Curve Matching Using the Fast Marching Method

Max Frenkel and Ronen Basri

Weizmann Institute of Science, Rehovot, Israel
{maksimf,ronen}@wisdom.weizmann.ac.il

Abstract. Common techniques for curve alignment find a solution in the form of a shortest network path by means of dynamic programming. In this paper we present an approach that employs Sethian's Fast Marching Method to find the solution with sub-resolution accuracy and in consistence with the underlying continuous problem. We demonstrate how the method may be applied to compare closed curves, morph one curve into another, and compute curve averages. Our method is based on a local curve dissimilarity function $F(t, s)$ that compares the two input curves $\mathcal{C}_1(t)$ and $\mathcal{C}_2(s)$ at given points t and s. In our experiments, we compare dissimilarity functions based on local curvature information and on shape contexts. We have tested the algorithm on a database of 110 sample curves by performing "best matches" experiments.

1 Introduction

Determining similarity is a problem that lies at the heart of computer vision. Given two line drawings of say a digit *2*, one that has a loop at the bottom and the other that has a sharp corner, such as the curves in Fig. 1(a), how can we judge that they both depict the digit *2* and not two different digits? In order to answer this question, we need to define some sort of a similarity measure to relate objects of the same class and distinguish between objects of different classes.

Suppose that the input curves are given to us as a sequence of curve coordinates in the order in which they were produced by a pen. One way of assessing similarity between such curves is by means of dynamic programming (a brief review of past work is given in Sect. 1.1). Input curves are treated as if they were strings, and curve points - as if they were letters. Just like a string can be edited to transform it into another string by inserting, deleting and relabelling characters, the first curve is stretched, shortened, and bent to match the second curve. For instance, in order to match SHALL with HELLO, we can delete an S, change an A to an E and insert an O. If we associate a cost with each of the three basic operations, then by summing the costs during each edit sequence we obtain a way of comparing different sequences and picking the optimal edit operation. The *edit distance* between the strings is defined as the minimal such sum of local costs. The string-to-string correction algorithm [18] employs dynamic programming to compute this distance between strings S_1 and S_2 and to recover a *trace* or a *matching* - a set of non-crossing lines between the letters of S_1 and the letters of S_2, where each line corresponds to a relabel operation in the optimal edit sequence. In the above example, the matching would be $\{$(S,ε), (H,H), (A,E), (L,L), (L,L), (ε, O)$\}$, where ε corresponds to the empty character.

In a similar fashion, discretized curves can be compared using dynamic programming by summing up the local costs of curve deformation operations, and the optimal *curve matching* may be recovered. In this paper we intend to study the dynamic programming approach, point out its inherent drawback related to curve discretization, and suggest a different way of obtaining the solution using Sethian's Fast Marching Method that avoids the shortcoming of the edit distance approach. Possible areas of application of curve matching include handwritten character recognition [17], image indexing [1], and tracking [4]. We will test our algorithm by comparing various handwritten digits and letters.

1.1 Past Work

Edit distances were used extensively in the past to compare curves. Here we only mention a few relevant examples. [4] introduced a measure that relied on intrinsic properties of curves (such as curvature), and compared curves using dynamic programming. [3] proposed a set of properties that a desired similarity function should possess. Among other things, these properties preferred the deformations that preserve the part structure of objects over those that modify the parts. [12, 16] emphasized the importance of a symmetric treatment of curves. [12] also presented an efficient method (based on [9]) to treat closed curves.

A wide variety of other approaches have been taken to solve the problem of curve alignment. For example, [6] compared non-rigid object shapes by measuring the amount of deformation required to register the shapes exactly. Assuming that the amount of deformation between shapes is small, they used gradient descent to determine the deformation parameters. [11] described shapes by a list of properties and their relations and used pattern matching techniques to judge similarity. [10] deformed shapes by aligning the principal modes of their mass and stiffness matrices.

Recently Belongie et al. [2] introduced the *shape context*, a rich local shape descriptor that aids in judging shape similarity and in finding correspondences between similar shape points.

Matching Unordered Point Sets with Shape Contexts. The method employed by Belongie et al. solves a more general problem of matching an unordered set of n points $\mathcal{P} = \{\boldsymbol{p}_1, ..., \boldsymbol{p}_n\}, \boldsymbol{p}_i \in \mathbb{R}^2$ sampled somehow (e.g., by an edge detector) from the given input image. We would like to match each point $\boldsymbol{p}_i$ from the first set to some point $\boldsymbol{q}_j$ from the second set. Consider the set of $n-1$ vectors that originate at $\boldsymbol{p}_i$ and extend to $\boldsymbol{p}_l$, for $l \neq i$. A shape context h_i is a histogram of coordinates of $\boldsymbol{p}_l$ relative to $\boldsymbol{p}_i$, namely $\boldsymbol{r}_{li} = \boldsymbol{p}_l - \boldsymbol{p}_i$. In the experiments of Belongie et al., h_i is a 5 by 12 array. Length of $\boldsymbol{r}_{li}$ is quantized into 5 bins, and its orientation is quantized into 12 bins. Thus the shape context encodes the distribution of relative positions in a robust and compact way.

From here, Belongie et al. proceed to define $C_{ij} = C(\boldsymbol{p}_i, \boldsymbol{q}_j)$ as the cost of matching $\boldsymbol{p}_i$ with $\boldsymbol{q}_j$ and use the χ^2 test statistic to compare the histograms $h_i(k)$ and $h_j(k)$ at $\boldsymbol{p}_i$ and $\boldsymbol{q}_j$ as follows:

$$C(\boldsymbol{p}_i, \boldsymbol{q}_j) = \frac{1}{2}\sum_{k=1}^{K} \frac{[h_i(k) - h_j(k)]^2}{h_i(k) + h_j(k)} \ , \tag{1}$$

where K is the total number of bins in the histograms. Once the local matching cost is defined, the total cost $H(\pi)$ is minimized.

$$H(\pi) = \sum_i C(\boldsymbol{p}_i, \boldsymbol{q}_{\pi(i)}) \ , \tag{2}$$

where π is a permutation of the second shape indexes, i.e., the matching is one-to-one. The permutation π that achieves the above minimum can be obtained using various weighted bipartite graph matching algorithms that take the matrix C_{ij} as input.

The curve matching problem that we are facing is more specific in that it requires that the points in the input set be ordered. The order of the points is an extra piece of information that may be utilized. The edit distance algorithm that we review next does just that.

