studies had little or no experience using the digitizing tablet and stylus, they were allowed to acquaint themselves with the hardware until they felt comfortable, which typically took about 2–3 min. Each experimental session involved only data collection, the data were processed at a later time. This approach was chosen to prevent participants from adjusting their drawing style based on our program’s output. During data collection, if users were not pleased

with what they drew, which occasionally occurred due to the unintentional slip of the stylus, they were allowed to redraw the symbol. However, participants rarely used this option. Additionally, although users were not instructed to draw symbols in any particular way, we found that there was no over stroking. Hence, future user studies will be necessary to formally quantify how well our approach handles the sorts of “sketchy” symbols shown in Fig. 1.

8.1. Graphic symbol recognition

Five users participated in the graphic symbol recognition study. Each user was asked to provide three sets of the 20 symbols shown in Fig. 8, yielding a total of 60 symbols per user. (Sample data collected from the users can be found in Fig. 10 at the end of the article.) Four

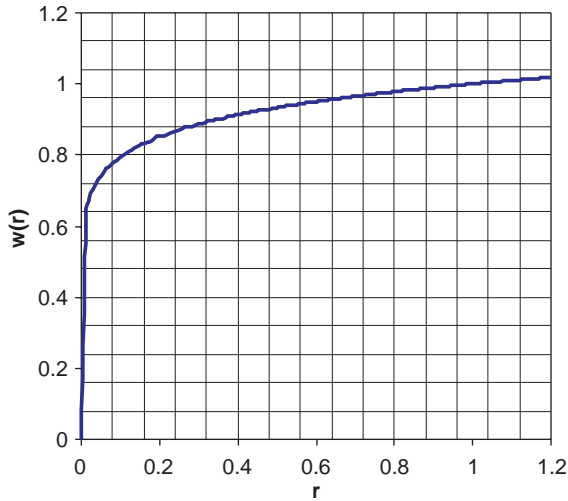


Fig. 7. Weighting function used to suppress the effect of pixels close to the image center.

different types of tests were conducted using the collected data. The tests differ based on (1) the number of definition symbols used for training, and (2) whether the test was conducted in a user-dependent (i.e., all training data from the given user) or user-independent manner. Below we detail each of these tests and the results.

Test 1. Single definition set, user-dependent: In this test, the recognizer was evaluated separately for each user. Each test consisted of three iterations, akin to the K -fold cross-validation technique with $K = 3$. In each iteration, one of the user's three sets of symbols was used for training, and the other two were used for testing. Different iterations employed different test sets. The performance for each user was computed as the average of the three iterations. The first row of Table 1 shows the results obtained from this study, averaged over the five users. In this table, the first column shows the recognition accuracy, or the rate at which the class ranked highest by the recognizer, is indeed the correct class. We call this the "top-one" accuracy. The second column shows the "top-two" accuracy, or the rate at which the correct class is either the highest or second highest ranked class. The last column shows the average recognition time in milliseconds.

Test 2. Two definition sets, user-dependent: This test is similar to the first test except, in each of the three runs, two sets of symbols were used for training while the remaining set was used for testing. Hence, during recognition, each unknown was compared to 40 definition symbols—2 definitions per symbol. As shown in the second row of Table 1, the additional training set increased the recognition accuracy at the expense of only a minor increase in the recognition times.

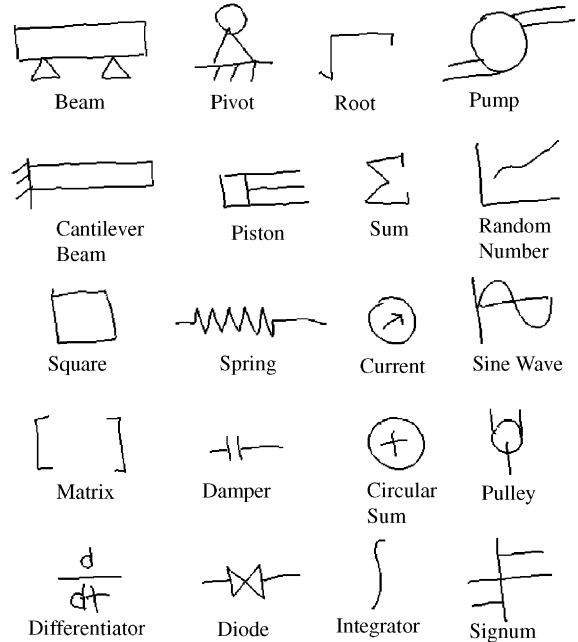


Fig. 8. Symbols used in the graphic symbol recognition experiment.

