PROFI

Project number:	FP6-511572
Project acronym:	PROFI
Title:	Perceptually-relevant Retrieval Of Figurative Images

Deliverable No: D4.3: Algorithms to match sets of curves

Short description:
Randomized algorithms are developed for finding similarities between two shapes A and B. Shapes are modelled by sets of line segments or polygonal curves. The major idea is to take random samples of points from both shapes and give a "vote" for that transformation (translation, rigid motion, or similarity) matching one sample with the other. If that experiment is repeated frequently we obtain by the votes, a certain distribution of points in the space of transformations. Clusters of this point set indicate which transformations give the best match between the two figures.

Due month:	12
Delivery month:	12
Lead partner:	FUB
Partners contributed:	FUB, UU
Classification:	Public

Information Society Technologies

Project funded by the European Community under the "Information Society Technologies" Programme

1 Introduction

In this work package we develop algorithms for *matching* two planar shapes. We assume that shapes are modeled by sets of plane polygonal curves (or sets of line segments). As possible classes of transformations we will consider translations, rigid motions (i.e., translations and rotations) and similarities (i.e., translations, rotations and scalings).

The general situation is that we are given two objects A and B and a set T of allowable transformations and we want to transform B optimally so that the transformed image of B is as close to A as possible. Usually the quality of match is measured by some distance function or similarity measure $\delta(A, B)$ which assigns a number to any pair of objects A and B. We call the problem of optimal matching of the complete shape B to the complete shape A a *complete-complete matching* (CCM).

Several similarity measures and algorithms are known to match two curves, especially polygonal curves, see [15, 23] for surveys. One of the "universal" similarity measures is the Hausdorff distance which is defined for any two compact sets A and B. It assigns to each point of one set the distance to its closest point on the other and takes the maximum over all these values. For arbitrary sets of n line segments the Hausdorff distance can be computed in $O(n \log n)$ time and the matching problem under translations and rigid motions can be solved in polynomial time [3, 4]. Being a maximum metric, the Hausdorff-distance is very sensitive to noise: a single outlier can determine the distance value, thus two otherwise similar shapes can have a large Hausdorff distance. There are approaches to overcome this drawback, such as percentile-based Hausdorff distance and mean Hausdorff distance [15]. On the other hand, there are examples where two less similar shapes get a small dissimilarity value. A few similarity measures are defined for pairs of curves, which capture the course of a curve, or the relative course of two curves: Fréchet distance [4], turning function distance [6], and dynamic time warping distance [12, 19]. There are no generalizations of those distances to sets of curves, though [5] gives a generalization of the Fréchet distance to geometric graphs. A similarity measure which is designed for sets of curves is the reflection visibility distance [14]. It is based on the visibility complex at every point in the plane. The reflection visibility distance is robust against different kinds of disturbances but is expensive to compute.

Our objective is to develop an algorithm which comes close to human similarity perception and allows an efficient implementation for the retrieval system. The method we introduce here attempts to capture an intuitive notion of "matching", i.e. we find one or more candidates for the best transformations, that when applied to the shape B maps the most similar parts of the shapes to each other.

2 Results

2.1 Probabilistic matching of sets of line segments

2.1.1 Translations

Given two sets $A, B \subset \mathbb{R}^2$ consisting of all points lying on a finite set of polygonal curves (or, equivalently, line segments). Find that translation t, which lets the translated image of B, $t(B)$, *match best* A, i.e. maps the most similar parts of the shapes A and B to each other.

Since a set of line segments is actually a one dimensional domain, we can define a uniform random distribution on this domain in a similar way as a uniform random distribution on an

interval of real numbers. Selecting a random point under uniform distribution can be done in a following way: select a random segment from the set, where the probabilities for the segments are weighted with the segment lengths; then select uniformly a random point on that segment. The algorithm is quite simple:

1. Take a random point a in A and a random point b in B under uniform distribution, and give one "vote" to the translation $t = a - b$.

2. Repeat this experiment many times. Then the distribution of votes in the (two-dimensional) translation space T approximates a certain probability distribution.

3. For a given neighborhood size δ take the points of T with the highest number of votes in their δ-neighborhood as candidates for good translations.

The idea behind this algorithm, is that the translations, which map large parts of shapes to each other should get significantly more votes than others. The size of the sampling neighborhood δ influences the quality of the match, the role of δ is explained more detailed in section 2.1.1.

In order to find a translation with the highest number of votes, or the sampling points, in its neighborhood, we consider the arrangement of the δ-neighborhoods of the sampling points. The basic observation is that if a sampling point s is contained in the δ-neighborhood of a translation t, then t is also contained in the δ-neighborhood of s. All translation vectors in the same cell of the arrangement have the same sampling points in their δ-neighborhoods. Therefore it is sufficient to traverse the arrangement and take the nodes with the highest number of sampling points whose δ-neighborhoods contain this node.

Probability distribution in translation space. Here we analyze the probability distribution in the translation space sampled by the algorithm above. The set of translations considered by our algorithm is $T = \{a - b \in \mathbb{R}^2 : a \in A,\ b \in B\}$, which corresponds to the Minkowski sum $A \oplus (-B)$. Before considering the general situation, let us take a closer look at two simple cases, where the sets A and B contain each a single straight line segment $s_a = \overline{a_1 a_2}$ and $s_b = \overline{b_1 b_2}$ respectively. Let l_a and l_b denote the lengths of the segments.

Case 1: The segments s_a and s_b are not parallel. Let α denote the absolute value of the acute angle between the straight lines supporting s_a and s_b. Then the translations sampled by our algorithm, T, are located within a parallelogram spanned by the points $a_1 - b_1,\ a_1 - b_2,\ a_2 - b_1,\ a_2 - b_2$ (s. Figure 1). Any point $a \in s_a$ can be represented as a linear combination of points a_1, a_2, $a = (1 - \lambda)a_1 + \lambda a_2 = a_1 + \lambda(a_2 - a_1)$, with $\lambda \in [0, 1]$, and a point $b \in s_b$ as a linear combination of points b_1, b_2, $b = b_1 + \mu(b_2 - b_1)$, with $\mu \in [0, 1]$. The corresponding translation is then $a - b = \lambda(a_2 - a_1) - \mu(b_2 - b_1) + a_1 - b_1$, which can be written as $\lambda((a_2 - b_1) - (a_1 - b_1)) + \mu((a_1 - b_2) - (a_1 - b_1)) + (a_1 - b_1)$. The latter expression is a linear combination of the vectors $(a_2 - b_1) - (a_1 - b_1)$ and $(a_1 - b_2) - (a_1 - b_1)$ translated by the vector $(a_1 - b_1)$, which characterizes exactly the points within the parallelogram described above.

According to our algorithm any pair of points $a \in A$ and $b \in B$ has equal chances to get selected within one random experiment, and since the segments are not parallel and thus have at most one intersection point for any translation of s_b, every pair contributes a unique translation $a - b$ to our sample. Let R be a region of the translation plane. From the considerations above follows: If R has no intersection with T, the probability $P(R)$ of R is 0,

and any region $R \subset T$ has the probability proportional to its area. In general, the probability of $R \subset \mathbb{R}^2$ is proportional to the area of $R \cap T$. This yields a uniform distribution on T. Knowing that the area of T equals $l_a l_b \sin\alpha$ and that the total probability of T should be 1, we get the following density function:

$$f(t) = \begin{cases} \frac{1}{l_a l_b \sin\alpha} & t \in T \\ 0 & t \in \mathbb{R}^2 \setminus T \end{cases}.$$

Figure 1 illustrates this case.

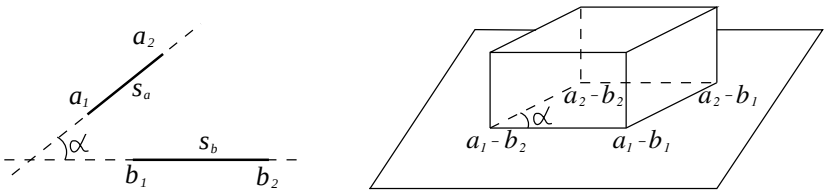


Figure 1: Density function of the probability distribution on the translation space for two non parallel line segments

Case 2: The segments s_a and s_b are parallel. Then the four corner points of the parallelogram T, described in the previous case, are collinear, and T is thus a straight line segment. W.l.o.g. let T have $a_1 - b_2$ and $a_2 - b_1$ as end points, and let $l_a < l_b$ (s. Figure 2). The other two important points are then $a_2 - b_2$ and $a_1 - b_1$. Let $p_1 = a_1 - b_2$, $p_2 = a_2 - b_2$, $p_3 = a_1 - b_1$, and $p_4 = a_2 - b_1$. As one can easily see, the length of the segment T is $l_a + l_b$ and the distance between the points p_2 and p_3 is $l_b - l_a$. Now any translated image of s_b has either no intersection with s_a, then the corresponding translation would never be generated by our algorithm, or the intersection is a segment, then there are many pairs "voting" for the corresponding translation. The number of those pairs is proportional to the length of the intersection segment. Note that, when a translation $t \in \overline{p_2 p_3}$ is applied to s_b, the intersection of the image of s_b with s_a is the whole segment s_a. This implies that the translations in $\overline{p_2 p_3}$ have equal chances to get selected by a random experiment, i.e. we have uniform distribution on $\overline{p_2 p_3}$. As t moves from p_2 to p_1 or from p_3 to p_4 along T, the length of the corresponding intersection segment decreases linearly. Further analysis leads to the following one-dimensional density function:

$$f(t) = \begin{cases} \frac{\|t-p_1\|}{l_a l_b} & \text{if } t = (1-\lambda)p_1 + \lambda p_2 \text{ for } \lambda \in [0,1] \\ \frac{1}{l_b} & \text{if } t = (1-\lambda)p_2 + \lambda p_3 \text{ for } \lambda \in [0,1] \\ \frac{\|t-p_4\|}{l_a l_b} & \text{if } t = (1-\lambda)p_3 + \lambda p_4 \text{ for } \lambda \in [0,1] \\ 0 & \text{if } t \in \mathbb{R}^2 \setminus T \end{cases}.$$

See Figure 2 for an illustration. The probability distribution $P(R)$ for a region R of the translation plane is then the value of the curve integral of the density function $f(t)$ on the segment $T \cap R$.

In the general situation A is a set of straight line segments $s_{a1}, \ldots, s_{an}$ and B is a set of segments $s_{b1}, \ldots, s_{bm}$ with $|s_{ai}| = l_{ai}, i = 1..n$, $|s_{bj}| = l_{bj}, j = 1..m$ and $|A| = \sum_{i=1}^{n} l_{ai} = l_a$, $|B| = \sum_{j=1}^{m} l_{bj} = l_b$, and α_{ij} is the absolute value of the acute angle between the lines supporting segments s_{ai}, s_{bj}. Let T_{ij} denote the sets of translations induced by the segments s_{ai}, s_{bj}, clearly, $T_{ij} = s_{ai} \oplus (-s_{bj})$. Let P_{ij} denote the probability distributions corresponding to the pairs of segments s_{ai}, s_{bj} defined as in case one or two described above, depending

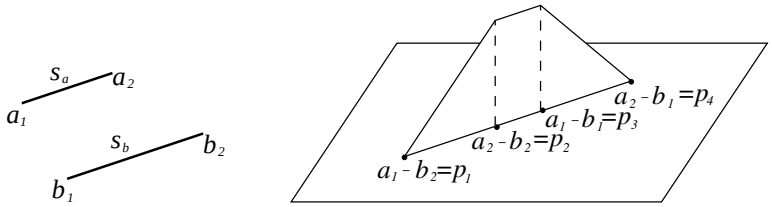


Figure 2: Density function of the probability distribution on the translation space for two parallel line segments

on which is applicable. The portion of point pairs generated by the algorithm which are contributed by the segments s_{ai}, s_{bj} is $\frac{l_{ai}l_{bj}}{l_a l_b}$. Thus, if R is a region of the translation plane the probability of R is $\sum_{i=1}^{n}\sum_{j=1}^{m}\frac{l_{ai}l_{bj}}{l_a l_b}P_{ij}(R)$. Figure 3 shows the density function for sets A and B each containing two line segments. The superimposition of the two sets in the bottom of Figure 3 results from a translation corresponding to the region with the highest density.