The Dynamic Programming Approach to Curve Matching. Denote the curve segments to be matched by $\mathcal{C}_1(t) = (x_1(t), y_1(t))$, $t \in [0, m]$ and $\mathcal{C}_2(s) = (x_2(s), y_2(s))$, $s : [0, n]$, where t is arc-length, x_1 and y_1 are the coordinates of curve points, m is curve length, and each is similarly defined for $\mathcal{C}_2$. A correspondence between $\mathcal{C}_1$ and $\mathcal{C}_2$ is specified by a function $s(t)$, which is a monotonic, one to one mapping from arc-length onto arc-length so that the point $\mathcal{C}_1(t)$ is matched with the point $\mathcal{C}_2(s(t))$. Given a correspondence $s(t)$, a similarity measure between $\mathcal{C}_1$ and $\mathcal{C}_2$ can be defined by a distance function $C(\mathcal{C}_1, \mathcal{C}_2)$ that judges the cost of deforming one curve into the other, e.g., [3]:

$$C(\mathcal{C}_1, \mathcal{C}_2) = \int_{\mathcal{C}_1} F(\kappa_1(t), \kappa_2(s(t)), \frac{ds}{dt})dt \ , \tag{3}$$

where $\kappa_1(t)$ is the curvature of $\mathcal{C}_1$ at t and $\kappa_2(s)$ is the curvature of $\mathcal{C}_2$ at $s(t)$. The cost function conveys the goodness of a correspondence between the two curves. C integrates over the curves a measure F of local differences of corresponding subsegments. The distance (dissimilarity) $C^*(\mathcal{C}_1, \mathcal{C}_2)$ between $\mathcal{C}_1$ and $\mathcal{C}_2$ is the distance that minimizes C over all possible matchings $s(t)$. One example of a local cost function is

$$F(\kappa_1, \kappa_2, s') = |\kappa_2 s' - \kappa_1| + \lambda|s' - 1| \ , \tag{4}$$

where the first term penalizes bending and the second term penalizes stretching.

To discretize the problem, the two input contours are represented as ordered chains: $T = \{u(t) : t = 0, 1, ..., m\}$ and $S = \{v(s) : s = 0, 1, ..., n\}$, where $u(t)$

are the coordinates of the first contour, parameterized by t, and $v(s)$ are the coordinates of the other contour, parameterized by s. We then seek to find the optimal matching $(\alpha_1, ..., \alpha_N)$, that minimizes $C(T, S)$. Here

$$\alpha_k = (u(t_k), v(s_k)),\ t_k \in \{1, ..., m\}, s_k \in \{1, ..., n\}, k = 1, ..., N\ , \tag{5}$$

and if it is assumed that the endpoints of the curves match, then $\alpha_1 = (u(1), v(1))$ and $\alpha_N = (u(m), v(n))$.

Returning to the string matching problem described in the introduction, the *edit distance* between strings is defined as the cost of the optimal sequence of edit operations: relabelling, deleting, and inserting characters [18]. In the curve matching problem, curve points serve as "characters". Bending a curve (changing the curvature) or stretching it at a point corresponds to relabelling a letter. Removing a point of $\mathcal{C}_1$ is analogous to a deletion, and removing a point on $\mathcal{C}_2$ - to an insertion. So the cost measure (3) is related to the optimal amount of *deforming* needed in order to make the two curves identical [3].

This problem can also be viewed as an optimal path problem. We let each match $w = (u(t), v(s))$ be a node in a graph G and link it to those nodes that correspond to predecessor matches $(u(t_\mathrm{p}), v(s_\mathrm{p}))$ of w. Further, we assign a weight to an edge between a predecessor match and the current match that corresponds to the cost of deforming the segment $[u(t_\mathrm{p}), u(t)]$ so that it coincides with the segment $[v(s_\mathrm{p}), v(s)]$. Then, finding the matching with the lowest cost reduces to finding an optimal path in G from the start node to the end node. An example of such a network path is shown in Fig. 1(b). Diagonal path segments signify a local "relabel" operation, while horizontal/vertical segments signify removing points on one of the curves. Stretching a curve at a point is marked by red (dotted in black and white production) diagonal segments that jump across several cells [12].

Sethian [14] discusses network path algorithms and mentions that the network imposes an unnatural metric on the problem. Imagine a rectangular graph with unit weights on all edges. If we were computing a path from the bottom left corner to the top right corner according to the L_1 distance metric, then several paths would qualify as "shortest" (see Fig. 1(c)). However, the true solution is a straight line diagonal between the source and the destination points (Fig. 1(d)), and none of the obtained network paths would be of desired length even if the resolution of the graph was refined (i.e. if the curves were sub-sampled). The drawback is in the discretization itself in that it is *inconsistent* with the underlying continuous problem. The true solution must be described by the underlying differential equation, and if we had managed to find a continuous solution then we would in fact solve the problem with sub-resolution accuracy. The Fast Marching Method provides the means to achieve that goal.

An interesting attempt to modify the dynamic programming procedure to allow for sub-pixel matching of curves has been proposed in [13]. This method attempts to store a parametric description of the optimal cost at every node, and this limits the method to quite specific cost functions. The Fast Marching Method provides a simpler and more generic way to achieve this goal.

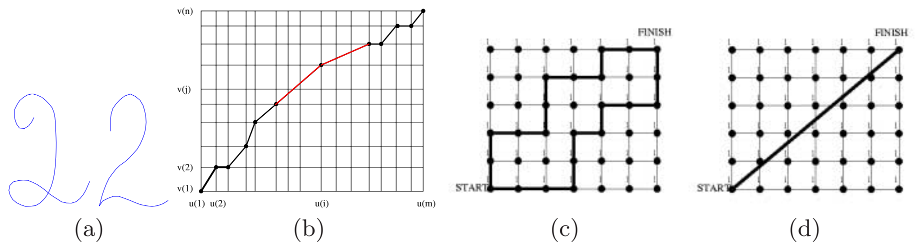


Fig. 1. (a) An example of a *2* drawn in different ways. (b) An example of a discrete alignment curve given as a shortest path in a graph. (c) Multiple "shortest" network paths and (b) the optimal diagonal path given by the Fast Marching Method

1.2 The Fast Marching Method

The Fast Marching Method is an $O(N \log N)$ technique for solving the Eikonal equation, $|\nabla T(x,y)| = F(x,y)$, for T given F on a rectangular grid, where N is the total number of grid points. An important property of the Fast Marching Method is that it converges to the continuous viscosity solution of the Eikonal equation as the rectangular numerical grid is refined [15]. In computer vision, the Fast Marching Method has been applied to various applications including active contours [5] and shape from shading [8].