Table 1
Results from the graphic symbol recognition study

	Top 1 (%)	Top 2 (%)	Recog. time (ms)
Test 1	90.7	96.3	332
Test 2	95.7	98.3	354
Test 3	94.7	97.3	623
Test 4	98.0	99.0	674

The first two columns show the top-one and top-two accuracy, respectively. All tests were conducted on a Pentium 4 machine at 2.0 GHz. with 256 MB of RAM.

Test 3. Twelve definition sets, user-independent: The aim in this test was to evaluate the recognizer when the training and test sets belonged to different users. When testing a particular user's data, the training database consisted of all users' symbol sets excluding the data from the user under consideration. In each run, the database thus consisted of a total of twelve sets: three sets from each of the four users not involved in that particular test. In effect, this test mimics a walk-up-and-draw scenario in which the user works directly from a pre-trained recognizer without providing his or her own training symbols. The third row of Table 1, shows the performance obtained in this setting.

Test 4. Fourteen definition sets, partial user-dependence: The difference between this test and the previous

Table 2

Results from the digit recognition study. All tests were conducted on a Pentium 4 machine at 2.0 GHz. with 256 MB of RAM

	Top 1 (%)	Top 2 (%)	Recog. time (ms)
Test 1	95.4	98.3	211
Test 2	97.7	98.5	225
Test 3	91.8	95.5	516
Test 4	97.7	99.2	586

one is that, the training database additionally contained two symbol sets from the user being tested, in addition to the twelve sets from other users. In terms of training sets employed, this experiment is thus a hybrid of Test 2 and Test 3. As shown in the last row of Table 1, the top-one accuracy in this case reaches 98%.

8.2. Digit recognition

Nine users participated in the digit study. Users were asked to provide three sets of digits from “0” to “9”, yielding a total of 30 digits per user.

Our examination of the collected data revealed that frequent misclassifications occurred due to a confusion between “6” and “9”, and occasionally between “2” and “7”, both of which are reasonable errors given the rotation invariant nature of our recognizer. As a remedy, we adjusted the recognizer in this study so that search for the correct orientation of a digit was restricted to $\pm 90^\circ$ of the digits’ original orientation. We believe this restriction on rotation is reasonable for the digit recognition study as traditional digit recognizers assume a fixed orientation.

We conducted the same four tests described in the symbol recognition study. However, with nine rather than five participants, Test 3 now has 24 training sets rather than 12, and Test 4 has 26 rather than 14. Table 2 shows the results obtained from this study.

State-of-the-art hand-drawn digit recognition systems achieve recognition rates above 96–97% in user-independent settings [7]. We achieve about 91.8% accuracy in a user-independent setting (with rotation limited to $\pm 90^\circ$). Nevertheless, we consider our approach to be quite attractive given that it works from only a handful of training examples. As one would expect, if the problem is to recognize digits only, it is better to use a dedicated digit recognizer. However, if the problem involves user defined symbols, our approach has distinct advantages.

9. Discussion

In the graphic symbol recognition study, we consider the top-two classification performance to be of con-

siderable importance, as a common method for correcting recognition errors in many interactive applications is to display a short list of potential candidates from which the user can pick the intended one. In such cases, if the correct class was not selected by the recognizer, it should at least be near the top of the list. Similarly, in systems that consider context, such as [1], the shape recognizer may produce a set of likely candidates which can be further analyzed using contextual information. In such cases, the system may decide that the second or third choice from the recognizer is the correct interpretation, for example. For such approaches to work, the correct interpretation must be near the top of the list of choices.

We believe that the results of our user study are quite promising when compared to results reported in the literature. For example, Landay and Myers [19] report a recognition rate of 89% on a set of 5 single-stroke editing gestures. In our case, however, there are 20 symbol definitions that can be drawn with any number of strokes. In a study involving 7 multi-stroke and 5 single-stroke shapes, Fonseca and Jorge [20] report recognition rates around 92%. In that study, half of the subjects were experts in using the hardware. Also, the recognizer required the shape features to be manually encoded for each individual shape, which makes training new shapes difficult. On a database of 13 symbols, Hse and Newton [21] report a recognition rate of 97.3% in a user-dependent setting, and 96.2% in a user-independent setting. Each symbol was trained using 30 samples. On a database of 20 symbols, we achieve an accuracy of 95.7% in a user-dependent setting, where each symbol was trained with 2 samples (Test 2). Likewise, with the same 20 symbols, we achieve an accuracy of 94.7% in a user-independent setting, where each symbol was trained with 12 samples (Test 3).

