ELSEVIER

Pattern Recognition Letters 18 (1997) 1435–1442

Pattern Recognition Letters

Finding rotational symmetries by cyclic string matching

Josep Lladós [a,*], Horst Bunke [b,1], Enric Martí [a,2]

[a] *Computer Vision Center, Dep. Informàtica. Universitat Autònoma de Barcelona, 08193 Bellaterra (Barcelona), Spain*
[b] *Institut für Informatik und Angewandte Mathematik. University of Bern, Neubrückstr. 10, CH-3012 Bern, Switzerland*

Received 4 October 1996; revised 4 September 1997

Abstract

Symmetry is an important shape feature. In this paper, a simple and fast method to detect perfect and distorted rotational symmetries of 2D objects is described. The boundary of a shape is polygonally approximated and represented as a string. Rotational symmetries are found by cyclic string matching between two identical copies of the shape string. The set of minimum cost edit sequences that transform the shape string to a cyclically shifted version of itself define the rotational symmetry and its order. Finally, a modification of the algorithm is proposed to detect reflectional symmetries. Some experimental results are presented to show the reliability of the proposed algorithm. © 1997 Elsevier Science B.V.

Keywords: Rotational symmetry; Reflectional symmetry; String matching

1. Introduction

Symmetry is one of the shape features that is often used in object recognition. Some computer vision systems use hashing techniques to recognize a given shape in an input image and symmetry-based indexing functions are a useful tool used in this kind of applications. Other computer vision applications, such as matching techniques that use registration, need to detect the orientation of a shape before it is matched with a model shape. Finding the principal axes of a shape is a useful technique to detect its orientation, but it does not work properly when the shape is rotationally symmetric. For symmetric shapes, special procedures must be applied. Therefore, it is necessary to know whether a shape is symmetric or not. Symmetry is also useful to recover a planar symmetric figure from an image without the need of models.

A shape is said to be rotationally symmetric if it becomes identical to itself after being rotated by a certain angle. There are different approaches to detect rotational symmetries. Tsai and Chou (1991) used high-order moment functions to find *n*th-order principal axes of the shape. In other works (e.g., (Pei and Lin, 1992; Pei and Liuo, 1994)) Fourier descriptors are used to normalize rotationally symmetric shapes. Yip et al. (1994) developed a Hough transform algorithm for the detection of rotational symmetry in the presence of noise and occlusion.

String matching is a widespread technique used in the recognition of 2D shapes represented by their boundaries. Some examples can be found in (Bunke

* Corresponding author. E-mail: josep@cvc.uab.es.
[1] E-mail: bunke@iam.unibe.ch.
[2] E-mail: enric@cvc.uab.es.

PII S0167-8655(97)00144-X

and Buhler, 1993; Bunke and Glauser, 1993; Maes, 1991; Tsay and Tsai, 1989). The basic idea is to represent the boundary of the input and the prototype shapes as strings, i.e. sequences of symbols, and use an approximate string matching algorithm (see (Wagner and Fischer, 1974)) to compute their similarity. String matching methods are easy to implement and have a low computational complexity. A key observation is that the boundary of a rotationally symmetric shape consists of a sequence of identical substrings. In this paper we will use cyclic string matching as a new method to detect whether a given 2D shape is rotationally symmetric or not. For any rotationally symmetric shape, the algorithm will also find the subshape, i.e. substring, that is repeated along the boundary.

The remainder of this paper is organized as follows: Section 2 summarizes the relevant string matching algorithms and their application to 2D shape recognition. In Section 3 we introduce some modifications of the cyclic string matching algorithm in order to detect rotational symmetries. Experimental results are presented in Section 4. Finally, conclusions are given in Section 5.