The algorithm is based on the following upwind approximation of the Eikonal equation:

$$((\max(D_{ij}^{-x}T, -D_{ij}^{+x}T, 0))^2 + (\max(D_{ij}^{-y}T, -D_{ij}^{+y}T, 0))^2)^{1/2} = f_{ij} \ , \qquad (6)$$

where $f_{ij} = F(i\Delta x, j\Delta y)$, and $D_{ij}^{-x}T = (T_{ij} - T_{i-1,j})/\Delta x$ is the standard backwards derivative approximation, $D_{ij}^{+x}T = (T_{i+1,j} - T_{ij})/\Delta x$ is the standard forward derivative approximation in the x direction, and similarly for the y direction. In the case of a uniform grid, we have $\Delta x = \Delta y = 1$. Then, the approximation (6) may equivalently be written as

$$(\max(T_{ij} - T_1, 0))^2 + (\max(T_{ij} - T_2, 0))^2 = f_{ij}^2 \ , \qquad (7)$$

where $T_1 = \min(T_{i-1,j}, T_{i+1,j})$ and $T_2 = \min(T_{i,j-1}, T_{i,j+1})$, and the update step for $T_{i,j}$ consists of setting up the quadratic equation

$$T_{i,j} = T_1 + T_2 + \sqrt{2f_{ij}^2 - (T_1 - T_2)^2} \ . \qquad (8)$$

If the real solution does not exist, then we set $T_{i,j} = f_{ij} + \min(T_1, T_2)$, which corresponds to the case when one of the terms in the approximation is zero [8].

The central idea behind the Fast Marching Method is to systematically construct the solution T using only upwind values. The upwind difference structure of Eqn. (6) allows us to propagate the information *one-way*, from the smaller values of T to the larger values. The front is swept along by keeping a narrow band of grid points around the existing front in a heap structure and marching it forward bringing in unprocessed points and fixing the smallest computed values in the band [15].

2 Theoretical Background

Next, we present some results from the theory of curve evolution that are necessary in order to formulate our equations of motion. The following material is based on [7].

The optimal path between points A and B in $\mathbb{R}^2$ is defined by the weighted arc-length $d\tilde{\tau}^2 = F^2(t,s)d\tau^2$, where $d\tau = \sqrt{dt^2 + ds^2}$ is the Euclidean arc-length differential and $F(t,s)$ is the weight over the domain. We search for the path $\boldsymbol{c}(\tau) = (t(\tau), s(\tau))$, where τ is the arc-length parametrization of $\boldsymbol{c}$ with $|\boldsymbol{c}'(\tau)| = 1$. The necessary boundary conditions for $\boldsymbol{c}(\tau)$ are $\boldsymbol{c}(0) = A$ and $\boldsymbol{c}(L) = B$, where L is the total arc-length. The desired path should minimize $\min_{\boldsymbol{c}} \int F(\boldsymbol{c}(\tau))d\tau$, with $|\boldsymbol{c}'(\tau)| = 1$. For an arbitrary parametrization p of $\boldsymbol{c}$, the above geometric functional reads

$$\min_{\boldsymbol{c}} \int_0^1 F(\boldsymbol{c}(p))|\boldsymbol{c}'(p)|dp \ . \tag{9}$$

Lemma 1. *If a path $\boldsymbol{c}(p)$ satisfies the equation*

$$\nabla F|\boldsymbol{c}'(p)| = \frac{d}{dp}(F(\boldsymbol{c}(p))\boldsymbol{T}(p)) \ , \tag{10}$$

where $\boldsymbol{T}(p)$ is the unit tangent vector to $\boldsymbol{c}(p)$ defined as $\boldsymbol{T}(p) = \frac{\boldsymbol{c}'(p)}{|\boldsymbol{c}'(p)|}$, then $\boldsymbol{c}(p)$ achieves the minimum in (9).

Thus, Eqn. (10) is the Euler-Lagrange (EL) equation of the measure (9). Next, let $T : \mathbb{R}^2 \rightarrow \mathbb{R}$ be a weighted distance function with $|\nabla T| = F(t,s)$, where $|\nabla T|$ is evaluated at (t,s), with given boundary conditions $T(0,0) = 0$. Let us show that the gradient descent curves of T minimize the measure (9).

Lemma 2. *The gradient descent curves $\boldsymbol{c}(p) = (t(p), s(p))$ defined by the ODE $\boldsymbol{c}'(p) = \nabla T$ satisfy the Euler-Lagrange equation of the measure*

$$\int F(\boldsymbol{c}(p))|\boldsymbol{c}'(p)|dp \ .$$

Proof. First, let us observe that $\boldsymbol{c}'(p) = (t'(p), s'(p)) = \nabla T = (\frac{\partial T}{\partial t}, \frac{\partial T}{\partial s}) = (T_t, T_s)$. Next, we have $F(\boldsymbol{c}(p))\boldsymbol{T}(p) = |\nabla T|\frac{\nabla T}{|\nabla T|} = \nabla T$. Hence, the right hand side of Eqn. (10) has the form

$$\begin{aligned}\frac{d}{dp}(F(\boldsymbol{c}(p))\boldsymbol{T}(p)) = \frac{d}{dp}\nabla T &= \frac{d}{dp}(T_t(t(p), s(p)), \quad T_s(t(p), s(p))) \\ &= (T_{tt}t'(p) + T_{ts}s'(p), \quad T_{st}t'(p) + T_{ss}s'(p)) \ .\end{aligned}$$

On the other hand,

$$\nabla F|\boldsymbol{c}'(p)| = \nabla(|\nabla T|)|\nabla T| = \frac{(T_{tt}T_t + T_{st}T_s, \quad T_{ts}T_t + T_{ss}T_s)|\nabla T|}{(T_t^2 + T_s^2)^{1/2}}$$
$$= (T_{tt}t'(p) + T_{ts}s'(p), \quad T_{st}t'(p) + T_{ss}s'(p)) \ . \qquad \square$$

We have just seen that by solving the Eikonal equation $|\nabla T| = F(t,s)$ for $T(t,s)$ we can backtrack the optimal path by starting at the final location B and by stepping in the direction of the gradient of T. Note that if we set $F(t,s) = 1$, then the distance map $T(t,s)$ will be just a series of concentric circles corresponding to the Euclidean distance map, and the optimal path will be a straight line from B to A.

The weighted distance function $T(t,s)$ reconstructed from the point A may be thought of as the minimal cost required to travel from A to the point (t,s). In terms of the original functional, that is

$$T(t,s) = \min_{\boldsymbol{c}} \int_A^{(t,s)} F(\boldsymbol{c}(\tau))d\tau \ . \tag{11}$$

The level set curve $T(t,s) = C$ is the set of all points in $\mathbb{R}^2$ that can be reached with minimal cost C [14].