To evaluate the efficiency of our polar coordinate analysis, we conducted a separate experiment in which the angular alignment of the images was computed in screen coordinates via incremental rotations. This not only bypassed the polar coordinate approach for computing optimal alignment, but also bypassed the accompanying pre-recognition step in which unlikely definitions are pruned. With these modifications, the average recognition time for Test 1 of the graphical symbol and digit recognition experiments increased to 3590 and 1350 ms, respectively, while the recognition accuracy remained the same in both cases. As these results suggest, the polar analysis provides significant savings in overall processing time without any decrease in accuracy.

10. Limitations

Our approach is insensitive to translation, rotation, and uniform scaling, but is sensitive to non-uniform

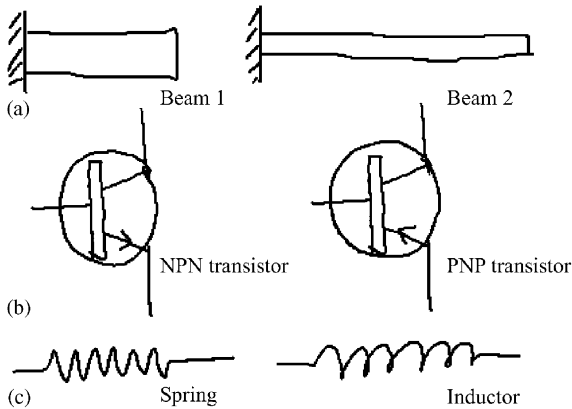


Fig. 9. Examples illustrating the limitations of the approach. (a) Two cantilever beams that are geometrically different but semantically equivalent. Our recognizer would treat them as different. (b) Transistors differ only by the directions of their arrows. Quantization may obscure this difference. (c) A spring and an inductor. Quantization may obscure the differences between them.

scaling. The latter property is advantageous when distinguishing between, say, a square and a rectangle. In other cases, however, it may be a hindrance. Consider, for example, the cantilever beams shown in Fig. 9a. Although the two beams are quite different when taken literally as two bitmap images, they would nonetheless have equivalent meanings in most practical situations. Our program, however, would treat these two symbols as being different, just the same way it would treat a square and a rectangle as different. For this sort of problem, the topology plays a much more important role than does the shape. In this case, structural matching methods [3,22], which put more emphasis on the pattern's topology, might be more appropriate.

Another limitation of our approach is that the quantization of input symbols into templates may wash out small image details. Hence, the performance of our recognizer will decrease for symbols that differ exclusively by such details. For instance, the recognizer will have difficulty distinguishing the two transistors shown in Fig. 9b, as the two shapes are similar except for the direction of their arrows. For similar reasons, the recognizer may have difficulty distinguishing between the spring and the inductor symbols shown in Fig. 9c. In such cases, it may be necessary to increase the resolution of templates at the expense of increased computation.

11. Related work

Research in automatic understanding of freehand drawings has produced a wide variety of representation and recognition techniques. In graph-based symbol

recognition methods [22,23], symbols are segmented into geometric primitives, and graphs encode both the intrinsic attributes of the primitives and the geometric relationships between them. Recognition is formulated as a graph-subgraph isomorphism problem. The practical limitations of graphical approaches are their computational complexity and their sensitivity to the segmentation process.

As an alternative to graphical methods, feature-based methods have also been used for symbol recognition. Fonseca et al. [24] use features such as the smallest convex hull that can be circumscribed around the shape, the largest triangle that can be inscribed in the hull, and the largest quadrilateral that can be inscribed. Because their classification relies on aggregate features of the pen strokes, it might be difficult to differentiate between similar shapes. Rubine [4] describes a trainable gesture recognizer designed for gesture-based interfaces. The recognizer is applicable only to single-stroke symbols, and is sensitive to the drawing direction and orientation. Matsakis [25] describes a system for converting handwritten mathematical expressions into a machine-interpretable typesetting command language. Each symbol requires a multitude of training examples, where each example must be preprocessed to eliminate variations in drawing directions and stroke orderings. However, the preprocessing makes their approach sensitive to rotations. Kara et al. [2] describe a trainable recognizer that uses nine geometric features to construct concise probabilistic models of input symbols. The approach is suitable for multi-stroke symbols with arbitrary drawing orders and orientations. However, it is sensitive to the results of stroke segmentation, and additionally does not handle over stroking and different line styles. The feature-based methods outlined above typically require a multitude of training examples to reliably learn new symbol definitions. Our recognizer, on the other hand, requires very little training data, and in fact can work from just a single example of a symbol.