2. Boundary matching by string edit distance

2.1. Definitions

Let Σ be an alphabet of symbols and $X = x_1 \dots x_n \in \Sigma^*$, $Y = y_1 \dots y_m \in \Sigma^*$ two strings; $n, m \geq 0$. The distance between X and Y is defined in terms of elementary edit operations required to transform X into Y with minimum cost. Conventionally, three edit operations are defined: (1) *substitution* of a symbol $x \in \Sigma$ in X by a symbol $y \in \Sigma$ in Y, denoted as $x \to y$; (2) *insertion* of a symbol $x \in \Sigma$ in Y, denoted as $\lambda \to x$; (3) *deletion* of a symbol $x \in \Sigma$ in X, denoted as $x \to \lambda$; where λ denotes the empty string. An *edit sequence* S is defined as an ordered sequence of edit operations $s_1, \dots, s_p$. Let c be a cost function that assigns a non-negative real number $c(s)$ to each edit operation s. The cost of an edit sequence S is defined as $c(S) = \sum_{i=1}^{p} c(s_i)$. The *edit distance* between the strings X and Y is defined by $d(X,Y) = \min\{c(S)$: S is a sequence of edit operations transforming X into $Y\}$. Let $X^{(i)}$ be the string X cyclically shifted by i positions, i.e. $X^{(i)} = x_i \dots x_n x_1 \dots x_{i-1}$. Two strings X and X' of length n will be called *equivalent* if $X' = X^{(i)}$ for some $1 \leq i \leq n$. This defines an equivalence relation on Σ^*. A *cyclic string* $\overline{X}$ is defined as the equivalence class of a string X. The *cyclic edit distance* between two strings X and Y is defined as $d_c(X,Y) = \min\{d(X^{(i)},Y^{(j)}) \mid i = 1, \dots, n, j = 1, \dots, m\}$.

2.2. Shape representation using strings

The closed boundary of a shape can be represented by means of a string. It is important to choose an appropriate set of primitives to be used as symbols. Bunke and Buhler (1993) proposed a pixel-based scheme where each symbol represents the curvature at each boundary point. Many approaches have been proposed to approximate a boundary by a polygon (e.g. (Maes, 1991; Tsai and Yu, 1984; Tsay and Tsai, 1989)). The use of a discrete set of symbols prevents, in some cases, a shape from being accurately described by a polygon. Tsai and Yu (1984) solved this problem by means of *attributed strings*, i.e. strings whose symbols are augmented by numerical values as, for example, the length or direction of line segments. Another problem with the representation of 2D shapes by strings is the starting symbol of the string, which is not uniquely defined if the contour is closed. This problem can be solved by means of cyclic strings instead of conventional linear ones.

In this paper we assume, without loss of generality, that shapes have been polygonally approximated. Each line segment s_i is represented by its length l_i and its angle ϕ_i. Traditionally, the angle attribute is computed with regard to a reference line segment but this may cause difficulties if the reference line segment has not been appropriately selected. To avoid this problem and to obtain a representation that is invariant under rotation, we define ϕ_i as the angle formed between s_i and the previous line segment s_{i-1}. The costs of the edit operations are defined as a weighted sum of an angle cost and a length cost, similarly to those used by Tsay and Tsai (1989). In

the experiments described in Section 4, we used the following cost function:

$$c(\lambda \to x) = K_i + l_x/l_X,$$

$$c(x \to \lambda) = K_d + l_x/l_X,$$

$$c(x \to y) = |\phi_x - \phi_y|/2\pi + |l_x/l_X - l_y/l_Y|,$$

where l_X and l_Y are the respective lengths of the strings which x and y belong to. K_i and K_d are two are two preselected constants between 0 and 1.

2.3. The string matching algorithm

The well-known algorithm of Wagner and Fischer (1974) computes $d(X,Y)$ in $O(nm)$ time and space. It is a dynamic programming procedure that computes the elements of an $(n+1)\times(m+1)$ matrix D. The value $D(i,j)$ corresponds to the edit cost $d(X',Y')$ where $X' = x_1 \ldots x_i$ and $Y' = y_1 \ldots y_j$. This value $D(i,j)$ is calculated, by means of cost minimization, from its three predecessors and the corresponding edit operation, i.e., going from $D(i-1, j-1)$ to $D(i,j)$ (substitution of x_i by y_j), from $D(i-1,j)$ to $D(i,j)$ (deletion of x_i), or from $D(i, j-1)$ to $D(i,j)$ (insertion of y_j). For each matrix element a pointer to the selected predecessor is stored in order to find the edit sequence, in a backward trace, that corresponds to $d(X,Y)$.