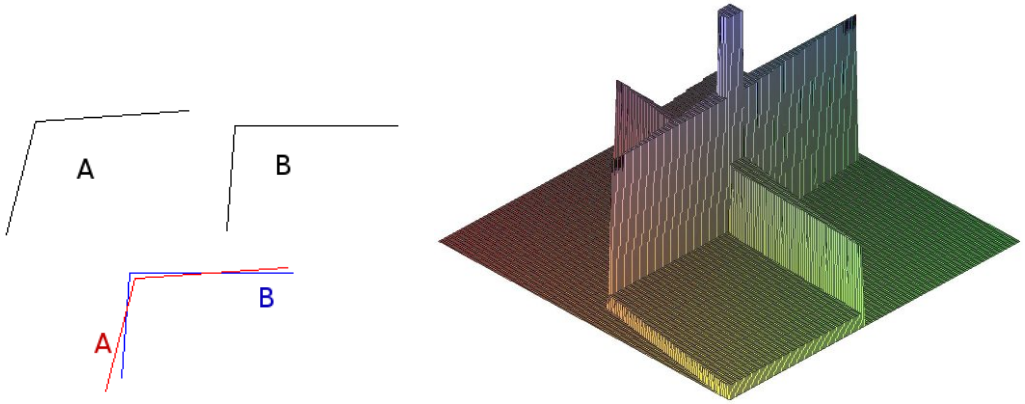


Figure 3: Probability distribution on the translation space for two sets of line segments

As one can see, the density function of the probability distribution in translation space has support $T = \bigcup_{i=1}^{n}\bigcup_{j=1}^{m} T_{ij}$, which is a union of $n \cdot m$ parallelograms or line segments. These parallelograms and line segments form an arrangement in the translation plane. Within each cell of the arrangement the value of the density function is the weighted sum of the density functions corresponding to the parallelograms or segments containing the cell. This arrangement can have a complexity of $\Theta(n^2m^2)$.

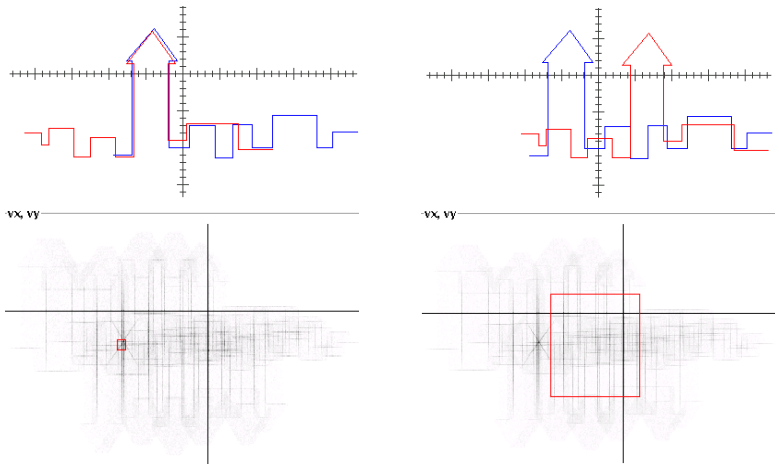
So, we could compute T and the values of the density function in each cell exactly, but because of the high complexity we approximate it with our algorithm described before.

Analysis of the algorithm. $T = A \oplus (-B)$ is the set of possible translations sampled by our algorithm. For a translation $t \in T$ let $U_\delta(t) = \left\{x \in \mathbb{R}^2 : \|x - t\| \leq \delta\right\}$ denote the δ-neighborhood of t.

Let S be a random sample of T from the experiment, $|S| = N \in \mathbb{N}$ and $a(t)$ denote the number of sample points within the δ-neighborhood of a translation t. Let $p(t)$ denote the probability of $U_\delta(t)$ and the random variable $\tilde{p}(t) = a(t)/N$ which is an estimate of $p(t)$.

What does the value $p(t)$ mean for the shapes A and B and the translation t? Consider a subset $M(t)$ of $A \times B$, defined as $M(t) = \{(a, b) : a \in A, b \in B, \|a - (b + t)\| \leq \delta\}$, that is a set of pairs of points $a \in A$ and $b \in B$ which are mapped into the δ-neighborhood of each other if B is translated by t. As the detailed analysis shows, maximizing $p(t)$ we maximize the measure of the set $M(t)$ within $A \times B$. For the most cases it also means, that the translation t_0 maximizing $p(t)$ brings the largest fitting parts of A and B into the δ-neighborhood of each other.

The choice of δ thus controls the trade-off between the quality of match and the size of the parts matched. With a small value of δ our algorithm would find a translation which maps nearly congruent parts of two shapes to each other, see Figure 4(a). A large value of δ leads to a translation which gives a rough match but for larger parts of the shapes, see Figure 4(b).



(a) (b)

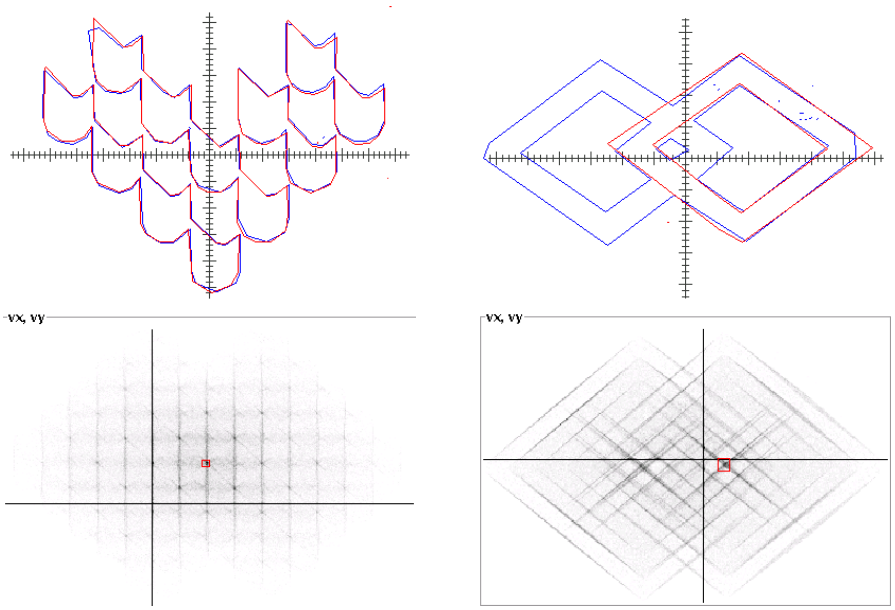
Figure 4: Matching with (a) small grid size and (b) large grid size

For nearly congruent figures small neighborhood size already leads to a complete-complete matching, see Figure 5(a), and with the same value for δ we can get complete-partial matching, see Figure 5(b). So δ does not alone determine what kind of matching we get or how large the matched parts are. We are working on a valuation function that would award matched parts and in case of complete-complete matching give penalties for the "outliers".

In the following we give some error bounds for the distribution approximation. From these error bound theorems we can derive the bounds on the number of the experiments needed to get a good estimate with high probability. Using Chernoff bounds (see Section 4.1 in [18]) we can bound the absolute and relative errors for the estimate:

Theorem 2.1 (Absolute error)**.** *Given two shapes modeled by sets of line segments of complexity n in the plane and parameter ε, η, $0 \leq \varepsilon, \eta \leq 1$. In time $O(n + \frac{\log \frac{1}{\eta} \log n}{\varepsilon^2} + \frac{\log^2 \frac{1}{\eta}}{\varepsilon^4})$ we can compute a translation t_{app} such that $|\tilde{p}(t_{\text{app}}) - p(t_{\text{opt}})| \leq \varepsilon$ with probability at least $1 - \eta$, where t_{opt} is a translation maximizing $p(t)$.*

The number of sample translation vectors sufficient to get an approximation error at most ε with probability at least $1 - \eta$ is then $N = \frac{16 \ln \frac{1}{\eta}}{\varepsilon^2} + 2$.



(a) (b)

Figure 5: Matching with a sampling neighborhood of the the same size in translation space for different shapes

Theorem 2.2 (Relative error)**.** *Given two shapes A and B modeled by sets of line segments in the plane of complexity n and of diameter D_A and D_B respectively, and parameter ε, η, $0 \leq \varepsilon, \eta \leq 1$. In time $O(n + \frac{\log \frac{1}{\eta}(D_A+D_B)^2 \log n}{\varepsilon^2\delta^2} + \frac{\log^2 \frac{1}{\eta}(D_A+D_B)^4}{\varepsilon^4\delta^4})$ we can compute a translation t_{app} such that $|\tilde{p}(t_{\text{app}}) - p(t_{\text{opt}})| \leq \varepsilon p(t_{\text{opt}})$ with probability at least $1 - \eta$.*

In order to get a relative approximation error of at most ε with probability at least $1 - \eta$ it is sufficient to take $N = \frac{32(D_A+D_B)^2 \ln \frac{2}{\eta}}{\varepsilon^2\delta^2} + 2$ sample translation vectors.

The proofs of the theorems are given in appendix A.

2.1.2 Rigid motions

Let again A and B be finite sets of polygonal curves in $\mathbb{R}^2$ and the set of allowed transformations the three-dimensional space T of rigid motions, i.e. translations and rotations. A rigid motion $t = (\alpha, v_x, v_y)$ is defined by a rotation angle α and a translation $v = (v_x, v_y)$ and maps a point $b \in \mathbb{R}^2$ to a point $t(b) = Mb + v$, where $M = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}$ is the rotation matrix.

Generating a random rigid motion bears some problems: whereas for two points a and b there exists a unique translation, which maps the point b to the point a, there are infinitely many rotation and translation combinations, under which the image of the point b equals point a. A pair of points a_1, a_2, and a pair of points b_1, b_2 define a unique similarity transformation mapping b_1 to a_1 and b_2 to a_2, see section 2.1.3, but it is not clear what rigid motion should be assigned to the pairs of points, since in general there is no rigid motion mapping b_1 to a_1 and b_2 to a_2.

If, however, together with each point $a \in A$ we have some unit length direction vector d_a

(e.g. direction of a tangent line) and for each $b \in B$ a direction vector d_b, then there exists a unique rigid motion $t = (\alpha, v_x, v_y)$, such that $t(d_b)$ is parallel to d_a and $t(b) = a$. t is a solution to the following linear system of equations: $Md_b = d_a$ and $Mb + v = a$, where M is the rotation matrix as defined above and v is a translation vector. So we could take a random point a in set A and the direction of the tangent line d_a at this point, and a random point b in set B with the direction d_b and give a vote to a rigid motion defined by this point-direction pair. But since we are dealing with line segments (or polygonal curves) we can only get a finite number of rotation angles and for some cases miss the most suitable angle. For two curves in Figure 6 using the direction of a tangent line or the slope of a segment we would only get two possible rotation angles: $\frac{\pi}{4}$ and $-\frac{\pi}{4}$, though the best rotation angle would be 0.

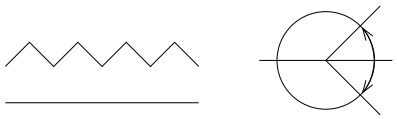


Figure 6: Matching under rigid motions: Difference between slopes of the segments does not necessarily "vote" for the optimal rotation angle.

We also considered taking a random rotation and a translation determined by a pair of points, randomly selected from the sets A and B, but then we would sample unnecessarily many transformations. A strategy that seems to give best results, is to select a random point a from set A, and take another point a' from the curve containing point a, which is a fixed curve-distance away from the first one, direction on the curve can be chosen randomly or be fixed for each curve by the order of its corner points. If there is no point a' on the curve with that fixed curve-distance, then take the end point of the curve as a'. Take the vector $d_a = a' - a$ as the direction vector for point a. Generate a random point-vector pair $b \in B$, d_b in a similar way. The direction vectors d_a and d_b are then scaled to a unit lengths. These two point-vector pairs determine a unique rigid motion as described above. This approach overcomes the problem with the limited number of tangent slopes.

The matching algorithm for rigid motions is similar to that for the translations:

1. Take a random point-vector pair $a \in A$, d_a (as described above) and a random point-vector pair $b \in B$, d_b under uniform distribution, and give one "vote" to the rigid motion t, which maps the vector d_b to the vector d_a and the point b to point a.

2. Repeat this experiment many times. This approximates a probability distribution in a three-dimensional space T of rigid motions.

3. For a given neighborhood size δ take the points of T with the highest number of sampling points in their δ-neighborhood (δ-neighborhood of a rigid motion is explained later) as candidates for good transformations.

The idea behind it is, again, that "good" rigid motions, i.e. those, that when applied to the set B, map the best matching parts of shapes A and B to each other, should get more votes, than the others. As before, the choice of δ determines how similar the matched parts of the shapes ought to be.

Let B_α denote the set B rotated by an angle α about the origin, and b_α a point b rotated by α. For a fixed angle α the subspace in the remaining two dimensions, i.e. the space of

translations, which is sampled by our algorithm, is exactly $A \oplus (-B_\alpha)$, as we have argued in the previous section. Then the probability distribution approximated by the algorithm has support $T = \{t = (\alpha, v_x, v_y) \in \mathbb{R}^3 : \alpha \in [0, 2\pi] \text{ and } \exists a \in A, b \in B \text{ s.t. } a - b_\alpha = (v_x, v_y)\}$.

We define the δ-neighborhood of a rigid motion t as the set of rigid motions under which the image of any point $b \in B$ is not more than δ away from its image under transformation t: $U_\delta(t) = \{t' \in \mathbb{R}^3 : \forall b \in B \, \|t(b) - t'(b)\| \leq \delta\}$.