Inspired by the advances in speech recognition, some systems facilitate recognition by requiring objects to be drawn with a predefined sequence of pen strokes [6,26]. While useful at reducing computational complexity, the strong temporal dependency in these methods forces the user to remember the correct order in which the strokes must be drawn.

Another class of systems is based on a descriptive approach to encoding shape information [27,28]. In these systems, geometric primitives and their relationships are either hand-coded for each new shape, or are constructed from heuristic rules designed to capture dominant shape features. The resulting shape descriptions then form the basis for a constraint-based matching procedure. While these approaches result in precise structural descriptions enhancing their discrimination power, they do not easily

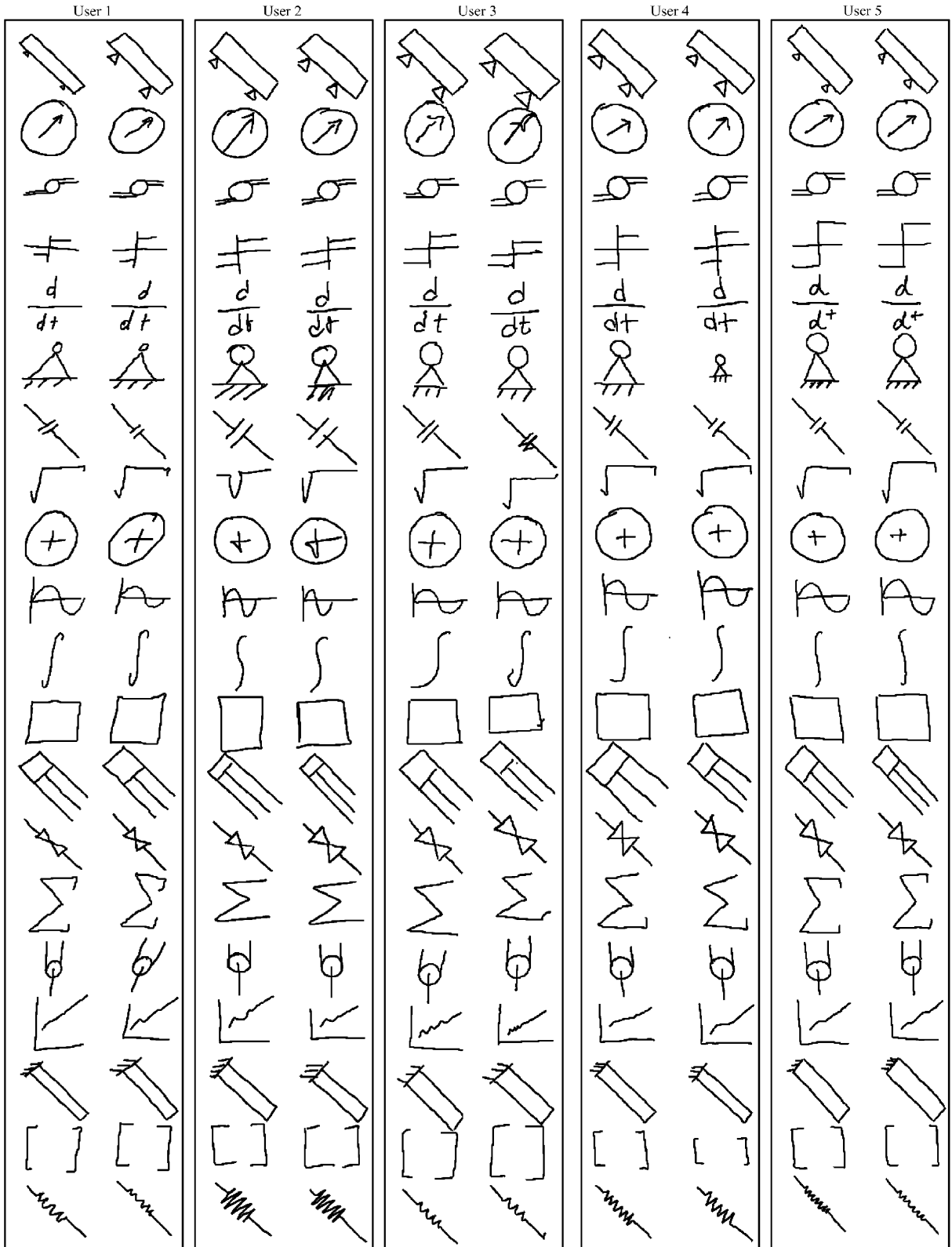


Fig. 10. Data collected for the graphic symbol recognition user study. Long thin symbols are shown rotated 45° to make the figure more compact.

lend themselves to recognizing sketchy symbols that are drawn with little precision.