A common problem in string matching for 2D shape recognition is that shape boundaries in images often contain some distortions. To overcome this problem Tsai and Yu (1984) extended the three basic edit operations by a *merging operation*. Let $X_{i,j} = x_i \ldots x_j$ be a substring of string $X = x_1 \ldots x_n$ $(1 \le i \le j \le n)$. We define $X_{i,j} \to x'$ as the merging operation that transforms the substring $X_{i,j}$ into a symbol $x' \in \Sigma$. This operation allows to calculate *block substitutions*, as did Bunke and Buhler (1993), i.e. a substitution of a whole sequence of symbols by another sequence of symbols $(X_{i,j} \to Y_{k,l})$. To perform this substitution we merge $X_{i,j}$ into a symbol x' and the substring $Y_{k,l}$ into a symbol y'. Then, we only have to calculate the substitution cost $c(x' \to y')$. It is also necessary to define merging costs $c(X_{i,j} \to x')$ and $c(Y_{k,l} \to y')$. In the experiments described in Section 4, string $x_1 x_2$ is merged into a symbol x' by defining $l_{x'}$ and $\phi_{x'}$ attributes in the following way:

$$l_{x'} = l_1 + l_2,$$

$$\phi_{x'} = (\phi_1 l_1 + \phi_2 l_2)/(l_1 + l_2).$$

Once the symbol x' has been defined, the merging cost is computed by the following equation:

$$c(x_1 x_2 \to x') = \frac{|\phi_{x'} - \phi_1|}{2\pi}\frac{l_1}{l_{x'}} + \frac{|\phi_{x'} - \phi_2|}{2\pi}\frac{l_2}{l_{x'}},$$

which, taking into account the definition of $l_{x'}$ and $\phi_{x'}$, can be reduced to the following equation:

$$c(x_1 x_2 \to x') = \frac{|\phi_2 - \phi_1|}{\pi}\frac{l_1 l_2}{l_{x'}^2}.$$

Physically we can assume that x' is a symbol which substitutes the concatenation of x_1 and x_2, and whose orientation is the average between the orientations of x_1 and x_2 weighted by their length. That is, the new merged segment is such a segment having the same distance to the original ones. Thus, the cost of x_1 to x' and x_2 to x' are identical. The merging cost stated in the previous equations estimates how x_1 and x_2 differ from x' (notice that this cost would be zero if the two merged symbols were collinear). The previous equations can be iteratively extended to merging substrings longer than two symbols. Notice that the substitution, deletion and insertion are particular cases of the block substitution using the merging operation.

As it was mentioned before, the initial symbol of a string that represents a closed contour is not uniquely defined. To solve this problem in a brute-force way, one can compute the edit distance between one string and all its cyclic shifts and then take the minimum. This procedure would take $O(nm^2)$ time. Bunke and Buhler (1993) used a cyclic string matching procedure that runs in $O(nm)$. It is based on the computation of the classic string edit distance between X and Y^2, where Y^2 is the string YY. It was shown that $d_c(X,Y)$ is equal to $d(X,Y')$ where Y' is the substring of Y^2 that is most similar to X. Fig. 1 shows this cyclic string matching algorithm considering also the merging operation. Using the string representation described in Section 2.2, the algorithm of Fig. 1 allows to compute the similarity

Algorithm cyclic_string_matching(X,Y)
• **input:** two strings $X = x_1 \dots x_n$ and $Y = y_1 \dots y_m$
• **output:** the cyclic edit distance $d_c(X,Y) = d(X,Y')$ such that Y' is the substring of Y^2 with the smallest edit distance to X
0. **begin**
1. $D(0,0).cost := 0$;
2. **for** $i = m+1$ **to** $2m$ **do** $y_i := y_{i-m}$;
3. **for** $i = 1$ **to** m **do** $D(0,i).cost := 0$;
4. **for** $i = m+1$ **to** $2m$ **do** $D(0,i).cost := \infty$;
5. **for** $i = 1$ **to** n **do** $D(i,0).cost := D(i-1,0).cost + c(x_i \to \lambda)$;
6. **for** $i = 1$ **to** n **do**
7. **for** $j = 1$ **to** $2m$ **do**
8. **begin**
9. $D(i,j).cost := min\{D(i-k,j-l) + c(x_{i-k+1,i} \to x') + c(y_{j-l+1,j} \to y') +$
10. $c(x' \to y') : k,l = 1,\dots,M\}$;
11. $D(i,j).ptr = (i-k', j-l')$ where k' and l' are the values of k and l where
12 the minimum in the previous statement occurs;
13. **end**
14. $d_c(X,Y) := min\{D(n,i),\ i = m+1,\dots,2m\}$;
15. **end.**