3 The Distance between a Pair of Curves

Equipped with this theory, we can now proceed to define a distance measure between a given pair of curves $\mathcal{C}_1(t) = (x_1(t), y_1(t)), \ t \in [0,m]$ and $\mathcal{C}_2(s) = (x_2(s), y_2(s)), \ s \in [0,n]$, where s and t are arc-length parameters and m and n are the lengths of the curves. Assuming that the endpoints of the input curves match, and given some local dissimilarity measure F, we are interested in a path $\boldsymbol{c}$ through t,s-space from $(0,0)$ to (m,n) such that

$$T(m,n) = \min_{\boldsymbol{c}} \int_{\boldsymbol{c}} F(\boldsymbol{c}(\tau))d\tau \ . \tag{12}$$

For such a path we define a distance measure between $\mathcal{C}_1$ and $\mathcal{C}_2$ as

$$d(\mathcal{C}_1, \mathcal{C}_2) = T(m,n) - \lambda\sqrt{m^2+n^2} + \left|1 - \frac{\min(m,n)}{\max(m,n)}\right| , \tag{13}$$

where λ is a *smoothing constant* such that $\lambda > 0$. Thus, we define the dissimilarity between $\mathcal{C}_1$ and $\mathcal{C}_2$ as the minimal sum of local dissimilarities between individual pairs of curve points, and here we note the similarity to the edit distance approach. The second term in this expression is needed for normalization and the third term penalizes global stretching of the curves. The significance of these terms will become apparent in the next section. Now, let us say a few words about our choice of F.

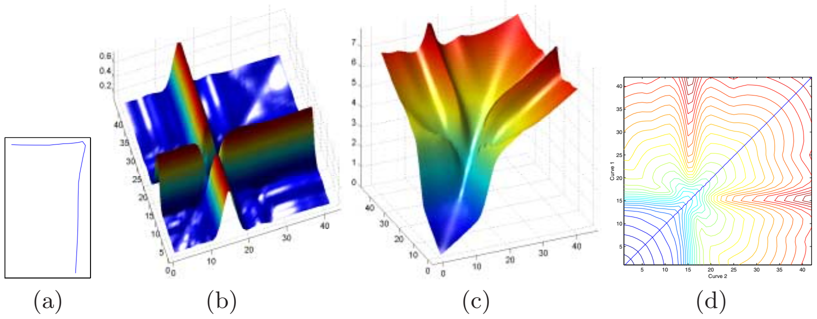


Fig. 2. Comparing a *7*, the shape in (a), to itself using curvature information and $\lambda = 0.111$. The graph in (b) shows the resulting F, (c) shows the reconstructed $T(t,s)$, and (d) - the contour lines of T with the optimal path superimposed

3.1 First Attempt: Curvature-Based Dissimilarity Function

As mentioned in the previous sections, intrinsic properties of curves such as curvature are frequently used in curve alignment algorithms. Let us first study a local dissimilarity function based on curvature:

$$F(t,s) = |\kappa_1(t) - \kappa_2(s)| + \lambda \ . \tag{14}$$

Here, κ_1 and κ_2 are the curvatures of $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively, and $\lambda > 0$. Hence, we have $F(t,s) > 0$ and the Fast Marching Method may be used to solve for $T(t,s)$, setting $T(0,0) = 0$ since the first points of the curves are assumed to match. Notice that such a choice of F resembles somewhat the expression (4) in that it penalizes bending. Stretching is also modelled by the resulting distance measure $d(\cdot\,,\cdot)$, since the more the optimal path $\boldsymbol{c}$ differs from the diagonal line, the higher the value of $T(m,n)$.

Proposition 3. *Let $\mathcal{C}_1(t) = (x_1(t), y_1(t))$, $t \in [0,m]$ and $\mathcal{C}_2(s) = (x_2(s), y_2(s))$, $s \in [0,n]$ be two curves of lengths m and n respectively, each parameterized by arc-length. Let $d(\mathcal{C}_1, \mathcal{C}_2)$ be defined as above. Then, the following hold:*

1. $d(\mathcal{C}_1, \mathcal{C}_1) = 0$,
2. $\mathcal{C}_1 \neq \mathcal{C}_2 \Rightarrow d(\mathcal{C}_1, \mathcal{C}_2) > 0$,
3. $d(\mathcal{C}_1, \mathcal{C}_2) = d(\mathcal{C}_2, \mathcal{C}_1)$.

Proof. To prove *1.*, let $\check{\boldsymbol{c}}(\tau) = (t(\tau), s(\tau)) = (\frac{\sqrt{2}}{2}\tau, \frac{\sqrt{2}}{2}\tau), \tau \in [0, m\sqrt{2}]$. Then, $F(\check{\boldsymbol{c}}(\tau)) = \lambda$, for $\tau \in [0, m\sqrt{2}]$. Let us show that $\check{\boldsymbol{c}}(\tau)$ satisfies Eqn. (10). Since $F(t,s) = \lambda$, on the left we have $\nabla F |\check{\boldsymbol{c}}'(\tau)| = (0,0)$. And since τ is the arc-length parameter of $\check{\boldsymbol{c}}$, we have $|\check{\boldsymbol{c}}'(\tau)| = 1$, and

$$\frac{d}{d\tau}(F(\check{\boldsymbol{c}}(\tau))\boldsymbol{T}(\tau)) = \frac{d}{d\tau}(\lambda\frac{\check{\boldsymbol{c}}'(\tau)}{|\check{\boldsymbol{c}}'(\tau)|}) = \frac{d}{d\tau}(\frac{\sqrt{2}}{2}\lambda, \frac{\sqrt{2}}{2}\lambda) = (0,0) \ .$$