Shilman and Viola describe a system for simultaneously grouping and recognizing sketched text and graphics [29]. Their approach first uses a constrained search-based optimization to generate candidate stroke groups, and then evaluates each candidate using a fast bitmap-based recognizer. To prevent exhaustive search, an upper limit is imposed on the number of strokes for each symbol, which makes the approach sensitive to over stroking and different line styles. Additionally, the features used in the bitmap-based recognizer makes the approach sensitive to rotations. Belongie et al. [30], in their work on point sets, use “shape contexts” for shape matching and recognition. In their approach, each point in the object maintains a histogram of log-polar bins that cluster the neighboring points. These histograms are then used to find the correspondence between points from different objects, which then helps produce a similarity score between two objects. Their approach is rotation invariant, and has the ability to handle relatively large deformations. However, large training sets are required to achieve high recognition rates.

Recognizing shapes irrespective of their orientation has been one of the key issues in pattern recognition. Existing solutions to image-based rotation invariance include moment invariants such as Hu transforms, Zernike Moments, affine moment invariants, Fourier descriptor methods, wavelet transforms and circular harmonic filters. Some of these methods are more suitable to gray-level images than bitmaps, while some are applicable only to closed contours or silhouettes. For the types of problems considered in this work, Hu transforms [31] and Zernike moments [32] stand out as the most suitable solutions. While these methods are most effective when applied to machine printed patterns, Hse and Newton [21] recently applied Zernike moments to classify hand-drawn symbols and they report promising results.

12. Conclusion

We have described a trainable, multi-stroke, hand-drawn symbol recognizer designed to be used in sketch-based interfaces. With our techniques, symbol definitions can be learned from single prototype examples, allowing users to train new symbols or adjust existing ones on the fly. Our approach avoids a number of problematic issues in symbol recognition, such as segmentation and feature extraction. Also, our approach is tolerant of over stroking, missing and extra pen strokes, variations in line style, and variations in drawing order.

Our recognizer employs a two-step recognition scheme. First, polar coordinates are used to efficiently determine angular alignment and eliminate unlikely

definitions. Then, the remaining definitions are examined in screen coordinates using an ensemble of four template classifiers. Each classifier produces a list of definitions ranked according to their similarity or dissimilarity to the unknown symbol. The results of the individual classifiers are combined to produce the recognizer’s final decision. Our experiments have demonstrated that this two-step approach is an order of magnitude more efficient than performing the alignment and recognition solely in screen coordinates.

We conducted two user studies to evaluate the performance of our recognizer. One study focused on handwritten, numeric digits. In this study, our general-purpose recognizer achieved recognition rates that were a little lower than those of dedicated digit recognizers. The latter, however, typically employ extensive training data. We believe that our system is a worthwhile compromise, as our approach allows novel symbols to be trained from single prototype examples. The other user study considered a set of graphic symbols typical of those found in the engineering domain (for graphic symbols see Fig. 10). We believe the results of this study are even more promising as our approach allows symbols to be drawn with any number of strokes drawn in any order, and new symbols can be learned from single examples.

References

- [1] Kara LB, Stahovich TF. Hierarchical parsing and recognition of hand-sketched diagrams. In: User Interface Software Technology (UIST), 2004.
- [2] Kara LB, Gennari L, Stahovich TF. A sketch-based interface for the design and analysis of simple vibratory mechanical systems. In: ASME International Design Engineering Technical Conferences, 2004.
- [3] Gennari L, Kara LB, Stahovich TF. Combining geometry and domain knowledge to interpret hand-drawn diagrams. In: AAAI Fall Symposium Series 2004: Making Pen-Based Interaction Intelligent and Natural, 2004.
- [4] Rubine D. Specifying gestures by example. *Computer Graphics* 1991;25:329–37.
- [5] Kimura TD, Apte A, Sengupta S. A graphic diagram editor for pen computers. *Software Concepts and Tools* 1994; 82–95.
- [6] Yasuda H, Takahashi K, Matsumoto T. A discrete hmm for online handwriting recognition. *International Journal of Pattern Recognition and Artificial Intelligence* 2000;14(5):675–88.
- [7] LeCun Y, Jackel LD, Bottou L, Brunot A, Cortes C, Denker JS, Drucker H, Guyon I, Muller UA, Sackinger E, Simard P, Vapnik V. Comparison of learning algorithms for handwritten digit recognition. In: International Conference on Artificial Neural Networks, Paris; 1995. p. 53–60.
- [8] Alimoglu F, Alpaydin E. Combining multiple representations for pen-based handwritten digit recognition.