Fig. 1. The cyclic string matching algorithm.

of two shapes invariant under translation, rotation and scaling. The parameter M in line 10 of the algorithm was set to 8 in the experiments described in Section 4.

3. String-matching for rotational symmetry detection

Given a 2D shape represented by an attributed string as described in Section 2.2, we can use the cyclic string matching algorithm to find whether it is rotationally symmetric or not. In terms of the string representation of the shape, we say that a string $X = x_1 \dots x_n$ represents a rotationally symmetric shape if $Z^o \in \overline{X}$ for some substring Z of X and $1 < o \le n$, with o being called the *order of symmetry* of X. As can be seen below, the rotational symmetries of X can be found by computing the cyclic edit distance between X and itself, and analyzing the corresponding $D(i,j)$ matrix.

Let S and S' be two edit sequences transforming a string X into another string Y. We say that S and S' are *independent edit sequences* if $S \cap S' = \emptyset$, where $S \cap S'$ denotes the common subsequence to S and S'. Fig. 3 shows all edit sequences in $D(i,j)$ after *cyclic_string_matching*(X,X) has been run, with X being the string representing the test shape of Fig. 2. The elements $D(n,i)$, for $i = n+1,\dots,2n$, have been grouped into sets P_k. Each P_k in Fig. 3 represents a set of edit sequences from X to itself that share a common subsequence ($1 \le k \le K \le n$, where K is the number of sets obtained). In this figure, the cost of each edit sequence is given at the bottom of the corresponding $D(n,i)$ position ($i = n+1,\dots,2n$). The minimum cost edit sequence of each set P_k (printed in bold face) is chosen as a representative of this set. Let S_k be the representative of the set P_k. Then S_k defines a match between X and a rotated version of itself. Notice that not all edit sequences S_k necessarily represent a valid match. Therefore, we select only those sequences whose edit cost is below a given threshold T. Let $\mathcal{S}$ be the set of selected edit sequences, $\mathcal{S} = \{S_{k_i} \mid i = 1,\dots,N,\ N$

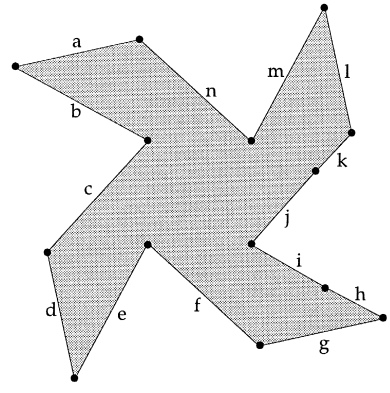


Fig. 2. Test shape $X = abcdefghijklmn$.

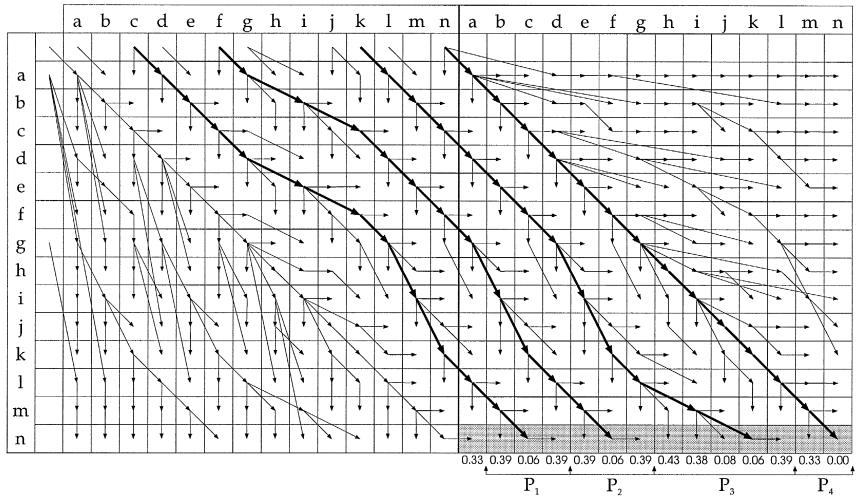


Fig. 3. $D(i,j)$-matrix for the cyclic string matching algorithm.