If a point b has distance r to the origin, then the rotation by angle α causes the displacement $d_1 = r\sqrt{2(1 - \cos\alpha)}$ by the law of cosines, see Figure 7. The total distance between a

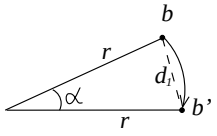


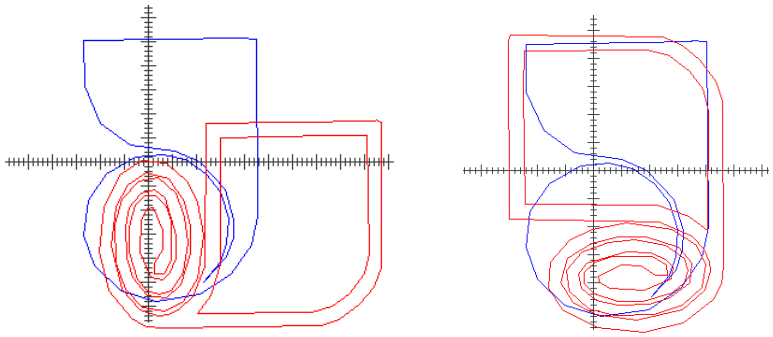
Figure 7: Displacement of a point b after a rotation by angle α

point with distance r to the origin and its image after applying a rotation α and translation v is at most $r\sqrt{2(1 - \cos\alpha)} + \|v\|$.

Let D_A and D_B denote the diameters of the sets A and B respectively. W.l.o.g. let the center of a bounding box of B be the origin, then the distance from any point in B to the origin is at most $r = D_B/2$. For practical usage we can restrict a δ-neighborhood of a rigid motion $t = (\alpha, v_x, v_y)$ to a set $V_\delta(t) \subset U_\delta(t)$ defined as

$V_\delta(t) = \left\{t' = (\alpha', v'_x, v'_y) \in \mathbb{R}^3 : r\sqrt{2(1 - \cos(\alpha' - \alpha))} + \left\|(v'_x, v'_y) - (v_x, v_y)\right\| \leq \delta\right\}$.

The number of votes a δ-neighborhood of a rigid motion t gets is proportional to the number of point pairs $a \in A$, $b \in B$ brought into the δ-neighborhood of each other by the transformation t applied to the set B.



(a) (b)

Figure 8: Matching under rigid motions: (a) original position, (b) matching produced by the algorithm

2.1.3 Similarities

Given two finite sets A, B of polygonal curves (or line segments) in $\mathbb{R}^2$. The set of allowed transformations is now the four-dimensional space of similarities, i.e. translations, rotations

and scalings. A similarity map $t = (\alpha, k, v_x, v_y)$ is defined by a rotation angle α, scaling factor k and a translation vector $v = (v_x, v_y)$. t maps a point $b \in \mathbb{R}^2$ to a point $t(b) = Mb + v$, where $M = \begin{pmatrix} k\cos\alpha & -k\sin\alpha \\ k\sin\alpha & k\cos\alpha \end{pmatrix}$. Two points a_1, a_2 in A, and two points b_1, b_2 in B, determine a unique similarity transformation t mapping b_1 to a_1 and b_2 to a_2. t is a solution to the following system of liner equations: $Mb_1 + v = a_1$ and $Mb_2 + v = a_2$, where M and v are defined as above. The matching algorithm is now:

1. Take two random points a_1, a_2 in A and two random points b_1, b_2 in B under uniform distribution, and give one "vote" to the similarity map t, which maps b_1 to a_1 and b_2 to a_2 as described above.

2. Repeat this experiment many times, yielding an approximation of a certain probability distribution in a four-dimensional space T of similarity maps.

3. For a given neighborhood size δ take the points of T with the highest number of sampling points in their δ-neighborhood (as defined below) as candidates for good transformations.

Again, we expect "good" similarity maps to receive more votes, than the ones that do not result in a good match. The role of the size of the sampling neighborhood is similar to that for the translations: δ determines how close, i.e. how similar, the shapes or parts of the shapes should be in order to get matched.

The set of possible similarity maps, which are sampled by the algorithm is
$T = \{t \in \mathbb{R}^4 : \exists a_1, a_2 \in A, b_1, b_2 \in B \text{ such that } t(b_1) = a_1 \text{ and } t(b_2) = a_2\}$.

We have to be careful defining a δ-neighborhood of a similarity map t: we want only those transformations t' to belong to the δ-neighborhood of t, under which the image of any point in B has at most distance δ to its image under transformation t:
$U_\delta(t) = \{t' \in \mathbb{R}^4 : \forall b \in B \, \|t(b) - t'(b)\| \leq \delta\}$.

Let us consider the displacement of a point b introduced by each of the individual transformations that make up a similarity map: rotation, scaling and transformation. As we have argued in previous section, the displacement of a point b with distance r to the origin, under rotation by angle α is $d_1 = r\sqrt{2(1-\cos\alpha)}$. Scaling by factor k moves a point b $d_2 = r(k-1)$ away from its original position if $k \geq 1$, and $d_2 = r(1-k)$, if $k < 1$, see Figure 9(a). The joint displacement introduced by rotation α and scaling k can be expressed using the law of cosines as $d_3(\alpha, k, r) = r\sqrt{1 + k^2 - 2k\cos\alpha}$, see Figure 9(b). The total displacement of a point b with distance r to the origin, caused by a similarity map $t = (\alpha, k, x, y)$ is then $d \leq d_3(\alpha, k, r) + \|v\|$, where $v = (x, y)$ is the translation vector.

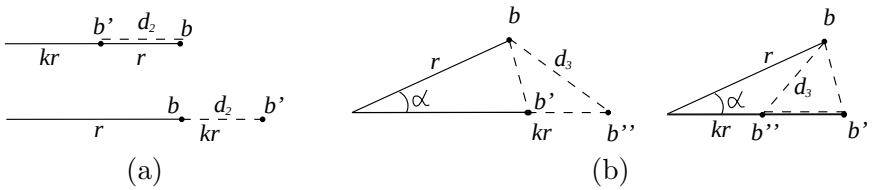
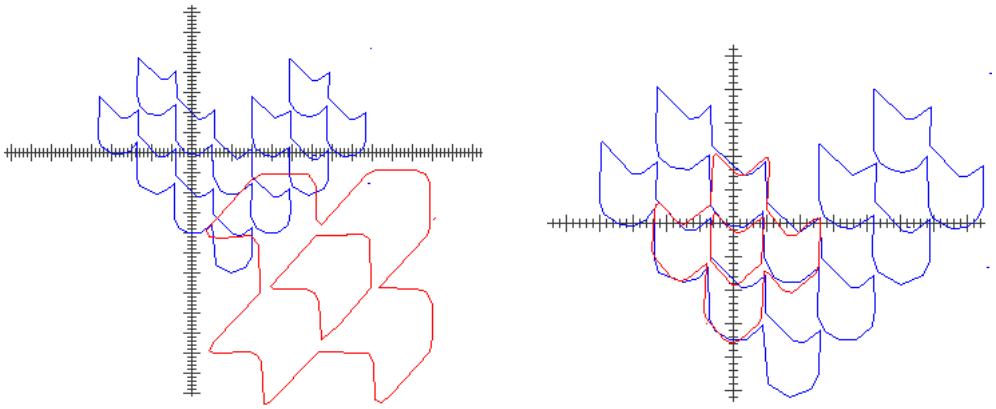


Figure 9: Displacement of a point b after (a) scaling by factor k, (b) rotation by angle α and scaling by factor k

If we first translate the set B, so that the center of a bounding box of B is placed at the origin, then the largest possible distance of a point in B to the origin is $r = D_B/2$, where D_B is the diameter of the set B. Now we can restrict a δ-neighborhood of a similarity map $t = (\alpha, k, v_x, v_y)$ for practical use to a set $V_\delta(t) \subset U_\delta(t)$:
$V_\delta(t) = \left\{ t' = (\alpha', k', v'_x, v'_y) \in \mathbb{R}^4 : r\sqrt{1 + (k'/k)^2 - 2(k'/k)\cos(\alpha' - \alpha)} + \|v' - v\| \leq \delta \right\}$.

In our preliminary implementation we sample T by placing a 4d-grid on it and counting the votes within each grid cell, then we take the centers of the grid cells with the highest number of votes as candidates for a good similarity map. For simplicity, we choose scale and grid size for each dimension in such a way that a change in any direction separately would cause a displacement at most δ. Therefore, the two dimensions, corresponding to the translation vector have grid size δ. From the above displacement calculations follows: the grid size of the rotation dimension is $\arccos(1 - \delta^2/2r^2)$ and the values of α range from 0 to 2π. For the scaling factor: axis has logarithmic scale to the base $s_k = 1 + \delta/r$ and two scaling factors $k_l < k_r \in \mathbb{R}^+$ bound a cell in the scaling dimension if $\log_{s_k} k_l$ and $\log_{s_k} k_r$ are integers and $\frac{k_r}{k_l} = s_k$.



(a) (b)

Figure 10: Matching under similarities: (a) original position, (b) matching produced by the algorithm

We can say that the similarity maps with many votes in their δ-neighborhood, if applied to the set B, bring more pairs of points $b_1, b_2 \in B$ into the δ-neighborhood of A than the transformations with few votes.

2.2 Probabilistic matching of sets of polygonal curves

We are also working on heuristics, inorder to speed up the matching process, that is to find good transformation candidates with less experiments. Here we describe an algorithm which during one random experiment computes a sequence of transformations matching finite ordered sets of points from one shape to finite ordered sets of points from the other shape. The quality of the match between two finite ordered point sets is measured by the weighted sum of quadratic distances between the corresponding points. The resulting preliminary transformation is weighted with the quality of match and the size of the matched point sets. Then we get a weighted sample of the transformation space, where the neighborhoods with large

weight are likely to contain candidates for transformations resulting in a good match for the shapes. The idea behind it, is that a transformation which gives a good match for the shapes, would give a good match for larger sets of points on these shapes. The details on selecting the point sets and computing the candidate transformations are presented in the following.

Input are two sets S_1, S_2 of planar polylines and a transformation t is searched for, that best maps S_1 to S_2. The classes of allowable transformations considered here are translations, homotheties (scaling + translation), rigid motions (rotation + translation), similarities (rotation + scaling + translation), and affine maps.

The problem of computing preliminary transformations consists of two subproblems: one is to find correspondences between features, the other is to find a transformation that maps the corresponding features to each other. If the correspondences are known, the transformation is easy to compute – if the transformation is known the correspondences (if properly defined) are easy to compute. But in the present case both are unknown.
In many approaches a minimum sample of features is used to compute the model parameters (transformation in this case). When the random sample consensus (RANSAC) [17] is applied these model parameters are validated with the complete data in a second step. In the pose clustering approach [22] every such sample gives a vote and the set of all possible votes is evaluated.
Here we use another approach: iteratively the sample is extended and the model is updated until the whole data is part of the sample, or until sample and model are not longer consistent.

2.2.1 Finding Correspondences

Conspicuous features of curves arise from regions of high curvature [7] – regarding polylines these regions are the vertices. But not every vertex, even though its turning angle may be great, yields a feature recognizable by a human observer. For this reason, the mapping algorithm trys to find corresponding vertices but also may match a vertex without corresponding peer to a point lying on a line segment.

The initial part of every vote is the random choice of a single vertex of the two sets of polylines each, and the direction for traversing the list of following verteces (also both possible directions may be processed). Starting from that pair a sequence of sets of pairs of verteces and vertex surrogates is generated. For every element of that sequence a corresponding transformation is computed. The transformations belonging to a sequence are compared. The one rated best gets an additional weight and is handed over to the clustering algorithm as a preliminary transformation.

Let p_0 be the randomly chosen vertex from the first set S_1 and let $p_1, \ldots, p_k$ be the successional verteces with respect to the randomly chosen direction. Analogous let q_0 be the randomly chosen vertex from the second set S_2 and let $q_1, \ldots, q_l$ be the successional verteces. The pair (p_0, q_0) is added to the – so far empty – sample set s.
For (p', q') being the last pair of points added to s, the distances to the successional vertex p_i resp. q_j is computed as $d_p = \|p_i - p'\|$ resp. $d_q = \|q_j - q'\|$.
If d_p and d_q are nearly equal i.e. if $|d_p - d_q| \leq \min(d_p, d_q) \cdot a$ with a being a constant called the alignment threshold, then the two verteces p_i and q_j are taken as a corresponding pair which is added to the sample set s.

Otherwise a vertex surrogate is created. A surrogate is a point lying on an edge of the polyline, but nevertheless is treated like a vertex. It is chosen to have the same distance to its predecessor as the corresponding two verteces of the other polyline have. In the case of $d_p < d_q$ the surrogate is the point $q_i' = q' \cdot (1 - d_p/d_q) + q_j \cdot (d_p/d_q)$ and the pair (p_i, q_i') is added to s. The case $d_q < d_p$ is handled analogously.

When the end of a polyline is reached, then starting from the initial pair the traversal is performed in the other direction.

2.2.2 Calculating the Transformations

For every new pair of verteces or vertex surrogates added to s a transformation is computed based on a least squares approach.
The easiest way would be to compute the transformation $t' \in T$ that minimizes the sum of the squared distances $\sum_{(p_i,q_i)\in s} \|q_i - t(p_i)\|^2$ of the verteces. This would favour parts with many verteces over parts with less verteces regardless of the extent and the expressiveness.
For that reason every pair (p, q) of corresponding verteces gets an additional weight w that reflects the length of the incident edges e_p^-, e_p^+ and e_q^-, e_q^+ respectively if they belong to the covered part of the polyline:
$w(p, q) = (|e_p^-| + |e_p^+| + |e_q^-| + |e_q^+|)/4$
For the class T of allowable transformations being the class of translations or rigid motions, the transformation $t \in T$ that minimizes the sum of the weighted squared distances $\sum_{(p_i,q_i)\in s} w(p_i, q_i) \|q_i - t(p_i)\|^2$ and the vertex correspondences may be determined independently from each other. But for the classes of transformations that allow scalings (i.e. homotheties, similarities, and affine maps), the assignment of the verteces depends on that scaling.