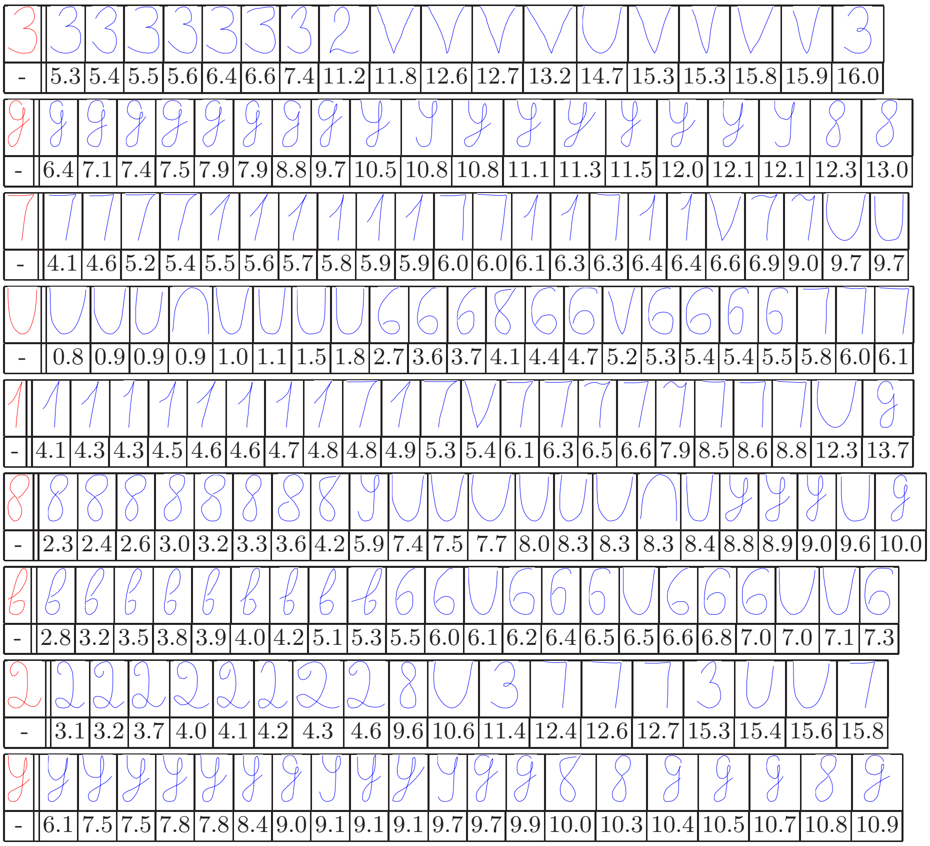


Fig. 3. Best matches out of the 110 curves in the database using the curvature-based dissimilarity function. The number underneath each curve represents its distance from the left-most curve. An average y curve was used as a y-class prototype

Since Eqn. (10) is the EL equation of (12), $\check{\boldsymbol{c}}$ must achieve the minimum in $T(m,m)$. Finally, since the length of $\check{\boldsymbol{c}}$ is $m\sqrt{2}$ we obtain $T(m,m) = \int_{\check{\boldsymbol{c}}} F(\check{\boldsymbol{c}}(\tau))d\tau = \lambda m\sqrt{2}$, which gives *1*. To prove *2*., let $\bar{\boldsymbol{c}}$ achieve the minimum in $T(m,n)$. First, suppose $m = n$. Since $\mathcal{C}_1 \neq \mathcal{C}_2$, $\exists\ \tau$ s.t. $\kappa_1(t(\tau)) \neq \kappa_2(s(\tau))$, which implies that $F(\bar{\boldsymbol{c}}(\tau)) > \lambda$, and the length of $\bar{\boldsymbol{c}}$, $\int_{\bar{\boldsymbol{c}}} d\tau > m\sqrt{2}$, which in turn implies $T(m,m) = \int_{\bar{\boldsymbol{c}}} F(\bar{\boldsymbol{c}}(\tau))d\tau > \lambda m\sqrt{2}$. Property *2*. then follows. If $m \neq n$, then we have $|1 - \min(m,n)/\max(m,n)| > 0$ and hence $d(\mathcal{C}_1, \mathcal{C}_2) > 0$. Property *3*. is true by the symmetry of $F(t,s)$. □

The above properties play an important role in applications such as handwritten character recognition. To illustrate with an example, let us compare a curve that has the shape of a digit *7* with itself (see Fig. 2).

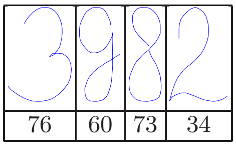


Fig. 4. Curvature test results. The curves shown did not place in their corresponding class tables

Using $\lambda = 0.111$ (see Sect. 3.4 for an intuition on how we chose λ and for more examples), we obtain a surface $T(t,s)$ that has two prominent ridges around otherwise smooth areas. The high-slope regions arise when one of the input curves has points of high curvature, and the other curve has regions of low curvature resulting in great local curvature differences - the high ridges in the graph of F. And since the Eikonal equation $|\nabla T| = F$ implies that the magnitude of the gradient is greater at the places where F is greater, we observe the high slopes on the graph of T in the areas around $t = 15$ and $s = 15$ and relatively low slopes in most other regions. Finally, the gradient descent path of T is a straight line from $(0,0)$ to $(42,42)$ as is expected.

For testing the algorithm, we have used a data-set of 110 sample curves. The curves were all drawn by a single subject with a mouse in an environment that tracked mouse motion and produced uniformly spaced curve points $\mathcal{C}_i = (x_i, y_i)$ to simulate arc-length parametrization. The curvature κ_i at each curve point was computed as follows

$$\kappa_i = \frac{4(y_{i+1} - 2y_i + y_{i-1})\Delta x_i - 4(x_{i+1} - 2x_i + x_{i-1})\Delta y_i}{((\Delta x_i)^2 + (\Delta y_i)^2)^{3/2}} ,$$

where $\Delta x_i = \frac{1}{2}(x_{i+1} - x_{i-1})$, and $\Delta y_i = \frac{1}{2}(y_{i+1} - y_{i-1})$. Note that the above measure tends to infinity as $\Delta x_i \to 0, \Delta y_i \to 0$. Therefore, we bounded κ_i to make sure that it stayed well defined. The obtained curvature vectors were convolved with a one-dimensional Gaussian filter of width 7 in an attempt to lessen the presence of noise in the input data. The parameter λ was chosen automatically per each curve pair (see Sect. 3.4). The prototypes were chosen in a way so as to represent the associated class "best" in terms of the number of class members that place at the top. In some cases, curve averages were used (see Sect. 3.3 for a description of how averages were obtained). Since the average curves' points were not uniformly spaced, a variant of the Fast Marching Method had to be employed that operates on a non-uniformly spaced grid [14]. Figure 3 shows the results of comparing some of the 110 curves in our data-set with the rest of the curves. The 13 best matches are shown together with their corresponding matching distances.