$\leq K, c(S_{k_i}) \leq T\}$, where N is the number of selected edit sequences. N also defines the order of symmetry of X. Once we have found $\mathscr{S}$, it only remains to determine the substrings Z_i of X, $1 \leq i \leq N$ such that $X = Z_1 Z_2 \ldots Z_N$ and $d(Z_i, Z_j) \leq T$ for all $1 \leq i,j \leq N$. These substrings define the rotationally symmetric subshape. To obtain it, we take a symbol $x_i \in X$ and look for its matching symbol for each edit sequence of $\mathscr{S}$. For example, in Fig. 3, the symbol n matches c, f, jk and n. Thus, the rotationally symmetric substrings are $Z_1 = cde$, $Z_2 = fghi$, $Z_3 = jklm$ and $Z_4 = nab$. Based on these considerations, we add the following lines to the algorithm of Fig. 1 in order to detect rotational symmetries:

1. **for** $i = 1$ **to** $2m$ **do** $D(0,i).begin := (0,i)$; /* to store the starting point of each edit sequence */
2. $D(i,j).begin = D(i-k', j-l').begin$
3. $k := 1$; $P_k := \emptyset$; $j := m+1$;
4. **while** $D(n,j).begin = D(n,m).begin$ **do** $j := j+1$; /* seek the first sequence of P_1 */
5. $P_1 := \{S$ starting at $D(n,j).begin$ and ending at $D(n,j)\}$;
6. **for** $i = j+1$ **to** $2m$ **do**
7. **begin**
8. **if** $D(n,i).begin \neq D(n,i-1).begin$ **then**
9. **begin**
10. choose S_k as the minimum cost edit sequence of P_k;
11. $k := k+1$; $P_k := \emptyset$;
12. **end**
13. $P_k := P_k \cup \{S$ starting at $D(n,i).begin$ and ending at $D(n,i)\}$;
14. **end**
15. $\mathscr{S} = \{S_{k_i}, c(S_{k_i}) \leq T\}$;
16. **for** each $S_{k_i} \in \mathscr{S}$ **do** take, as starting point of a rotationally symmetric substring Z_i,
17. the substring of X^2 that matches, in S_{k_i}, the symbol $x_n \in X$.

For each position of the matrix $D(i,j)$ we add a pointer to the starting point of the edit sequence that passes through this position. This allows to keep trace of the independent edit sequences when the matrix $D(i,j)$ is filled in. Lines 14.x determine the rotationally symmetric substrings Z_i as it has been explained above. Intuitively, the algorithm iterates from $D(n,m)$ to $D(n,2m)$ looking for the independent minimum cost edit sequences ending at these positions of the matrix. Afterwards, only those edit sequences whose edit cost is below a given threshold T are kept. The symbols of X^2 mapped from the symbol $x_n \in X$ within two consecutive edit sequences define a rotationally symmetric substring.

Not only does the previous algorithm detect rotational symmetries but it can also, by slightly modifying it, find reflectional symmetries. Let X^{-1} be the *reverse string* of a string X, i.e. $X^{-1} = x_n, \ldots, x_1$.