Scalings If scalings are allowed, for every single vote a prescaling factor c_i is randomly chosen such that $\mathrm{ld}(c_i)$ is normally distributed with mean value $\mathrm{ld}(\bar{c}_i)$ where $\bar{c}_i$ is the prescaling factor of the transformation rated best so far. $S_1^{c_i}$ denotes the set S_1 scaled by c_i. For the rest of the vote, every operation concerning the first set S_1 (e.g. computing the distance between two vertices) is performed on $S_1^{c_i}$.

Translations A translation is defined by a translation vector $v = (v_x, v_y)^\perp$. A point $p = (p_x, p_y)^\perp$ is mapped to $p + v$.

For a set of pairs of points $s = \{(p_1, q_1), \ldots, (p_k, q_k)\}$ the translation $t = v$ that minimizes the sum of the squared distances $\varepsilon^* = \sum_{i=1}^k \|t(p_i) - q_i\|^2$ can easily be computed as $v = \frac{1}{k} \sum_{i=1}^k q_i - p_i$

The translation $t = v$ that minimizes the sum of the weighted squared distances $\varepsilon = \sum_{i=1}^k w(p_i, q_i) \|t(p_i) - q_i\|^2$ can easily be computed as $v = \frac{1}{w(s)} \sum_{i=1}^k w(p_i, q_i)(q_i - p_i)$ with $w(s)$ being the sum of all weights.
That means that having computed the optimal transformation for the set $s' \subset s$ of cardinality $k - 1$ the optimal transformation for the set s can be computed in constant time.

Homotheties A homothety is defined by a scaling factor c and a translation vector $v = (v_x, v_y)^\perp$. A point $p = (p_x, p_y)^\perp$ is mapped to $p \cdot c + v$.

The optimal homothety is not computed directly, but first the prescaling is applied as described above, and then the optimal translation is computed in a second step. The scaling factor is restricted to be a positive real number, so that inversions are waived.

Rigid Motions A rigid motion is defined by a rotation angle φ and a translation vector $v = (v_x, v_y)^\perp$. A point $p = (p_x, p_y)^\perp$ is mapped to $mp + v$ with m being the matrix $\begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix}$.

For a set of pairs of points $s = \{(p_1, q_1), \ldots, (p_k, q_k)\}$ the rigid motion $t = (\varphi, v)$ that minimizes the sum of the squared distances $\varepsilon^* = \sum_{i=1}^{k} \|t(p_i) - q_i\|^2$ can easily be computed (as described by [9]):
Let $\bar{p} = \frac{1}{k}\sum_{i=1}^{k} p_i$, $\bar{q} = \frac{1}{k}\sum_{i=1}^{k} q_i$, $\hat{p}_i = p_i - \bar{p}$, and $\hat{q}_i = q_i - \bar{q}$.
For $a = \sum_{i=1}^{k} (q_{ix}p_{ix} + q_{iy}p_{iy})$ and $b = \sum_{i=1}^{k} (q_{ix}p_{iy} - q_{iy}p_{ix})$ the rotation matrix is $m = \frac{1}{\sqrt{a^2+b^2}} \cdot \begin{pmatrix} a & b \\ -b & a \end{pmatrix}$
The translation vector v is determined as $v = \bar{q} - m \cdot \bar{p}$.

The rigid motion $t = (\varphi, v)$ that minimizes the sum of the weighted squared distances $\varepsilon = \sum_{i=1}^{k} w(p_i, q_i) \|t(p_i) - q_i\|^2$ can be computed using the method described above with slight modifications:
The centers of mass $\bar{p}$ and $\bar{q}$ are replaced by the weighted centers of mass. To compute the rotation according to the weighted squared distances instead of the squared distances, the radii (distance of a point to the weighted center of mass) are resized for that part. $\hat{p}_i = \sqrt{w(p_i, q_i)} \cdot (p_i - \bar{p})$, and $\hat{q}_i = \sqrt{w(p_i, q_i)} \cdot (q_i - \bar{q})$. With this new pairs of points, the conventional computation of m jields the desired result.
All the terms can be transformed to avoid the explicit computation of the (weighted) centers of mass $\bar{p}$ and $\bar{q}$ such that having computed the optimal transformation for the set $s' \subset s$ of cardinality $k-1$ the optimal transformation for the set s can be computed in constant time.

Similarities A similarity is defined by a rotation angle φ, a scale c, and a translation vector $v = (v_x, v_y)^\perp$. A point $p = (p_x, p_y)^\perp$ is mapped to $mp + v$ with m being the matrix $\begin{pmatrix} c \cdot \cos\varphi & -c \cdot \sin\varphi \\ c \cdot \sin\varphi & c \cdot \cos\varphi \end{pmatrix}$.

First the prescaling is applied as described above, and then the optimal similarity is computed in a second step.
For a set of pairs of points $s = \{(p_1, q_1), \ldots, (p_k, q_k)\}$ the similarity $t = (m, v)$ that minimizes the sum of the weighted squared distances $\varepsilon = \sum_{i=1}^{k} w(p_i, q_i) \|t(p_i) - q_i\|^2$ can easily be computed by solving the linear equation system

$$\sum_{i=1}^{k} w(p_i, q_i) \begin{pmatrix} {p_{ix}}^2 + {p_{iy}}^2 & 0 & p_{ix} & p_{iy} \\ 0 & {p_{ix}}^2 + {p_{iy}}^2 & p_{iy} & -p_{ix} \\ p_{ix} & p_{iy} & 1 & 0 \\ p_{iy} & -p_{ix} & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} m_{11} \\ m_{12} \\ v_x \\ v_y \end{pmatrix} = \sum_{i=1}^{k} w(p_i, q_i) \begin{pmatrix} q_{ix}p_{ix} + q_{iy}p_{iy} \\ q_{ix}p_{iy} - q_{iy}p_{ix} \\ q_{ix} \\ q_{iy} \end{pmatrix}$$

That means that having computed the optimal transformation for the set $s' \subset s$ of cardinality $k-1$ the optimal transformation for the set s can be computed in constant time.

Affine Maps An affine map is defined by an arbitrary matrix $m = \begin{pmatrix} m_{1,1} & m_{1,2} \\ m_{2,1} & m_{2,2} \end{pmatrix}$ and a translation vector $v = (v_x, v_y)^\perp$. A point $p = (p_x, p_y)^\perp$ is mapped to $mp + v$.

First the prescaling is applied as described above, and then the optimal affine map is computed in a second step.
For a set of pairs of points $s = \{(p_1, q_1), \ldots, (p_k, q_k)\}$ the affine map $t = (m, v)$ that minimizes the sum of the weighted squared distances $\varepsilon = \sum_{i=1}^{k} w(p_i, q_i) \, \|t(p_i) - q_i\|^2$ can easily be computed by solving the linear equation system

$$\sum_{i=1}^{k} w(p_i, q_i) \begin{pmatrix} p_{ix}p_{ix} & p_{ix}p_{iy} & p_{ix} & & & \\ p_{ix}p_{iy} & p_{iy}p_{iy} & p_{iy} & & 0 & \\ p_{ix} & p_{iy} & 1 & & & \\ & & & p_{ix}p_{ix} & p_{ix}p_{iy} & p_{ix} \\ & 0 & & p_{ix}p_{iy} & p_{iy}p_{iy} & p_{iy} \\ & & & p_{ix} & p_{iy} & 1 \end{pmatrix} \cdot \begin{pmatrix} m_{1,1} \\ m_{1,2} \\ v_x \\ m_{2,1} \\ m_{2,2} \\ v_y \end{pmatrix} = \sum_{i=1}^{k} w(p_i, q_i) \begin{pmatrix} q_{ix}p_{ix} \\ q_{ix}p_{iy} \\ q_{ix} \\ q_{iy}p_{ix} \\ q_{iy}p_{iy} \\ q_{iy} \end{pmatrix}$$

If the points are collinear, the affine map is not uniquely defined and the linear equation system has no unique solution. In this special case, the optimal rigid motion is taken instead. Again, having computed the optimal transformation for the set $s' \subset s$ of cardinality $k-1$ the optimal transformation for the set s can be computed in constant time.

2.2.3 Checking Consistency

For every element of the sequence the consistency of the sample set $(p_0, q_0) \ldots (p_i, q_i)$ and the appendant transformation t_i has to be checked. Although consistency is a property concerning the whole sample set, only the last added pair is considered because the emersion of crucial inconsistency has to be appreciable in the pair causing it.
For $l_{bb,i}$ being the perimeter of the bounding box containing the covered part of the polyline from the initial pair (p_0, q_0) up to the pair (p_i, q_i) a maximum tolerated error is defined as $e_{max,i} = l_{bb,i} \cdot r$ with r being a constant called the relative error threshold. If the distance $e_i = \|q_i - t_i(p_i)\|$ exceeds the maximum tolerated error, the traversal of the polylines is ceased. The index j for which $e_{max,j} - e_j$ takes the maximum value, defines the transformation t_j for the vote (see fig. 11 and 12).
This definition of the stop criterion and the choice of the best index are invariant under scalings and can be done in constant time. To achieve invariance under rotations also, the bounding box had to be replaced by the minimum enclosing circle.

2.2.4 Weighting the Transformations

The two factors that have to be considered for weighting a transformation t are the expressiveness of the sample and the quality of the match. Both have to be balanced out so that

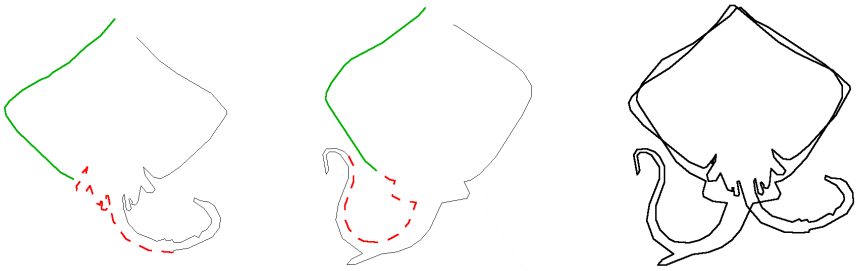


Figure 11: Two instances of the mpeg7shapeB dataset (ray-7, ray-20 and both mapped)

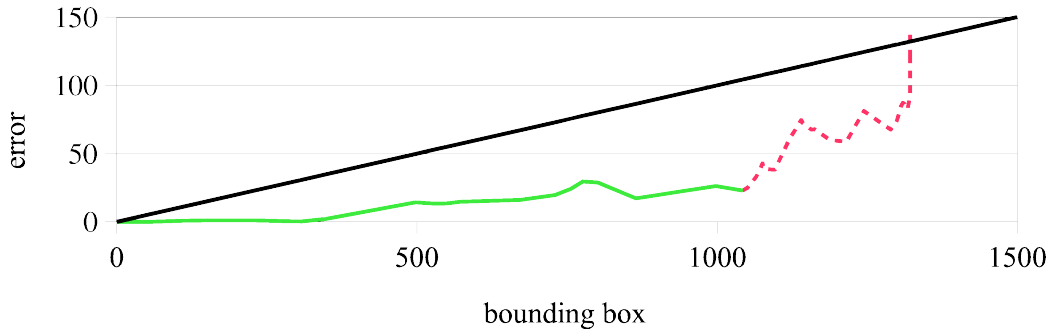


Figure 12: Perimeter of the bounding box vs. error of last added pair of points. The part of the polyline defining the transformation for this vote is plotted green, the part skipped is plotted red and dashed.

no part overrules the other.
The size of the set s is not a good choice for the expressiveness. A jag may be represented by three verteces whereas the number of verteces needed to represent an arc is depending on the resolution and may be arbitrary large. Another possible value is the length l of the corresponding part of the polyline. It is easy to compute and does not suffer from the same imbalance.

In a similar context [21] use the mean squared difference e' of two graphs in the arclength v. turning angle diagram for valuing the quality of a match. Their score function for comparing matches is basically $l/(1+e')$ but they mention that other score functions are possible.

Their definition has the drawback that the comparison of two matches is not scale invariant. Let ε be the residual of the minimized objective function, let $w(s)$ be the sum of all weights of the pairs of points in the sample set s, and let d_{bb} be the diameter of the bounding-box containing the covered part of the polyline. Defining the relative root mean square error $e = \sqrt{\varepsilon/w(s)}/d_{bb}$ jields a value representing the quality of the match which is invariant under scalings.
The match score or weight $w(t)$ of a transformation t is then defined as $w(t) = l/(1+\gamma \cdot e)$ with γ being an arbitrarily chosen constant for balancing out the impact of the length l and the error e.