Notice that the results are intuitive. The *3*-shapes are the closest ones to the reference *3*-shape, the *7*'s are the closest ones to the reference *7*, etc. Also, shapes that belong to different classes, but appear similar, such as a *y* and a *g* or a *6* and a *U*, have similar distance measures from the references. It is interesting to note

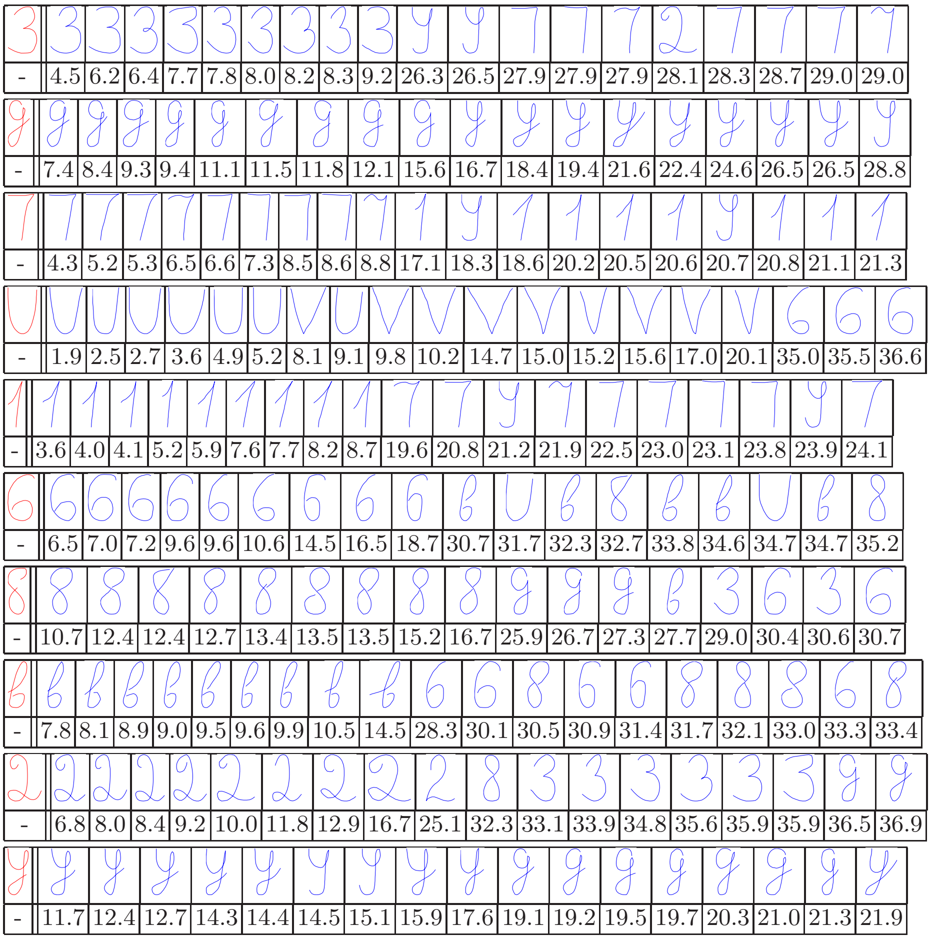


Fig. 5. Best matches using shape contexts. Average *y* and *8* shapes were used as class prototypes

that the upside-down *U*-shape ranks very high among the regular *U*-shapes. This phenomenon is due to the fact that curvature is an intrinsic curve property that is invariant with respect to rotation. The local nature of the curvature measure also causes six of the *1*-shapes to rank closer to a reference *7*-shape than some of the *7*'s whose corner is somewhat smoother than that of the reference *7*. This is due to the fact that curvature was bounded and that the *7* and the *1* sample shapes do not differ much *intrinsically* at other points.

Figure 4 shows the curves that did not place in their class tables. The corresponding ranks are also shown. Locality is also a major cause that misplaces the *3*'s with a loop in the middle (instead of a sharp corner that the others possessed), a *g* with a sharp corner instead of a loop, and a *2* with a sharp corner

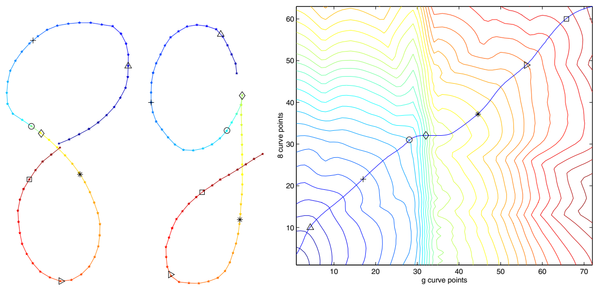


Fig. 6. Some sample curve point correspondences represented by coloring (curve density in black and white production) and by markers. The curve coloring is independent of the contour line coloring on the graph

instead of a loop. The misplaced *8* was drawn in such a way that the endpoints were not matched with the prototype *8*.

3.2 Second Attempt: Shape Contexts to the Rescue

The problems due to locality described above served as motivation to seek a dissimilarity measure that takes into account all the points in the curve instead of being confined to a local region. Shape contexts (see Sect. 1.1), being more descriptive in nature, proved also to be more effective in our experiments.

As we have mentioned, $C(\mathbf{p}_i, \mathbf{q}_j)$ in expression (1) is a local cost of matching the points $\mathbf{p}_i$ and $\mathbf{q}_j$ that lie on the two input shapes. In the case when the two shapes are curves, C may be used in place of F, and the expression (2) resembles (11) in the sense that local dissimilarities are summed. The smoothing parameter λ was still employed and $d(\cdot, \cdot)$ remained otherwise unchanged so that Proposition 3 held.

During shape context computation, point distance was quantized into 5 bins, as in [2], ranging from 0 to the maximum distance between a pair of points on a given curve, thereby achieving invariance to scale. The size of the bins increased exponentially as distance increased in order to give more weight to nearby points. Orientation was quantized into 12 bins.

Figure 5 shows the results of the top 13 experiment performed on the same database of 110 curves using the shape contexts. The first thing to notice is that the problems due to locality are solved. The *3* with a loop instead of a corner places third in the first table on top even though it is compared to a shape that has a corner. The g that was problematic before also places in the g class. Perhaps, most notably, the *2* that has a sharp corner instead of a loop on the second table from the bottom is given a rank of 25.1 which is closer to the *2*-class than the next closest shape, an *8* of rank 32.3. In general, curves from the same class place closer together and are easier to separate from the outliers than before. For example, the *7*'s with curly arms place well within the *7* class. The algorithm has some trouble with rotated shapes. As may be seen on the

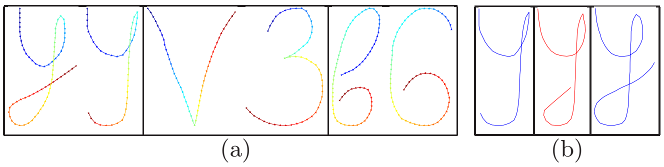


Fig. 7. (a) Sample curve point matchings obtained using curvature differences. The matching points are color coded (or marked by density in black and white production). (b) Averaging two y's. The red shape is the *average* shape. We used a similar average as a y-class prototype in the "top matches" experiments (see Sects. 3.1 and 3.2)

figure, one y is more slanted than the curves in the y class, and that causes it to place 17'th. That was not a problem before, since curvature is a rotation-invariant measure. Belongie et al., however, mention that shape contexts may be computed taking into account the direction of the curve tangents thereby achieving rotation-invariance.

