Contour Based Shape Retrieval

Levente Kovács

Distributed Events Analysis Research Group
Computer and Automation Research Institute, Hungarian Academy of Sciences
Kende u. 13-17, 1111 Budapest, Hungary
`levente.kovacs@sztaki.hu`
`http://web.eee.sztaki.hu`

Abstract. This paper presents a contour-based indexing and retrieval method for content-based image/video retrieval applications. It is based on extracting closed contours, smoothing the contour, indexing with a variation of BK-trees, and using a turning function metric for data comparison. The method is very lightweight, fast and robust - the goal being retaining close to realtime speeds for real applicability. We provide evaluation data showing that the method performs well and fast, and is suitable for inclusion into content based retrieval systems as a descriptor for recognition of in-frame objects and shapes.

1 Introduction

Content-based media retrieval deals with a broad area of indexing and searching among audio-visual data. This includes example-based retrievals of image and video data. The basis of high level, semantic searches is always a combination of low level features and descriptors, each of which concentrates on a representative local or global feature, on which higher level semantics can be built. One of such lower level descriptors is the representation of extracted object contours and/or shapes, aiding the indexing and retrieval of similar shapes and objects. Query formulations based on shape descriptors enable users to find specific objects, and also the possibility of shape-based classification, e.g. target recognition for tracking, signaling the appearance of certain objects on a scene, and so on.

In this paper we will present a very fast, lightweight, yet robust and easy-to-modularize shape retrieval approach, based on contour extraction, denoising/smoothing the contour-lines, indexing into a quickly searchable tree structure, and retrieval results showing high precision. The goal of the presented method is to be used as an additional low-level descriptor in a high level content-based retrieval engine [1]. The foremost purpose is to be able to automatically categorize newly extracted shapes into pre-existing classes, enabling the automatic recognition of scene elements, thus aiding the higher level understanding of image/scene contents.

Traditionally, contours/shape descriptors have been extracted and compared with a series of methods, including Hidden Markov Models [2,3], Scale Invariant Feature points (SIFT) [4], tangent/turning functions [5,6], curvature maps

[7], shock graphs [8], Fourier descriptors [9,10], and so on. They all have their benefits and drawbacks, regarding computational complexity, precision capabilities, implementation issues, robustness and scalability. See [11] for one of many comparisons performed between some of these methods.

The works in [2,3] curvature features of contour points are extracted and used to build Hidden Markov Models, and some weighted likelihood discriminator function is used to minimize classification errors between the different models, and good results (64-100% recognition rates) are presented achieved in the case of plane shape classification. In [7] curvature maps are used to compare 3D contours/shapes. In [9,10] Fourier descriptors are used, as probably the most traditional way of representing contour curves, for comparison purposes. In [10] Support Vector Machine based classification and self-organizing maps are both used for contour classification, which results in a robust, yet highly complex and computationally expensive method, resulting in recognition (precision) rates above 60%, and above 80% in most cases. Turning/tangent function based contour description and comparison [5,6] are also used, mostly for comparison purposes, for it being lightweight, fairly easy to implement, yet research seems to concentrate to continuously depart from it. These methods work by representing the contours as a function of the local directional angle of the contour points along the whole object, and comparing two such representations with different methods, most of them being also rotation invariant.

For the purposes of this paper we have chosen to use the turning function based contour representation. The main reason has been the goal of creating a very lightweight implementation of contour matching and classification, which is also robust, and easy to implement and modularize.

2 The Shape Retrieval

The dataset we use in this paper are those from [2] with minor additions. The set contains 590 individual shapes, grouped into 23 classes, each class containing versions (rotated, modified but similar, noisier, etc.) of the same shape. Some examples for the different classes are shown on Fig. 1. The data set is a collection of Matlab mat files, which we transformed into different size binary images, each containing the shape in the middle of the image, having a 5 pixel border around the shape. The reason why the method uses images instead of the available point lists is that the implemented method is intended to be a general contour descriptor and indexer module as a part of a content based retrieval engine (the one also used in [1]), thus being able to work with any type of input as long as it is a binary image containing a freeform, but closed contour, object shape.