2.2.5 Transformation Clustering

The most common technique for the clustering of transformations (often referred to as pose clustering) – histogramming the transformations in the multidimensional transformation space (see [13]) – discards the effects on the transformed shapes. Two rotations may yield nearly the same results if applied to a shape whith its center being the origin, or totally different results if the shape's center is far away from the origin. To avoid this imbalance, a distance measure for transformations is used here, which consideres the shapes' properties.
Let t_1 and t_2 be two arbitrary transformations and let S be the transformed shape. The favoured distance measure would be $d_S(t_1, t_2) = \max_{p \in S} \|t_1(p) - t_2(p)\|$. To reduce the computation costs a surrogat only consideres a discrete set S' of sample points, e.g. the vertices of the bounding box of S: $d'_S(t_1, t_2) = \max_{p \in S'} \|t_1(p) - t_2(p)\|$
Under the assumption that the four points of S' are pairwise different, this forms a metric space for affine maps, and independently of the class of allowable transformations the triangle inequality holds.

So the task is to cluster discrete weighted points in a metric space.

Clusters
A cluster in our sense represents a region of limited diameter, wich subsumes a considerable amount of weights of the enclosed input points (transformations).

Let T_n be the set of n preliminary transformations and w_i be the weight of transformation $t_i \in T_n$. For every transformation t_k the set of its dominators in range r is defined as $s_r(t_k) = t_j \in T_n | d(t_k, t_j) < r, w_k < w_j, s_r(t_j) = \emptyset$. Every transformation t_c which has no dominators in range r_c defines a cluster with radius r_c as the set $\{t_i \in T_n | d(t_c, t_i) < r_c\}$, i.e. a transformation is either assigned to its dominators' clusters or it defines a cluster itself. The weight of a cluster is defined as the sum of the weights of its elements. This definition allows for a fast computation of all clusters and their weights.

Partitioning the Transformation Space
Given the n preliminary transformations T_n and a radius r_c for the clusters, a rooted tree that defines a partition of the transformation space is iteratively built up in the following way:
A node N on level i represents a ball with radius r_i around its center. It may have arbitrarily many children, each one representing a ball with radius $r_{i+1} = r_i/2$ and a center that lies inside N's ball. A node may hold a cluster. If so, the node's center is defined to be the clusters center. The root node represents a ball with radius r_0 containing all elements of T_n. For every cluster c being inserted into a node representing a ball with radius r_i if there exists (at least one) child node representing a ball containing the center of c, c is recursively inserted into the first such child. Otherwise a new child holding c with radius $r_i/2$ is created and appended to the list of child nodes.

Building the Clusters
The preliminary transformations are sorted according to their weight and they are processed in descending order. Given a transformation t, the tree is searched for all clusters $c_1, \ldots, c_k$ such that $d(t_c, t) \leq r_c$ for the cluster's center t_c. If at least one cluster is found, t is added to the clusters. Otherwise, a new cluster c with center t is created and inserted into the tree.
The childs of a node are ordered hierarchically, each one only responsible for the part of the

space not covered by its predecessors. When a node with center t_N and radius r is searched for clusters neighbouring a transformation t, the distance $d(t_N, t)$ is computed. If $d(t_N, t) < r - r_c$, all the clusters worth considering have to lie inside the node's ball and the successors may be discarded. If $d(t_N, t) > r + r_c$ the clusters have to lie outside and the node may be discarded. In the other cases the node, its successors, and its children have to be considered.

Properties of the Tree
Assuming that the considered space T_i is bounded, i.e. $\exists r_i \in \mathbb{R} : \forall x, y \in T_i : d(x, y) < r_i$. A subset $C_i \subset T_i$ is called an ε-packing iff $\forall x, y \in C_i : d(x, y) > 2\varepsilon$. The size of the largest ε-packing is called the packing number $P(T_i, \varepsilon)$. For $\varepsilon \to 0$, $P(T_i, \varepsilon) = O\left(\left(\frac{r_i}{\varepsilon}\right)^{\mathcal{D}}\right)$ for $\mathcal{D}$ being the dimension of the space [11]. Therefore the maximum number of balls of radius $r_i/2$ with centers in T_i such that no center is contained in another ball, is in $O(2^{\mathcal{D}})$.
That means that the number of childs a node of the tree may have is bounded by a constant only depending on the dimension of the transformation space.
The depth of tree is at most $\lceil \lg(r_0/r_c) \rceil$.

Initial Nodes
The center $t_{N,0}$ and the radius r_0 of the ball represented by the root node may be easily computed for the classes of transformations that do not allow scalings:
Let c_{bb1} be the center and d_{bb1} be the diameter of the bounding box of S_1, let c_{bb2} be the center and d_{bb2} be the diameter of the bounding box of S_2 respectively. Then $t_{N,0}$ is chosen to be the translation defined by $c_{bb2} - c_{bb1}$. Any generated preliminary transformation t will fulfill the condition that the transformed bounding box of S_1 at least touches the bounding box of S_2. Therefore $d_{S_1}(c_0, t) \leq \frac{d_{bb1}}{2} + \frac{d_{bb2}}{2} + d_{bb1}$ for all accruing transformations.
For the classes of transformations that do allow scalings the space is not bounded in such a natural way. However, the application provides some other bounds: The goal is to find transformations that support the human perception. Extreme scalings and shearings may bring features into congruence, that a human would never be able to perceive.
With a maximum allowable scaling factor s_{max} an appropriate upper bound for the radius is defined by $d_{S_1}(c_0, t) \leq \frac{d_{bb1}}{2} + \frac{d_{bb2}}{2} + s_{max} d_{bb1}$ for similarity transformations.

2.2.6 Candidate Transformations

After having processed all preliminary transformations, the clusters are sorted according to their weights. The clusters with the highest weights provide the candidate transformations. The number of candidate transformations considered may be chosen as a constant or a threshold $0 \leq u \leq 1$ may be defined to consider only the transformations with weight $w \geq w_{max} \cdot u$.

2.3 Similarity of matched sets of line segmants or polygonal curves

After having computed transformation that match the shapes, a distance measure has to be applied to rate the disimilarity (the similarity respectively) of the two shapes. To manage some of the problems arising when applying the well known Hausdorff distance or Fréchet distance, a new similarity measure is described here. It...

- averages over the whole polyline / set of polylines so that noise is suppressed
- takes into account the special properties of line segments

- is resistent to different parameterization or splitting of polylines

2.3.1 Resemblance Function

The resemblance function is defined for every point of the polylines and stands for how good the point is represented by the other set. It is composed of the point's distance to the other shape's points and of the similarity of slopes.
Let s be a line segment of S_1 with endpoints p_0 and $p_0 + v$ and let s' be it's supporting line. Let g be a line segment of S_2 with endpoints p_1 and p_2 and let g' be it's supporting line.

distance function
If s and g are not orthogonal a distance function $\delta_{s,g}$ is defined as follows: To every point $p'(\lambda) = p_0 + \lambda v$ of s', a point $p(\lambda)$ of g' is assigned such that $p'(\lambda)$ is the orthogonal projection of $p(\lambda)$ onto s'. The orthogonal projection of p_1 onto s' is denoted by $p'_1 = p_0 + \lambda_1 \cdot v$ and its distance vector is denoted by $d_1 = p_1 - p'_1$. The orthogonal projection of p_2 onto s' is denoted by $p'_2 = p_0 + \lambda_2 \cdot v$. Its distance vector is denoted by $d_2 = p_2 - p'_2$
The distance vector belonging to $p_0 = p(0)$ is the extrapolation $d'_1 = d_2 \cdot (0 - \lambda_1)/(\lambda_2 - \lambda_1) + d_1 \cdot (1 - (0 - \lambda_1)/(\lambda_2 - \lambda_1))$ and the distance vector belonging to $p_0 + v = p(1)$ is the extrapolation $d'_2 = d_2 \cdot (1 - \lambda_1)/(\lambda_2 - \lambda_1) + d_1 \cdot (1 - (1 - \lambda_1)/(\lambda_2 - \lambda_1))$ (see fig. 13).
W.l.o.g. let $\lambda_1 < \lambda_2$.
The distance function $\delta_{s,g}$ for this pair of line segments is defined as

$$\delta_{s,g}(\lambda) = \begin{cases} \|d_1\| + \|(\lambda - \lambda_1) \cdot v\|, & 0 \leqslant \lambda < \lambda_1 \\ \|(1 - \lambda) \cdot d'_1 + \lambda \cdot d'_2\|, & \lambda_1 \leqslant \lambda \leqslant \lambda_2 \\ \|d_2\| + \|(\lambda - \lambda_2) \cdot v\|, & \lambda_2 < \lambda \leqslant 1 \end{cases}$$

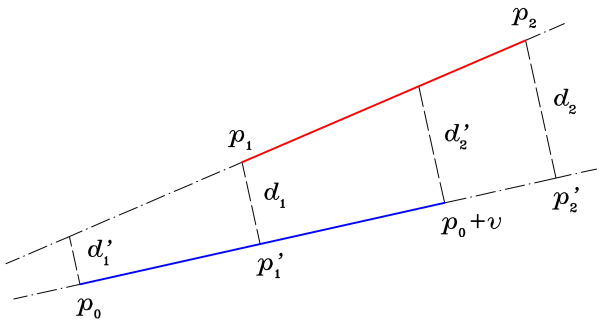


Figure 13: definition of the distance between two line segments

inverse distance function
If the distance $\delta_{s,g}(\lambda_p)$ of a point $p = p_0 + \lambda_p v$ (belonging to line segment $s \in S_1$) and the segment $g \in S_2$ equals zero, then p is exactly represented by g – the grade of being represented therefore is 1. The greater the distance gets, the lesser p is represented by g but the decrease surely depends on the extent of the shape S_1.

In a similar context [8] use a – what they call – inverse distance function for the rating of transformations in an optimization problem. For their task they chose a function that exponentially decreases with higher Euclidean distance to value the correspondence of features

(see fig. 14a).
In the present case the goal is not to find an optimum, but to rate a given configuration. Small deviations in the position of the features of the two sets should not result in an excessive devaluation. Therefore an inversion function with a high (negative) slope around the y-axis is inapplicable. The function $\alpha'_{s,g}(\lambda) = \exp(25(\delta_{s,g}(\lambda)/D_{S_1})^2)$ seems more promising. It rates pairs with a distance less than 5 % of the diameter very high (over 0.9) and with a distance of more than 25 % of the diameter very low – around 0.2 (see fig. 14b).
To make the computation easier, the piecewise quadratic function

$$\alpha_{s,g}(\lambda) = \max(1 - 25\left(\frac{\delta_{s,g}(\lambda)}{D_{S_1}}\right)^2, 0)$$

is chosen. It has the same characteristics for small distances (up to 10 %) but decreases faster for greater distances (see fig. 14c).

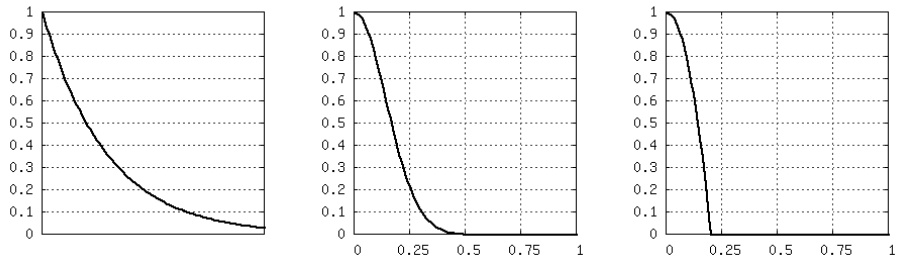


Figure 14: (a) exponentially decreasing inverse distance; (b) inversion function α'; (c) inverse distance function α

slope
The resemblance of two line segments also depends on their slopes. Line segments with similar slopes should get a higher resemblance value, so a slope factor $\beta_{s,g}$ is defined as

$$\beta_{s,g} = \cos\left(\angle(s,g)\right)^4 = \left(\frac{\langle s, g\rangle}{\|s\| \cdot \|g\|}\right)^4$$

It rates pairs with a difference in slopes of less than 10° very high (over 0.9) and with a difference of more than 45° very low (below 0.25). **resemblance**
The resemblance function ϕ_s for a line s is defined as

$$\phi_s(\lambda) = \max_{g \in S_2}\left(\alpha_{s,g}(\lambda) \cdot \beta_{s,g}\right) \tag{1}$$

The functions α and β are chosen arbitrarily – any monotone function that decreases from $(0,1)$ to $(1,0)$ resp. $(\pi/2, 0)$ may be utilized – but the two proposed here yield the best results encountered so far and they allow for a fast computation.

weight
To prevent parts with many parallel lines from dominating over parts with solitary lines, a weighting function ω is defined analogical to the resemblance function. It rates the density

of similar lines of an image. For a line $s \in S_1$ it is defined as

$$\omega_s(\lambda) = \frac{1}{\sum_{g \in S_1} (\alpha_{s,g}(\lambda) \cdot \beta_{s,g})} \tag{2}$$

2.3.2 Resemblance / Deviation

The directed resemblance measure $\Phi_{\rightarrow}(S_1, S_2)$ is defined by

$$\Phi_{\rightarrow}(S_1, S_2) = \frac{\sum_{s \in S_1} \left(\int_{\lambda=0}^{1} \phi_s(\lambda) \cdot \omega_s(\lambda)\, d\lambda \cdot l_s \right)}{\Omega(S_1)}$$

with l_s being the length of s and $\Omega(S_1)$ being the total weight of S_1:
$\Omega(S_1) = \sum_{s \in S_1} \left(\int_{\lambda=0}^{1} \omega_s(\lambda)\, d\lambda \cdot l_s \right)$

The undirected resemblance measure $\Phi(S_1, S_2)$ is defined as the weighted arithmetic mean:

$$\Phi(S_1, S_2) = \frac{\Phi_{\rightarrow}(S_1, S_2) \cdot \Omega(S_1) + \Phi_{\rightarrow}(S_2, S_1) \cdot \Omega(S_2)}{\Omega(S_1) + \Omega(S_2)}$$

From this resemblance measure a deviation or distance measure may be derived, but of course this will never be a metric as the triangle inequality does not hold.