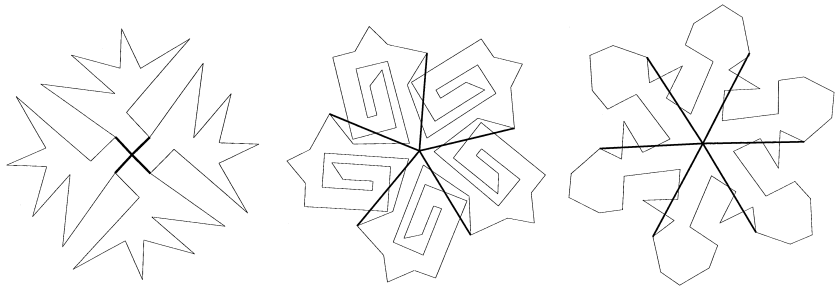


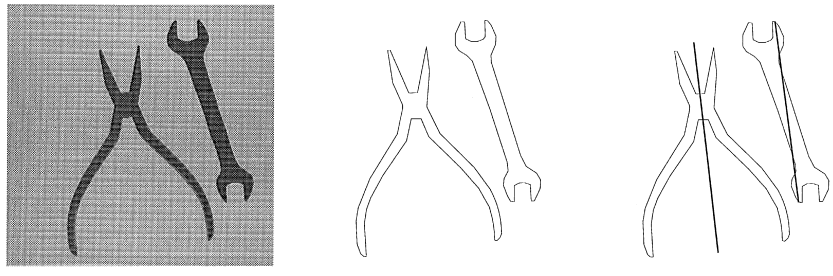
Fig. 4. Rotational symmetries detection in shapes of order 4, 5 and 6 respectively.

Then a shape represented by a string $X = x_1, \ldots, x_n$ is reflectionally symmetric if there exists a substring Z of X such that $X = ZZ^{-1}$. It is possible to find out if a string X is reflectionally symmetric by computing *cyclic_string_matching*(X, X^{-1}). X will be reflectionally symmetric if $d_c(X, X^{-1}) \leq T'$, for a given threshold T'. Analyzing the minimum cost edit sequence returned by the cyclic string matching procedure, the substring Z can be obtained and the medial axis of X determined.

4. Results

Fig. 4 shows the results obtained by the algorithm after it was applied to shapes of different orders of symmetry. Although they are ideal shapes, the polygonal approximation results in a slightly distorted boundary string. Thus, the threshold T has been set to a low nonzero value ($T = 0.15$ in the examples). Lines in bold face connect the center of gravity of each shape with the starting point of each rotationally symmetric substring. The starting point of these substrings depends on the starting point of the whole string. The same results would be obtained for any other higher threshold value because, in the reported examples, it is the minimum value which makes all the edit sequences in the matrix $D(i,j)$ to be selected. For smaller threshold values, some edit sequence might not be selected and, thus, an erroneous order of symmetry would be obtained.

The symmetry detection algorithm has also been applied to the real image of Fig. 5(a). First, we have extracted the boundaries using a morphological gradient operator. Fig. 5(b) shows the polygonal approximation of the shapes of Fig. 5(a). Finally, Fig. 5(c) shows the result: the algorithm has found a rotational symmetric shape, and a reflectional symmetric shape of order 2. Notice that neither of two shapes is exactly symmetric but they are both symmetric with respect to a given threshold (in this case, we have set $T = 0.8$). This threshold allows to find both symme-



(a) (b) (c)

Fig. 5. Detection of symmetries in a real image: (a) original image with boundaries overlayed (b) polygonal approximation of shape boundaries, (c) result after finding one reflectional and one rotational symmetry.

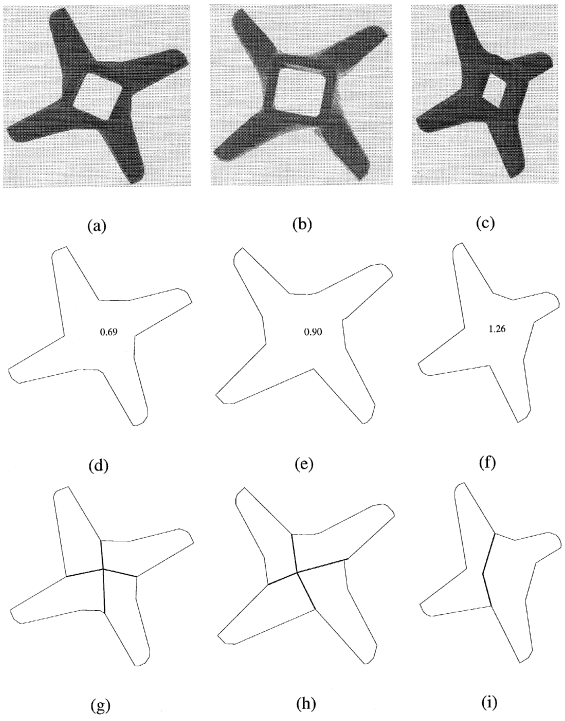


Fig. 6. Symmetry under distortion: (a,b,c) Input images with different degrees of distortion, (d,e,f) polygonal approximation of external boundaries and degree of symmetry distortion obtained, (g,h,i) rotationally symmetric substrings.

tries despite the inaccurate polygonal approximation and, in case of the wrench, a difference in the length of the two sides.