- ELEKTRIK: Turkish Journal of Electrical Engineering and Computer Sciences 2001;9(1):1–12.
- [9] Brunelli R, Poggio T. Face recognition: features versus templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 1993;15(10):1042–52.
- [10] Rucklidge WJ. Efficient visual recognition using the Hausdorff distance. *Lecture Notes in Computer Science*, vol. 1173. Berlin: Springer, 1996.
- [11] Cheung K-W, Yeung D-Y, Chin RT. Bidirectional deformable matching with application to handwritten character extraction. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2002;24(8):1133–9.
- [12] Miller EG, Matsakis NE, Viola PA. Learning from one example through shared densities of transforms. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2000. p. 464–71.
- [13] Fligner M, Verducci J, Bjoraker J, Blower P. A new association coefficient for molecular dissimilarity. In: *The Second Joint Sheffield Conference on Chemoinformatics*. Sheffield, England; 2001.
- [14] Tubbs JD. A note on binary template matching. *Pattern Recognition* 1989;22(4):359–65.
- [15] Dubuisson MP, Jain AK. A modified hausdorff distance for object matching. In: *12th International Conference on Pattern Recognition*. Jerusalem, Israel; 1994. p. 566–8.
- [16] Kittler J, Hatef M, Duin RPW, Matas J. On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 1998;20(3):226–39.
- [17] Davoli R, Tamburini F, Gaioni R. A translation, rotation and scale invariant transform for grey scale images: a parallel implementation. In: *Fifth European SGI/Cray MPP Workshop*, Bologna, Italy, 1999.
- [18] Wolberg G, Zokai S. Robust image registration using log-polar transform. In: *IEEE International Conference on Image Processing*. Vancouver, Canada; 2000.
- [19] Landay JA, Myers BA. Sketching interfaces: toward more human interface design. *IEEE Computer* 2001;34(3):56–64.
- [20] Fonseca MJ, Jorge JA. Using fuzzy logic to recognize geometric shapes interactively. In: *Proceedings of the 9th International Conference on Fuzzy Systems (FUZZ-IEEE 2000)*, San Antonio, USA; 2000.
- [21] Hse H, Newton AR. Sketched symbol recognition using zernike moments. Technical report, EECS, University of California, 2003.
- [22] Calhoun C, Stahovich TF, Kurtoglu T, Kara LB. Recognizing multi-stroke symbols. In: *AAAI Spring Symposium on Sketch Understanding*. 2002. p. 15–23.
- [23] Bunke H, Jiang X. Graph matching and similarity. In: *Teodorescu H-N, Mlynek D, Kandel A, Zimmermann HJ, editors. Intelligent systems and interfaces*. Dordrecht, The Netherlands: Kluwer Academic Publishers; 2000.
- [24] Fonseca MJ, Pimentel C, Jorge JA. Cali—an online scribble recognizer for calligraphic interfaces. In: *AAAI Spring Symposium on Sketch Understanding*. 2002. pp. 51–8.
- [25] Matsakis NE. Recognition of handwritten mathematical expressions. Master thesis, MIT, 1999.
- [26] Sezgin TM, Davis R. Hmm-based efficient sketch recognition. In: *International Conference on Intelligent User Interfaces (IUI'05)*, New York, 2005.
- [27] Hammond T, Davis R. Automatically transforming symbolic shape descriptions for use in sketch recognition. In: *19th National Conference on Artificial Intelligence (AAAI-2004)*, 2004.
- [28] Veselova O, Davis R. Perceptually based learning of shape descriptions for sketch recognition. In: *The Nineteenth National Conference on Artificial Intelligence (AAAI-04)*, 2004.
- [29] Shilman M, Viola P. Spatial recognition and grouping of text and graphics. In: *EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling*. 2004.
- [30] Belongie S, Malik J, Puzicha J. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2002;24(4):509–22.
- [31] Hu MK. Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory* 1962;IT-8:179–87.
- [32] Khotanzad A, Hong YH. Rotation invariant image recognition using features selected via a systematic method. *Pattern Recognition* 1990;23(10):1089–101.