2.3.3 Complexity

The resemblance value is computed evaluating the integrals of a combination of the resemblance function and the weighting function for every line segment.
For two sets with n line segments each, the resemblance function – as defined in eq 1 – for a single line segment is the upper envelope of at most $4 \cdot n + 1$ regular (partially defined) functions. Using quadratic functions, each pair intersects at most 2 times (unless equal). According to the upper bound on the length of Davenport-Schinzel sequences the complexity of the upper envelope of the $4 \cdot n + 1$ functions is bounded by $O(n \cdot 2^{\alpha(n)})$ [1] with α being the inverse Ackermann function.
The weighting function for a single line segment – as defined in eq 2 – is the sum of n functions, each one split into at most 4 regular pieces. The number of intervals for the sum is at most $3n + 1$.

So the overall complexity for all the line segments is bounded by $O(n^2 \cdot 2^{\alpha(n)})$. Two shapes that yield quadratic complexity can easily be found (see the example in fig. 15).

2.4 Turning function based matching

The turning function Θ_A of a polygon A measures the angle of the counterclockwise tangent with respect to a reference orientation as a function of the arc-length s, measured from some reference point on the boundary of A. It is a piecewise constant function, with jumps corresponding to the vertices of A. A rotation of A by an angle θ corresponds to a shifting of Θ_A over a distance θ in the vertical direction. Moving the location of the reference point $A(0)$ over a distance t along the boundary of A corresponds to shifting Θ_A horizontally over a distance t.

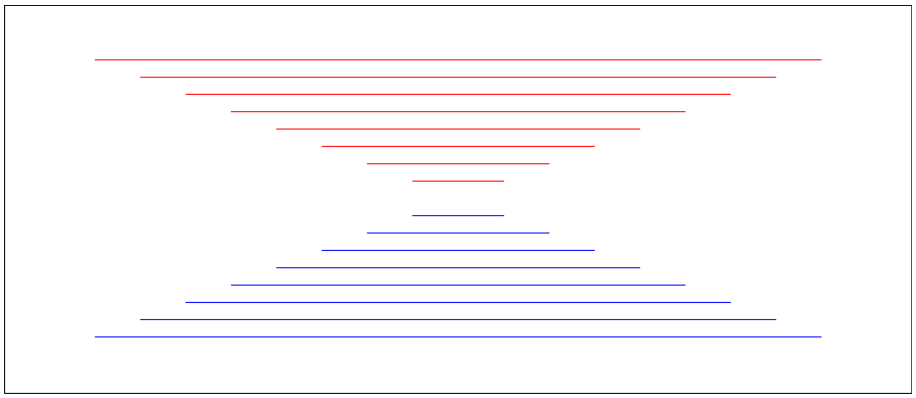


Figure 15: shapes with weighting function of complexity $O(n^2)$

The distance between two polygons A and B is defined as the L_2 distance between their two turning functions Θ_A and Θ_B, minimized with respect to the vertical and horizontal shifts of these functions (in other words, minimized with respect to rotation and choice of reference point). More formally, suppose A and B are two polygons with perimeter length l_A and l_B, respectively, and the polygon B is placed over A in such a way that the reference point $B(0)$ of B coincides with point $A(t)$ at distance t along A from the reference point $A(0)$, and B is rotated clockwise by an angle θ with respect to the reference orientation. A pair (t, θ) will be referred to as a placement of B over A. The first component t of a placement (t, θ) is also called a horizontal shift, since it corresponds to a horizontal shifting of Θ_A over a distance t, while the second component θ is also called a vertical shift, since it corresponds to a vertical shifting of Θ_A over a distance θ. We define the quadratic similarity $f(A, B, t, \theta)$ between A and B for a given placement (t, θ) of B over A, as the square of the L_2-distance between their two turning functions Θ_A and Θ_B:

$$f(A, B, t, \theta) = \int (\Theta_A(s + t) - \Theta_B(s) + \theta)^2 \, \mathrm{d}\, s \quad .$$

The similarity between two polygons A and B is then given by:

$$\min_{(\theta,t)} \{f(A, B, t, \theta)\} \quad .$$

An efficient algorithm to compute this similarity measure is given by Arkin et al. [6]. For measuring the difference between a polygon and a polyline, or between two polylines, the same measure can be used with a few slight adaptations.

We have implemented these functions in C++. See for the reference manual in Appendix B.

References

[1] Pankaj K. Agarwal and Micha Sharir. Davenport-shinzel sequences and their geometric applications, 1995.

[2] Alberto S. Aguado, Eugenia Montiel, and Mark S. Nixon. Invariant characterisation of the hough transform for pose estimation of arbitrary shapes. *Pattern Recognition*, 35:1083–1097, 2002.

[3] Helmut Alt, Bernd Behrends, and Johannes Blömer. Approximate matching of polygonal shapes. *Annals of Mathematics and Artificial Intelligence*, 13:251–265, 1995.

[4] Helmut Alt and Leonidas J. Guibas. Discrete geometric shapes: Matching, interpolation, and approximation. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of computational geometry*. Elsiever Science Publishers B.V. North-Holland, Amsterdam, 1999.

[5] Helmut Alt, Günter Rote, Carola Wenk, and Alon Efrat. Matching planar maps. *J. of Algorithms*, pages 262–283, 2003.

[6] E. Arkin, P. Chew, D. Huttenlocher, K. Kedem, and J. Mitchell. An efficiently computable metric for comparing polygonal shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3):209–215, March 1991.

[7] Fred Attneave. Some informational aspects of visual perception. *Psychological Review*, 61(3):183–193, 1954.

[8] Harald Ganster Axel Pinz, Manfred Prantl. A robust affine matching algorithm using an exponentially decreasing distance function. *Journal of Universal Computer Science*, 1(8):614–631, 1995.

[9] John H. Challis. Estimation of the finite center of rotation in planar movements.

[10] Otfried Cheong, Alon Efrat, and Sariel Har-Peled. On finding a guard that sees most and a shop that sells most. In *Proc. 15th ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 1091–1100, 2004.

[11] Kenneth L. Clarkson. Nearest-neighbor searching and metric space dimensions, 2005.

[12] Alon Efrat and Suresh Venkatasubramanian. Curve matching, time warping, and light fields. Technical Report AT&T TD-4z5TMU, AT&T.

[13] Clark f. Olson. Efficient pose clustering using a randomized algorithm. *International Journal of Computer Vision*, 23:131–147, 1997.

[14] M. Hagedoorn, M.H. Overmars, and R.C. Veltkamp. A new visibility partition for affine pattern matching. In *Proc. Discrete Geometry for Computer Imagery conference, DGCI 2000*, pages 358–370, Berlin, 2000. Springer-Verlag.

[15] M. Hagedoorn and R. Veltkamp. State-of-the-art in shape matching. Technical Report UU-CS-1999-27, Utrecht University, the Netherlands, 1999.

[16] D. Huttenlocher and S. Ullman. Recognizing solid objects by alignment with an image. *International Journal of Computer Vision*, 5(2):195–212, 1990.

[17] Robert C. Bolles Martin A. Fischler. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24:381–395, 1981.

[18] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[19] M. E. Munich and P. Perona. Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification. In *Proceedings of 7th International Conference on Computer Vision*, Korfu, Greece, September 1999.

[20] Clark Francis Olson. *Fast Object Recognition by Selectively Examining Hypotheses*. PhD thesis, University of California at Berkeley, 1994.

[21] Leonidas J. Guibas Scott D Cohen. Partial matching of planar polylines under similarity transformations.

[22] George Stockman. Object recognition and localization via pose clustering. *Computer Vision, Graphics, and Image Processing*, 40:361–387, 1987.

[23] Remco C. Veltkamp. Shape matching: Similarity measures and algorithms. Technical Report UU-CS-2001-03, Utrecht University, 2001.

[24] Haim J. Wolfson and Isidore Rigoutsos. Geometric hashing: An overview. *IEEE Computational Science and Engineering*, 04(4):10–21, 1997.

A Proofs for probabilistic matching of sets of line segments

Here we give the proofs for the running time of the matching algorithm under translations described in section 2.1.1. We use the same notation as in section 2.1.1.

A.1 Absolute Error

Let $0 \leq \varepsilon \leq 1$ be a parameter, then for a single translation $t \in \mathbb{R}^2$ holds:

Lemma A.1. *Let S be a sample of N translation vectors and $t \in \mathbb{R}^2$ a translation, then $P(|\tilde{p}(t) - p(t)| > \varepsilon) \leq 2e^{-\frac{\varepsilon^2 N}{2}}$.*

Proof. Let $S = \{s_1, \ldots, s_N\}$, consider independent random variables $X_i,\ i = 1..N$ defined as $X_i = 1$ if s_i is in the δ-neighborhood of t and $X_i = 0$ else, and a random variable $X = \sum_{i=1}^{N} X_i$, $X = a(t)$. Expectation value of X is $E(X) = Np(t)$.

Applying Lemma 4.1 in [10] we get $P(|X - E(X)| > \varepsilon N) < e^{-\varepsilon^2 N/2}$.
$P(|X - E(X)| > \varepsilon N) = P(|a(t) - Np(t)| > \varepsilon N) = P(|\tilde{p}(t) - p(t)| > \varepsilon) < e^{-\varepsilon^2 N/2}$. □

Consider the arrangement of δ-neighborhoods of the translation vectors of some sample S. If a translation t is in the δ-neighborhood of the sample vector s, then obviously s is also in the δ-neighborhood of t. The translations within the same cell of the arrangement have the same sample vectors of S in their δ-neighborhoods. For the vertices of the arrangement we have then:

Lemma A.2. *Let $0 < \varepsilon, \eta < 1$ be parameters, let S be a sample of $N \geq \frac{c \ln \frac{1}{\eta}}{\varepsilon^2} + 2$ translation vectors for a suitable constant c, then*

$$P(\exists t \in \mathbb{R}^2 \text{ such that } |\tilde{p}(t) - p(t)| > \varepsilon) \leq \eta \ .$$

Proof. In Lemma A.1 we proved that for any translation vector t in $\mathbb{R}^2$ the probability $P(|\tilde{p}(t) - p(t)| > \varepsilon) \leq 2e^{-\frac{\varepsilon^2 N}{2}}$. Here we consider separately the vertices of the arrangement, since they depend on the experiment.

Let t be a vertex of the arrangement, it is then an intersection point of the boundaries of two δ-neighborhoods of vectors $s_1, s_2 \in S$. Consider a sample $Q = S \setminus \{s_1, s_2\}$, $|Q| = N - 2$. Let $a_Q(t)$ and $a_S(t)$ denote the number of sample vectors in the δ-neighborhood of t in samples Q and S respectively, and $\tilde{p}_Q(t) = a_Q(t)/(N-2)$, $\tilde{p}_S(t) = a_S(t)/N$. If we consider open neighborhoods, then $a_Q(t) = a_S(t)$ and

$$\tilde{p}_Q(t) = \frac{a_Q(t)}{N-2} = \frac{a_S(t)}{N}\frac{N}{N-2} = \tilde{p}_S(t)\left(1 + \frac{2}{N-2}\right) \leq \tilde{p}_S(t) + \frac{2}{N-2} \ .$$

Therefore,

$$|\tilde{p}_S(t) - p(t)| \leq |\tilde{p}_Q(t) - p(t)| + |\tilde{p}_S(t) - \tilde{p}_Q(t)| \leq |\tilde{p}_Q(t) - p(t)| + \frac{2}{N-2} \ .$$

Then,

$$\begin{aligned} P(|\tilde{p}_S(t) - p(t)| > \varepsilon) &\leq P\left(|\tilde{p}_Q(t) - p(t)| + \frac{2}{N-2} > \varepsilon\right) \\ &= P\left(|\tilde{p}_Q(t) - p(t)| > \varepsilon - \frac{2}{N-2}\right) \end{aligned}$$

with $N \geq \frac{16 \ln \frac{1}{\eta}}{\varepsilon^2} + 2 \geq \frac{4}{\varepsilon} + 2$ is $\varepsilon - \frac{2}{N-2} \geq \frac{\varepsilon}{2}$

$$\begin{aligned} &\leq P\left(|\tilde{p}_Q(t) - p(t)| > \frac{\varepsilon}{2}\right) \quad \text{(by Lemma A.1)} \\ &\leq 2e^{-\frac{\varepsilon^2(N-2)}{8}} \end{aligned}$$

The arrangement consists of N (circular or square) δ-neighborhoods and has at most N^2 vertices. This implies that the probability that there exists a vertex t such that $|\tilde{p}_S(t) - p(t)| > \varepsilon$ is at most $N^2 2e^{-\frac{\varepsilon^2(N-2)}{8}}$.