The reliability of the method under different degrees of distortion can be assessed in Fig. 6. An iterative process has been designed applying, at each step, the symmetry detection algorithm with an increasing value of the threshold T. This allows to detect, at each step, shapes with an increasing degree of distortion measured in terms of the string edit distance for symmetry detection. Fig. 6(a) and Fig. 6(b) show two images of an object acquired under different illumination conditions, and Fig. 6(c) shows a skewed version of the object. The results (polygonal approximation and symmetry distortion degree) are given in Fig. 6(d), Fig. 6(e) and Fig. 6(f). Finally, Fig. 6(g), Fig. 6(h) and Fig. 6(i) show the corresponding rotational symmetric substrings. Notice in Fig. 6(i) that the symmetry with minimum distortion degree for this image has order 2 instead of 4 as it would be expected at first glance. These figures also illustrate how the variation in the selected threshold value T may influence on the results.

5. Conclusion

In this paper, the problem of finding rotationally symmetric shapes has been studied. Shapes are polygonally approximated and then represented by attributed strings. The cyclic string matching algorithm has been extended to detect rotationally symmetric shapes of any order of symmetry. It computes a matching between two identical instances of the string representing the shape under investigation. The algorithm runs in $O(n^2)$ where n is the length of the string that represents the shape. It can be used to determine the order of symmetry and the substring that is repeated along the boundary defining the symmetry. If a rotationally symmetric shape is used as a prototype shape in a pattern recognition system, it is sufficient to use the repeated substring as a model and match it with substrings of the input shape. Thus the computational load of the recognition task can be reduced. The cyclic string matching algorithm can also be used to detect reflectional symmetries. In this case, we have to apply the algorithm to the boundary string and its reverse, obtaining the medial axis of the shape as a result. Experimental results of both cases, rotational and reflectional symmetries, show that the algorithm yields reliable results on different images.

The method works for ideal shapes but also in presence of distortions. The admissible degree of distortion under which a shape is still considered symmetric, can be controlled by a single parameter.

References

Bunke, H., Buhler, U., 1993. Applications of approximate string matching to 2D shape recognition. Pattern Recognition 26 (12), 1797–1812.

Bunke, H., Glauser, T., 1993. Viewpoint independent representation and recognition of polygonal faces in 3D. IEEE Trans. Robotics and Automation 9 (4), 457–463.

Maes, M., 1991. Polygonal shape recognition using string-matching techniques. Pattern Recognition 24 (5), 433–440.

Pei, S.C., Lin, C.N., 1992. Normalization of rotationally symmetric shapes for pattern recognition. Pattern Recognition 25 (9), 913–920.

Pei, S.C., Liuo, L.G., 1994. Automatic symmetry determination and normalization for rotationally symmetric 2D shapes and 3D solid objects. Pattern Recognition 27 (9), 1193–1208.

Tsai, W.H., Chou, S.L., 1991. Detection of generalized principal axes in rotationally symmetric shapes. Pattern Recognition 24 (2), 95–104.

Tsai, W.H., Yu, S.S., 1984. Attributed string matching with merging for shape recognition. In: Proc. 7th. ICPR, Montreal, Canada, pp. 1162–1164.

Tsay, Y.T., Tsai, W.H., 1989. Model-guided attributed string matching by split-and-merge for shape recognition. Internat. J. Pattern Recognition and Artif. Intell. 3 (2), 159–179.

Wagner, R.A., Fischer, M.J., 1974. The string-to-string correction problem. J. ACM 21 (1), 168–173.

Yip, R.K.K., Lam, W.C.Y., Tam, P.K.S., Leung, D.N.K., 1994. A Hough transform technique for the detection of rotational symmetry. Pattern Recognition Letters 15, 919–928.