Using the series of images generated, the steps of the retrieval the following:

1. Load image, extract contour points, drop reoccurring coordinates.
2. Build the index tree: the original point list is the stored data, but a denoised version is generated for the turning function based comparison, which is the heart of the indexing.
3. Classification/retrieval step: a query shape is the input, the results are the most similar shapes from the indexed data.

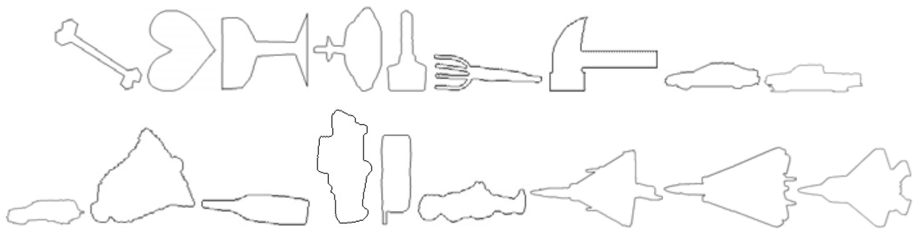

Fig. 1. Examples of shape classes

2.1 Contour Extraction

For contour representation, a list of coordinates is extracted from the input shape images. The point list is extracted by a bug follower contour tracking method with backtracking, able to start from any random object point and follow a contour of the object shape, having any free form.

The resulting point list contains hundreds to thousands of neighboring coordinate pairs, and sometimes points can be repeated in the case of certain small bulges present on the contour. Thus, a post-processing step eliminates coordinate re-occurrences. The resulting list will be finally stored.

During the indexing and retrieval process, a turning function based distance metric will be used - described below in section 2.2 - for which we also generate a denoised contour version, thus improving the indexing and retrieval times, and the also the recognition rate (detailed below in sections 2.3-5).

The smoothing procedure is also a lightweight approach. Let

$$C = C_i = (x_i, y_i)|i = \overline{0, n} \tag{1}$$

be the contour with C_i as the contour points, n the number of point coordinates, starting from a random contour position (x_0, y_0), and ending with a neighbor of the start position. Then, the smoothed contour will be

$$C' = C'_i = (x'_i, y'_i)|i = \overline{0, m}, m < n \tag{2}$$

$$(x_i, y_i) \in C' \text{ if } d((x_{i-k}, y_{i-k}), (x_{i+l}, y_{i+l})) < \varepsilon \tag{3}$$

where ε is a threshold (generally between 2.8 and 4.24 and $k >= 2$, $l >= 2$. This is basically a very primitive version of an outlier detector, working well enough for most practical purposes (see e.g. Fig. 2), yet remaining lightweight and very fast. The resulting contours are only used in the comparison distance metric in the indexing and retrieval phase, the points of the original contour are not changed. This simple smoothing does not disturb the features of the contour, but results in considerable improvement in recognition rates and indexing/retrieval times.

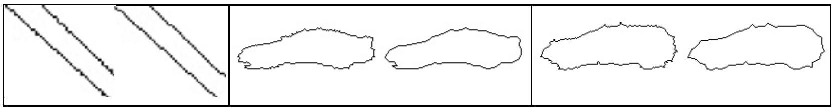

Fig. 2. For each pair: original contour (left). smoothed (right).

2.2 Indexing

The indexing step of the presented method takes as input the contours extracted in the previous section, and produces a serialization of an index tree structure, which will be used in the retrieval step.