For sufficiently large N: $N^2 2e^{-\frac{\varepsilon^2(N-2)}{8}} \leq e^{-\frac{\varepsilon^2(N-2)}{16}}$, and with $N \geq \frac{16 \ln \frac{1}{\eta}}{\varepsilon^2} + 2$ the probability that there exists a vertex of the arrangement that has a bad estimate is at most η. □

Let t_{opt} be a vector maximizing $p(t)$, t^* a vertex of the cell of the arrangement containing t_{opt}, and let t_{app} be a vertex of the arrangement maximizing $\tilde{p}(t)$, then we find that $P\left(|\tilde{p}(t^*) - p(t_{\text{opt}})| > \varepsilon\right) \leq 2e^{-\frac{\varepsilon^2 N}{8}}$, which is less than η for $N \geq \frac{c \ln \frac{1}{\eta}}{\varepsilon^2} + 2$. Then with probability at least $1 - \eta$ $|\tilde{p}(t_{\text{app}}) - p(t_{\text{app}})| \leq \varepsilon$ and $|\tilde{p}(t^*) - p(t_{\text{opt}})| \leq \varepsilon$ and

$$\begin{aligned} &\tilde{p}(t_{\text{app}}) \geq \tilde{p}(t^*) \geq p(t_{\text{opt}}) - \varepsilon \qquad \text{and} \\ &\tilde{p}(t_{\text{app}}) \leq p(t_{\text{app}}) + \varepsilon \leq p(t_{\text{opt}}) + \varepsilon \ . \end{aligned}$$

Therefore $|\tilde{p}(t_{\text{app}}) - p(t_{\text{opt}})| \leq \varepsilon$ with high probability.

Proof. (Of Theorem 2.1)
With linear time preprocessing we can generate a random point of the shape in time $O(\log n)$. That is, the time to generate N random points on both shapes is $O(n + N \log n)$. The arrangement of the δ-neighborhoods of N vectors has the complexity $O(N^2)$ and can be computed and traversed in time $O(N^2)$. □

A.2 Relative Error

Let $0 < \varepsilon, \eta, \nu \leq 1$ be parameters. Then for a single translation we can prove the following relative error bound:

Lemma A.3. *Let S be a sample of size $N \geq \frac{c_1}{\varepsilon^2 \nu} \ln \frac{2}{\eta}$, where c_1 is a suitable constant. Let $t \in \mathbb{R}^2$ be a translation vector, then*

- $p(t) \leq \nu \Rightarrow P(\tilde{p}(t) > \varepsilon\nu) \leq \eta$
- $p(t) \geq \nu \Rightarrow P(|\tilde{p}(t) - p(t)| > \varepsilon p(t)) \leq \eta$.

Proof. Let $S = \{s_1, \ldots, s_N\}$, consider independent random variables X_i, $i = 1..N$ defined as $X_i = 1$ if s_i is in the δ-neighborhood of t and $X_i = 0$ else, and a random variable $X = \sum_{i=1}^{N} X_i$, $X = a(t)$. Expectation value of X is $E(X) = Np(t)$.

In case $p(t) \geq \nu$:

$$\begin{aligned}
P(|\tilde{p}(t) - p(t)| > \varepsilon p(t)) &= P(|a(t) - p(t)N| > \varepsilon p(t)N) \\
&= P(|X - E(X)| > \varepsilon E(X)) \\
&\quad \text{by the simplified Chernoff bound (Section 4.1 in [18])} \\
&\leq e^{-\frac{\varepsilon^2 E(X)}{2}} + e^{-\frac{\varepsilon^2 E(X)}{4}} \\
&\leq 2e^{-\frac{\varepsilon^2 p(t) N}{4}} \qquad (\text{since } p(t) \geq \nu) \\
&\leq 2e^{-\frac{\varepsilon^2 \nu N}{4}} \\
&\leq \eta \qquad (\text{with } N \geq \frac{4}{\varepsilon^2 \nu} \ln \frac{2}{\eta})
\end{aligned}$$

If $p(t) \leq \nu$:

$$\begin{aligned}
P(\tilde{p}(t) > (1+\varepsilon)\nu) &= P(X > (1+\varepsilon)\nu N) \\
&< \frac{E(e^{rX})}{e^{r(1+\varepsilon)\nu N}} \qquad (\text{by Markov ineq.}) \\
&= \frac{e^{(e^t - 1)p(t)N}}{e^{r(1+\varepsilon)\nu N}} \\
&\leq \left(\frac{e^{(e^t-1)}}{e^{r(1+\varepsilon)}} \right)^{\nu N} \qquad (\text{since } p(t) \leq \nu) \\
&\leq \left(\frac{e^{\varepsilon}}{(1+\varepsilon)^{(1+\varepsilon)}} \right)^{\nu N} \qquad (\text{with } r = \ln(1+\varepsilon)) \\
&\leq e^{-\frac{\varepsilon^2 \nu N}{3}} \qquad (\text{with } 0 < \varepsilon < 1) \\
&\leq \frac{\eta}{2}
\end{aligned}$$

□

Now consider a vertex t of the δ-neighborhood arrangement of S. The following lemma states estimate bounds for a vertex of the arrangement.

Lemma A.4. *Let $0 < \varepsilon, \eta, \nu < 1$ be parameters, let S be a sample of $N \geq \frac{c \ln \frac{2}{\eta}}{\varepsilon^2 \nu} + 2$ translation vectors for a suitable constant c, and let $t \in \mathbb{R}^2$ be a vertex of the arrangement of the δ-neighborhoods of vectors in S, then*

- $p(t) \leq \nu \Rightarrow P(\tilde{p}(t) > \varepsilon\nu) \leq \eta$
- $p(t) \geq \nu \Rightarrow P(|\tilde{p}(t) - p(t)| > \varepsilon p(t)) \leq \eta$.

Proof. Let $s_1, s_2 \in S$ be the sample vectors whose δ-neighborhoods intersect in vertex t. Consider the sample $Q = S \setminus \{s_1, s_2\}$. As in Lemma A.2 we can argue that

$$\tilde{p}_Q(t) \leq \tilde{p}_S(t) + \frac{2}{N-2} \qquad \text{and}$$
$$|\tilde{p}_S(t) - p(t)| \leq |\tilde{p}_Q(t) - p(t)| + \frac{2}{N-2} \ .$$

In case $p(t) \leq \nu$

$$\begin{aligned} P(\tilde{p}_S(t) > (1+\varepsilon)\nu) &\leq P(\tilde{p}_Q(t) > (1+\varepsilon)\nu) && \text{(since } \tilde{p}_S(t) \leq \tilde{p}_Q(t)\text{)} \\ &\leq e^{-\frac{\varepsilon^2(N-2)\nu}{3}} && \text{(by Lemma A.1)} \\ &\leq \eta \end{aligned}$$

If $p(t) \geq \nu$:

$$\begin{aligned} P(|\tilde{p}_S(t) - p(t)| > \varepsilon p(t)) &\leq P\left(|\tilde{p}_Q(t) - p(t)| + \frac{2}{N-2} > \varepsilon p(t)\right) \\ &\leq P\left(|\tilde{p}_Q(t) - p(t)| > \frac{\varepsilon}{2} p(t)\right) \qquad \text{(for } N \geq \tfrac{4}{\varepsilon\nu}\text{)} \\ &\leq 2e^{-\frac{(\varepsilon/2)^2\nu(N-2)}{4}} \qquad \text{(by Lemma A.1)} \\ &= 2e^{-\frac{\varepsilon^2\nu(N-2)}{16}} \\ &\leq \eta \end{aligned}$$

□

Since the arrangement of the δ-neighborhoods of sample vectors has at most N^2 vertices, the probability that for any vertex t $|\tilde{p}(t) - p(t)| > \varepsilon$ is at most $N^2 2e^{-\frac{\varepsilon^2\nu(N-2)}{16}}$, which is at most $e^{-\frac{\varepsilon^2\nu(N-2)}{32}}$, and is less than η for $N \geq \frac{32\ln\frac{1}{\eta}}{\varepsilon^2\nu} + 2$.

Let A and B be two shapes modeled by sets of line segments of complexity n, and let D_A, D_B denote the diameter of A and B respectively. The support of $p(t)$ is $T = A \oplus -B$ and has diameter at most $D_A + D_B$. The largest possible area of T is then $\frac{\pi(D_A+D_B)^2}{4}$. If we have a uniform distribution on T, then for any $t \in T$ with distance at least δ from the boundary of T, $p(t) = \frac{4\pi\delta^2}{4\pi(D_A+D_B)^2} = \frac{\delta^2}{(D_A+D_B)^2} = \nu^*$, for circular δ-neighborhood, or $p(t) = \frac{4\delta^2}{\pi(D_A+D_B)^2} = \nu^*$ for square δ-neighborhood. For an arbitrary probability distribution on T exists at least one translation vector t with $p(t) \geq \nu^*$.

Therefore it is sufficient to take a sample S of $N \geq \frac{32\ln\frac{1}{\eta}}{\varepsilon^2\nu^*} + 2$ vectors to get an ε-approximation of $p(t_{\text{opt}})$.

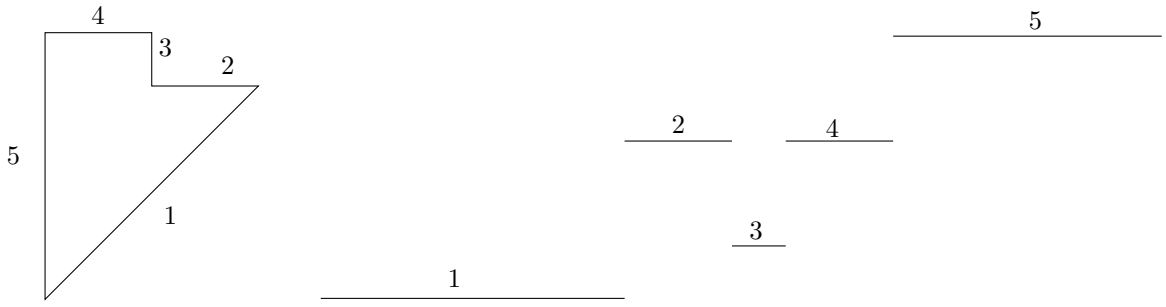
Proof. (Of Theorem 2.2)
With linear time preprocessing we can generate a random point of the shape in time $O(\log n)$, then the time to generate N random points on both shapes is $O(n+N\log n)$. The arrangement of the δ-neighborhoods of N vectors has the complexity $O(N^2)$ and can be computed and traversed in time $O(N^2)$. □

turning_angle_matching_polygon_polyline

Definition

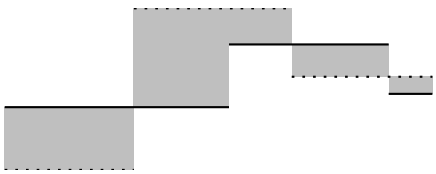
The function *turning_angle_matching_polygon_polyline* computes how a polyline can be best matched against a polygon under certain conditions. The function returns how the best match is achieved and what the distance is at this position. The computed distance is invariant under translation and rotation of the polygonal chain. The distance measure is based on the work of Arkin et al.[1].

The distance between two polylines is defined by means of the turning angle representation. In this representation every segment of a polygon is represented by its length and direction. The direction is measured as the counterclockwise angle with the positive x-axis in radians. This still leaves some freedom: an angle θ is the same as an angle $\theta + 2\pi$. We restrain the representation further by requiring that at every discontinuity (that is, at every vertex), the jump is less than π. Below you can see an example polygon and the associated turning angle representation.



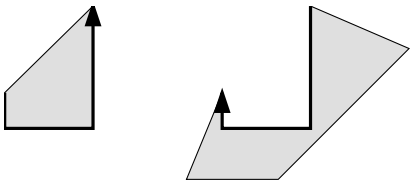
Polygon Turning angle representation of the same polygon

The distance between a chain and a polygon is defined in terms of the distance between their turning angle representations. In the figure below, we see two turning angle representations; one is dotted, the other one solid. The distance between the two is indicated by the gray area. In fact, we take the square of the distance between the segments and integrate: $d(f,g) = \int (f(x) - g(x))^2 dx$, where f and g are the turning angle representations. In the paper by Arkin et al, the square root of this integral is taken as the distance.



For computing the distance between the polyline and the polygon, we rotate the polyline and choose a starting point at the polygon and then compute the distance using the turning angle distance. We have freedom to choose the starting point and the rotation. We choose them such that the distance is minimal. If the length of the polygon is less than the length of the chain, the polyline cannot match the polygon. No distance is computed in that case.

Polygons and polylines can have an orientation. Sometimes we want to take this into account for the matching, but at other times we don't. Both are possible; the choice is made with a flag. For instance, have a look at the polygons below.