3.3 Curve Point Correspondence

An extra feature that may be obtained once we have computed the distance between the input curves is a correspondence between the curve points. As mentioned in Sect. 2, backtracking along the gradient from the final location (m, n) recovers the optimal path that achieves the minimum in (12). Such gradient descent paths may be viewed as sub-pixel resolution curve point correspondences, since they are continuous paths in (s, t) space, and are analogous to the alignment curves α in (5). Figures 6 and 7(a) show several examples of correspondences obtained using the curvature-based measure. It is interesting to note that the high curvature parts of *V* and *3* match. Also the upper and lower loops of *8* and g that turn in the same directions (first to the left, then to the right) are matched regardless of g's corner. The corner is in a way "skipped" in favor of matching the loops as the descent path levels out to a horizontal slope.

Once a correspondence has been recovered, it can be useful in several ways. For instance, we can form weighted averages between the corresponding point coordinates, resulting in an *average* shape (see Fig. 7(b)), that can, for instance, serve as a representative of a class of shapes. We have used averages in our experiments with a curve database (see Sects. 3.1 and 3.2). On the other hand, if we vary the weights in a sequence going from 0 to 1 using some small predefined step, interpolating the coordinates, we can generate frames of a *morphing sequence*. An example is shown in Fig. 8.

3.4 The Choice of the Smoothing Parameter λ

Both choices of F - either based on curvature or on the shape contexts - contain the smoothing term λ. But how should we choose it? The parameter λ may be

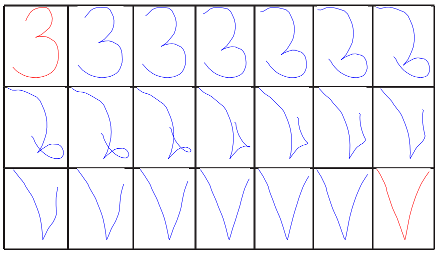


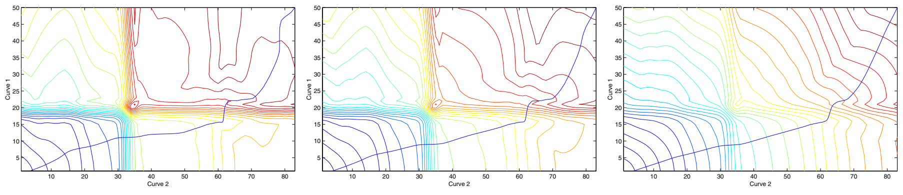
Fig. 8. *Morphing* a *3* into a *V*. The matching was obtained using a curvature-based measure, hence the sharp corners of *3* and *V* match

viewed as a driving force that makes sure that the Fast Marching Method keeps going even when the local dissimilarity term - either the difference of curvatures or the χ^2 test statistic - is zero. Thus λ should not be zero. As mentioned above, changing the parameter λ also has an effect of *smoothing out* the solution. As illustrated on Fig. 9, both the reconstructed surface T and the gradient descent curve are affected by the choice of λ. When λ is low, surface features are more pronounced. Plateaus tend to be flatter, making descent more difficult, and ridges sharpen up. On the other hand, setting λ too high to the point where it dominates the dissimilarity measure, tends to over-smooth the solution making the measure insensitive to the local differences, and leading to a distance map that is essentially a series of concentric circles. In our experiments, we chose λ automatically, driven by the intuition that it should be comparable in magnitude to the dissimilarity term so as neither of the terms would dominate the measure. The average value of F for a given pair of curves $\mathcal{C}_1$ and $\mathcal{C}_2$ of lengths m and n served that purpose relatively well in our tests:

$$\lambda(\mathcal{C}_1, \mathcal{C}_2) = \frac{1}{mn} \int_{\mathcal{C}_1} \int_{\mathcal{C}_2} F(t, s) ds dt \ . \tag{15}$$

3.5 Comparing Closed Curves

The original dynamic programming technique can handle closed curves in a manner similar to the edit distance approach of [9] to comparing cyclic strings. The

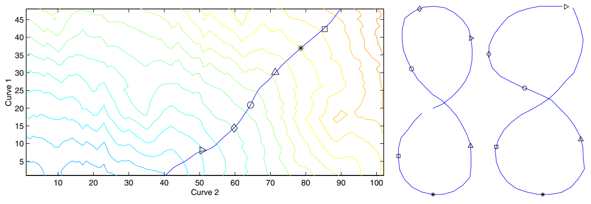


(a) $\lambda = 0.03125$ (b) $\lambda = 0.125$ (c) $\lambda = 0.5$

Fig. 9. The effects of increasing λ on T and on the resulting optimal path

$$\begin{array}{cc} & \mathcal{C}_2 \\ \mathcal{C}_1 & \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \lambda & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \lambda \\ \lambda & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \lambda & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \lambda & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \lambda & \cdot & \cdot \end{pmatrix} \end{array}$$

(a)

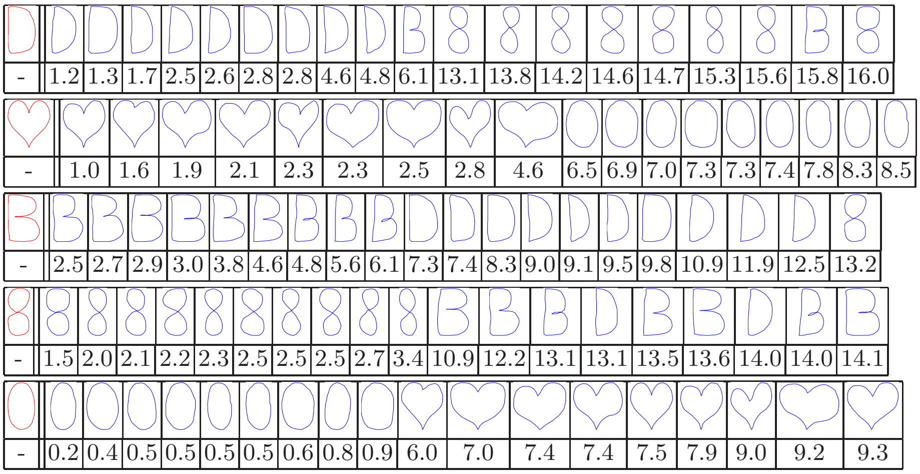


(b) (c) (d)