The fat part of the boundaries is the same, except for their orientation. They will match perfectly if we don't take orientation into account, but badly if we do.

#include <*tuan_matching.h*>

template <*class PgnSegmentsIter, class ChainSegmentsIter, class SegmentCvt*>
std::pair<*Tuan_chain_info, bool*>

turning_angle_matching_polygon_polyline(PgnSegmentsIter polygon_begin,
PgnSegmentsIter polygon_end,
ChainSegmentsIter chain_begin,
ChainSegmentsIter chain_end,
SegmentCvt segment_converter,
bool reversal_allowed = false)

- *PgnSegmentsIter* is an iterator type with value type that we will refer to as *Segment1*. You can choose any type you like for this, but it should represent a segment. The range from *polygon_begin* to *polygon_end* specifies the segments of the polygon.
- *ChainSegmentsIter* is an iterator type with value type that we will refer to as *Segment2*. You can choose any type you like for this, for instance the same type as Segment1. The range from *chain_begin* to *chain_end* specifies the segments of the chain.
- *SegmentCvt* is the type for a functor that converts a segment to a pair of segment length and segment angle. The return type must be *std::pair*<*double,double*>. It must be able to convert segments of type *Segment1* and of type *Segment2*. That is, *double_pair=segment_converter(seg1);* and *double_pair=segment_converter(seg2);* should both be valid, if *seg1* is of type *Segment1*, *seg2* is of type *Segment2* and *double_pair* is a pair of doubles.
- If the parameter *reversal_allowed* is *true*, the matching will be tried both in the original orientation of the chain and with the chain reversed. The best match is reported.
- The return value is a pair of a *Tuan_chain_info* and a *bool*. The first part gives the distance and details on how the match is made. See the documentation of the for more information. The second part states if the match was made in the original orientation (*true*) or not. The second part will always be *true* if *reversal_allowed* was *false*.

  If the polygon has a smaller length than the chain, a negative distance is reported to indicate that the distance is not well defined.

Example

In this example we present one way of representing polylines and polygons and computing the best match based on this representation. Keep in mind that this particular representation is just one of many possible ones. Its main value is that it is concise.

We represent polygons and polylines by means of an array of coordinates (doubles). For polygons we repeat the first point, that is, the first two coordinates, at the end. This makes it easy to iterate over

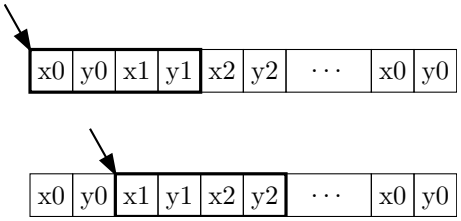
the segments. Actually, we will use a vector to store the coordinates. But we make use of it in an array like fashion.

We define a *const_segment_iterator* with the help of the iterator facade of the Boost Iterator library. A segment iterator points into the array. The member variable *m_coord1* holds this pointer. The iterator can be initialised with a pointer into an array or with a vector iterator.

When we move the iterator one step, we move two steps in the array (x and y coordinate). This makes the previous end vertex the new start vertex. This is implemented in the function *increment*.

Equality testing is done in the function *equal*. We test the embedded pointers for equality.

Dereferencing the iterator –implemented by *dereference*– yields a segment. A segment is represented by a pointer to an array of 4 coordinates. This is the same pointer as the one stored in the iterator, but with a different role.



In the figure above we see the pointer in both of its roles. The two different positions indicate the iteration over the segments. The fat box indicates the current segment.

```
#include <vector>
#include <boost/iterator/iterator_facade.hpp>
#include "boost/tuple/tuple.hpp"

using std::vector;

class const_segment_iterator
  : public boost::iterator_facade<
       const_segment_iterator
     , double const * const    // the value type of the iterator
     , boost::forward_traversal_tag    // make a forward iterator
    >
{
  public:
    const_segment_iterator() : m_coord1(0) {}
    explicit const_segment_iterator(double const* p)
      : m_coord1(p) {}
    explicit const_segment_iterator(vector<double>::const_iterator p)
      : m_coord1(&(*p)) {}
  private:
    friend class boost::iterator_core_access;
    // The friend declaration is needed to give iterator_facade access to the
    // following private functions.
    // The iterator_facade class makes the public iterator interface,
    // that is, operator++, operator*, operator-> etc.
    void increment() { m_coord1 += 2; }
    bool equal(const_segment_iterator const& other) const
    { return this->m_coord1 == other.m_coord1; }
```

```
    double const * const& dereference() const { return m_coord1; }
    double const * m_coord1;
};

class Converter {
public:
    std::pair<double,double> operator()(double const * array4) {
        return to_tuan_segment_rep(array4[2]-array4[0],array4[3]-array4[1]);
    }
};

// MyChain is a very minimal class for representing polygonal chains.
class MyChain {
    vector<double> m_rep;    // store for the coordinates
  public:
    // The constructor copies the coordinates of an input range,
    // possibly repeating the first point (for closed chains).
    template <class Iter>
    MyChain(Iter coords_begin, Iter coords_end, bool repeat_first = false)
    : m_rep(coords_begin, coords_end)
    {
        if (repeat_first) {
            m_rep.push_back(m_rep[0]);
            m_rep.push_back(m_rep[1]);
        }
    };
    const_segment_iterator segments_begin() const
    { return const_segment_iterator(m_rep.begin());}
    // The segment iterator is at its end when the last point is reached,
    // because a segment needs two points.
    // A point has two coordinates, hence the -2.
    const_segment_iterator segments_end() const
    { return const_segment_iterator(m_rep.end()-2);}
};

typedef MyChain MyPolygon;

template <class T, size_t N>
T* end(T (&a)[N])
{ return a+N;}

void compute_match(MyPolygon const &pgn, MyChain const &chain)
{
    Tuan_chain_info matching_info;
    double dist;
    boost::tie(matching_info,dist) = tuan_polygon_single_chain_matching(
        pgn.segments_begin(), pgn.segments_end(),
        chain.segments_begin(),chain.segments_end(),
        Converter());
}

void chain_test()
{
    double pgn_coords[] = { 0.0,1, 10,0, 10,3.5, 0.9,70, };
```

```
    double chain_coords[] = { 0,0, 1,3, 6,2.5,};
    MyPolygon pgn(pgn_coords, end(pgn_coords),true);
    MyChain chain(chain_coords, end(chain_coords));
    compute_match(pgn,chain);
}
```

turning_angle_matching_polygon_polylineseq

Definition

The function *turning_angle_matching_polygon_polylineseq* is like the function *turning_angle_matching_polygon_polyline*, except that it operates on a sequence of polylines instead of a single polyline. The extension is such that:

- Every chain can be rotated independently of the others.
- The polylines are matched against non overlapping parts of the polygon. Those polygon parts appear ordered along the polygon in the same order as the polylines in the collection.
- As a consequence, the total length of the polylines must be equal or less than the perimeter of the polygon.

#include <tuan_matching.h>

template <class PgnSegmentsIter, class ChainsIter, class SegmentCvt, class ChainInfoIter>
boost::tuple<double, ChainInfoIter, bool>

turning_angle_matching_polygon_polylineseq(PgnSegmentsIter polygon_begin,
PgnSegmentsIter polygon_end,
ChainsIter chains_begin,
ChainsIter chains_end,
SegmentCvt segment_converter,
ChainInfoIter chain_info_output,
bool reversal_allowed = false)

- *PgnSegmentsIter* is an iterator type with value type that we will refer to as *Segment1*. You can choose any type you like for this, but it should represent a segment. The range from *polygon_begin* to *polygon_end* specifies the segments of the polygon.
- *ChainsIter* is an iterator type with value type some container of *Segment2*. *Segment2* can be any type of your liking. A container must be a container in the sense of the C++standard. At least it should have member functions *begin()* and *end()* and a *value_type* type. Beware, here we talk about *ChainsIter::value_type::value_type*. This value type should be *Segment2*.
- *SegmentCvt* is the type for a functor that can convert a *Segment1* and a *Segment2* to a pair of segment length and segment angle. The return type must be *std::pair<double,double>*.
- *ChainsIter* is an output iterator type with value type *Tuan_chain_info*. Exactly the number of polylines values will be written to it.
- The return value is a tuple of three elements. See the boost library for a description of tuples. The first tuple element is a double that gives the total distance. It is the sum of the distances of the individual polylines. The second tuple element returns the *chain_info_output* iterator after all updates were done. This will point one past the end of the sequence. The third element indicates if the match was made in the original orientation (*true*) or the reversed orientation (*false*).

See Also

turning_angle_matching_polygon_polyline

Example

The example shown here builds on the example of *turning_angle_matching_polygon_polyline*. We add a class that is a range of segment iterators, having a begin and end member.

```
class segment_iter_range {
public:
    segment_iter_range(const_segment_iterator begin, const_segment_iterator end)
        :m_begin(begin), m_end(end){}
        typedef const_segment_iterator::value_type value_type;
    typedef const_segment_iterator const_iterator;
    typedef const_segment_iterator iterator;

    const_segment_iterator const begin() const {return m_begin;}
    const_segment_iterator const end() const {return m_end;}
private:
    const_segment_iterator m_begin, m_end;
};

void compute_distance(MyPolygon const &pgn, vector<segment_iter_range> const &chains)
{
    using boost::tuples::ignore;
    vector<Tuan_chain_info> matching_info(chains.size());
    double dist;
    boost::tie(dist, ignore, ignore)=
        tuan_polygon_chains_matching(pgn.segments_begin(), pgn.segments_end(),
            chains.begin(),chains.end(), Converter(), matching_info.begin());
}

void chains_test()
{
    double pgn_coords[] = { 0.0,1,10,0,10,3.5,0.9,70, };
    double chain_coords1[] = { 0,0,1,3,6,2.5,};
    double chain_coords2[] = {0,0, 0,30, 1,30, };
    MyPolygon pgn(pgn_coords, end(pgn_coords),true);
    MyChain chain1(chain_coords1, end(chain_coords1));
    MyChain chain2(chain_coords2, end(chain_coords2));
    vector<segment_iter_range> chains;
    chains.push_back(segment_iter_range(chain1.segments_begin(), chain1.segments_end()));
    chains.push_back(segment_iter_range(chain2.segments_begin(), chain2.segments_end()));
    compute_distance(pgn,chains);
}
```

to_tuan_segment_rep

Definition

The function *to_tuan_segment_rep* is a utility function that can aid in converting a custom representation of a segment to a turning angle representation.

#include <*tuan_matching.h*>

std::pair<*double,double*>

to_tuan_segment_rep(double dx, double dy)

Example

In this example we show a converter that can convert a segment in two representations to the turning angle representation. The first representation is an explicit representation of segments in a class, where the coordinates are integers. In the second representation, the coordinates are doubles in an array. The first array elements are the x and y coordinate of the begin point, the next two elements are the x and y coordinate of the end point of the segment.

```
#include "tuan_matching.h"

struct IntPoint
{ int x,y; };

struct IntSegment
{ IntPoint start, end;};

class MyConverter {
public:
    std::pair<double,double> operator()(IntSegment const &seg) {
        return to_tuan_segment_rep(seg.end.x-seg.start.x,seg.end.y-seg.start.y);
    }
    std::pair<double,double> operator()(double const *segp) {
        return to_tuan_segment_rep(segp[2]-segp[0], segp[3]-segp[1]);
    }
};

void convert_test()
{
    IntSegment seg1;
    seg1.start.x = 3;
    seg1.start.y = 1;
    seg1.end.x = 6;
    seg1.end.y = 5;
    double seg2[] = {3.0,1.0,6.0,5.0};
    MyConverter converter;
    std::pair<double,double> turnrep1 = converter(seg1);
    std::pair<double,double> turnrep2 = converter(seg2);
}
```

Tuan_chain_info

Definition

The class *Tuan_chain_info* holds information how a chain is matched.

#include <*tuan_matching.h*>

Creation

Tuan_chain_info ci; default constructor.

Query Functions

double *ci.distance()* Distance contributed by this chain to total distance. It is the distance of the chain, rotated over *matching_angle()*, to the part of the polygon perimeter between *matching_start()* and *matching_end()*.

double *ci.matching_angle()*

Counterclockwise angle over which the chain is rotated to get the match. The angle is in radians.

void *ci.matching_start(std::size_t &segment_index, double &segment_ratio)*

Indicates what part of the polygon is matched. The start of the chain is matched to a point on segment *segment_index* of the input. The *segment_ratio* indicates the exact point on the segment. It ranges from 0 (inclusive) to 1 (not inclusive). The value 0 indicates the start of the segment, the value 1 indicates the end of the segment. Other values are linearly interpolated.

void *ci.matching_end(std::size_t &segment_index, double &segment_ratio)*

Describes where the point matching the end point of the chain is located on the polygon.

See Also

tuan_polygon_chains_matching, *tuan_polygon_single_chain_matching*.

Bibliography

[1] Esther M. Arkin, L. P. Chew, D. P. Huttenlocher, K. Kedem, and Joseph S. B. Mitchell. An efficiently computable metric for comparing polygonal shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(3):209–216, 1991.