Fig. 10. (a) The matrix of F values given two curves $\mathcal{C}_1$ and $\mathcal{C}_2$ of six points, where $\mathcal{C}_2$ is $\mathcal{C}_1$ shifted by three points. (b) The matching curve obtained when comparing (c) with (d). Curve (c) points go along the y-axis, and curve (d) points go along the x-axis. Some matching points are shown. The gaps on the curves signify the starting points of curve parametrization

method finds the optimal solution in $O(nm \log m)$ steps. One way of applying the Fast Marching Method to closed curves would be to extend the dynamic programming technique, but that would require developing a version of the Fast Marching Method that restricts the computation to a section of the grid between two given curves and would result in at least a factor of $O(\log m)$ increase of the computation time.

Another approach that is more empirical in nature, but one that does not increase the complexity, is based on the following observation. Suppose we are comparing a closed curve $\mathcal{C}_1$ discretized at six points with a version of itself (denoted $\mathcal{C}_2$) re-parameterized such that its coordinates begin after a shift of three points. Then the matrix F where the rows are points of $\mathcal{C}_1$ and the columns are points of $\mathcal{C}_2$ (see Fig. 10) will have λ values along diagonals in places where



-	1.2	1.3	1.7	2.5	2.6	2.8	2.8	4.6	4.8	6.1	13.1	13.8	14.2	14.6	14.7	15.3	15.6	15.8	16.0	
-	1.0	1.6	1.9	2.1	2.3	2.3	2.5	2.8	4.6	6.5	6.9	7.0	7.3	7.3	7.4	7.8	8.3	8.5		
-	2.5	2.7	2.9	3.0	3.8	4.6	4.8	5.6	6.1	7.3	7.4	8.3	9.0	9.1	9.5	9.8	10.9	11.9	12.5	13.2
-	1.5	2.0	2.1	2.2	2.3	2.5	2.5	2.5	2.7	3.4	10.9	12.2	13.1	13.1	13.5	13.6	14.0	14.0	14.1	
-	0.2	0.4	0.5	0.5	0.5	0.5	0.6	0.8	0.9	6.0	7.0	7.4	7.4	7.5	7.9	9.0	9.2	9.3		

Fig. 11. Best matches of a data-set of closed curves using shape contexts

curve points match. Then, concatenating F with itself horizontally to obtain $\widetilde{F}$ that has 6 rows and 12 columns, which is equivalent to duplicating the points of $\mathcal{C}_2$ along the x-axis produces a diagonal of λ values from entry $(1, 5)$ to the entry $(6, 10)$. Propagating a front from the point $(1, 5)$ according to such an F and then backtracking from $(6, 10)$ would recover the optimal matching curve. Concatenating $\widetilde{F}$ vertically twice to obtain $\overline{F}$ that has 18 rows and 12 columns, propagating a front from $(0, 0)$, and backtracking from (18,12) would produce a path that would include the optimal matching curve from $(1, 5)$ to $(6, 10)$. In order to compute the distance between two closed curves, given such a path, we can re-parameterize $\mathcal{C}_2$ so that both curves start at the same point and run the regular algorithm. Fig. 10 shows an example of a matching curve (b) obtained when comparing two closed shapes (c) and (d). Fig. 11 shows the results of running the above procedure using shape contexts on a smaller data-set of 50 closed curves.

References

[1] N. Ayache and O. Faugeras, "HYPER: A new approach for recognition and positioning of two-dimensional objects," *IEEE Trans. on PAMI*, **8**(1) (1986) 44–54. 36

[2] S. Belongie, J. Malik, and J. Puzicha, "Shape Context: A New Descriptor for Shape Matching and Object Recognition," *NIPS* **13**: (2001) 831–837. 36, 46

[3] R. Basri, L. Costa, D. Geiger, and D.W. Jacobs, "Determining the Similarity of Deformable Shapes," *Vision Research*, **38**: (1998) 2365–2385. 36, 37, 38

[4] I. Cohen, N. Ayachi, and P. Sulger, "Tracking points on deformable objects using curvature information," In *ECCV*, (1992) 458–466. 36

[5] L.D. Cohen and R. Kimmel, "Global minimum for active contour models: a minimal path approach," *IJCV*, **24**(1) (1997) 57–78. 39

[6] E. Hildreth, *The Measurement of Visual Motion*, MIT Press, Cambridge (1983). 36

[7] R. Kimmel, J.A. Sethian, "Fast Marching Methods for Robotic Navigation with Constraints," Report, Univ. of California, Berkley, May (1996). 40

[8] R. Kimmel and J.A. Sethian, "Optimal algorithm for shape from shading and path planning," *Journal of Mathematical Imaging and Vision*, **14**(2) (2001) 237–244. 39

[9] M. Maes, "On a cyclic string-to-string correction problem," *Information Processing Letters*, **35** (1990) 73–78. 36, 48

[10] A. Pentland, and S. Sclaroff, "Closed-Form Solutions for Physically Based Shape Modeling and Recognition," *IEEE Trans. on PAMI*, **13**(7) (1991) 715–729. 36

[11] A. Pope, and D. Lowe, "Learning Object Recognition Models from Images," *ICCV* (1993) 296–301. 36

[12] T. Sebastian, P. Klein, B. Kimia, "On aligning curves," *IEEE Trans. on PAMI*, **25**(1) (2003) 116–124. 36, 38

[13] B. Serra and M. Berthod, "Optimal subpixel matching of contour chains and segments," *IJCV* (1995) 402–407. 38

[14] J. Sethian, *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision and Materials Sciences*, Cambridge Univ. Press (1996). 38, 41, 44

[15] J. A. Sethian, "A fast marching level set method for monotonically advancing fronts," *Proc. Nat. Acad. Sci.*, **93**(4) (1996) 1591–1595. 39, 40

[16] H. D. Tagare, "Shape-based nonrigid correspondence with application to heart motion analysis," *IEEE Trans. Medical Imaging*, **18**(7) (1999) 570–578. 36

[17] C. Tappert, "Cursive script recognition by elastic matching," *IBM Journal of Research Development*, **26**(6) (1982) 765–771. 36

[18] R. Wagner, and M. Fischer, "The string-to-string correction problem," *Journal of the Association for Computing Machinery*, **21** (1974) 168–173. 35, 38