The trees we use are customized BK-trees [12], which we will call BK*-trees. Traditionally BK-trees have been used for string matching algorithms. Essentially they are representations of point distributions in discrete metric spaces. That is, if we have feature points with an associated distance metric, then we can populate a BK*-tree with these points in the following way:

1. Pick one of the points as the root node, R.
2. Each node will have a constant number of M child nodes.
3. A point P_j will be placed into the child node N_i $(i = 0...M-1)$, if

$$i \cdot \frac{d}{M} < d(P_i, P_j) < (i+1) \cdot \frac{d}{M} \tag{4}$$

   where d is the maximum distance that two points can have (respective the associated metric) and P_j is P_i's parent node. Thus, a node will contain a point if its distance from the parent falls into the interval specified above; each node representing a difference interval $[i \cdot d/M; (i+1) \cdot d/M]$.
4. Continue recursively until there are no more points left to insert.

As it is, this structure can be used to build quickly searchable index trees for any descriptor which has a metric. We also use this indexing structure for content-based video retrieval (in the system also being part of [1]), as the base structure of indexed content descriptors, where multidimensional queries can be performed using the BK*-tree indexes of different feature descriptors.

The performance of the indexer: it takes 10.8 seconds to index the 590 shape database we use for the purposes of this paper, on a 2.4GHz Core2 CPU core.

We compared the indexing of the raw contour point sets (in which only the reoccurring coordinate points have been eliminated) with the indexing where the smoothed contour versions were used as the input of the comparison function. Fig. 3 shows the difference, where we obtained a decrease of 15 times in the running time of the indexer, if the smoothing step was used. The effects of the smoothing on the recognition rates are detailed in section 2.3 and 2.4.

The distance metric of the indexing - also used in querying the index structure in the retrieval phase - is a comparison of the results of turning function over the shapes. The output of the turning function is a 2D function representing the directions of the shape points over its contour positions.

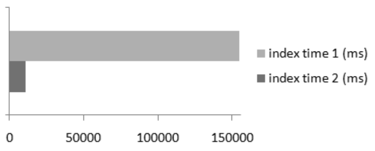

Fig. 3. Using the smoothed contour representations as the base of comparison increases the indexer's speed by as much as 15 times on average

The turning function $\theta(s)$ is a standard way of representing a polygon. It measures the angle of tangent as a function of arc length s. The most important feature of this representation is that it is invariant for translation and scaling, but it is unstable when noise is present - this is why we implemented the smoothing step presented above. We measure the distance between two turning function representations with the following formula:

$$D(P_1, P_2) = \min_t \left[\int |\theta_1(s+t) - \theta_2(s) + \theta| ds \right] \quad (5)$$

where θ is a translation parameter, which makes the distance metric rotation invariant, by comparing the shifted versions of the first function to the second function. Thus the distance becomes translation, scaling and rotation invariant.

2.3 Retrieval and Classification

Given the index tree generated with the above described method, the retrieval of shapes can be performed. The query of the retrieval is a shape image similar in contents of the ones indexed, that is a black and white image containing the contour of the query object.

Given a content-based query (Q), the index tree is searched for similar entries:

1. If $d_0 = d(Q, R) < t$ (t is user-adjustable), the root R element is a result.
2. Let N_i be the children of node P_j ($P_0 = R$), and let $d_k = d(N_k, P_j)$ for the child N_k where

$$\begin{cases} k \cdot \frac{d}{M} \in [d_{j-1} - t, d_{j-1} + t] & \text{or} \\ (k+1) \cdot \frac{d}{M} \in [d_{j-1} - t, d_{j-1} + t] & \text{or} \\ k \cdot \frac{d}{M} \leq d_{j-1} - t \text{ and } (k+1) \cdot \frac{d}{M} \geq d_{j-1} + t \end{cases} \quad (6)$$

   then if $d_k < t$ the element from child N_k is a result.
3. Repeat step 2 recursively until the whole tree is visited.
4. Sort all the results in the increasing order of their d distances and return the ordered result list.

The performance of the retrieval is detailed in sections 2.4-5. Fig. 3 showed that the indexing speed is greatly increased by the smoothing, but that in itself brings nothing, unless the retrieval performance is conserved. We will show that not only does the smoothing conserve the recognition rate, but it also improves the retrieval speed more than 20 times over. Fig. 4 shows retrieval times without and

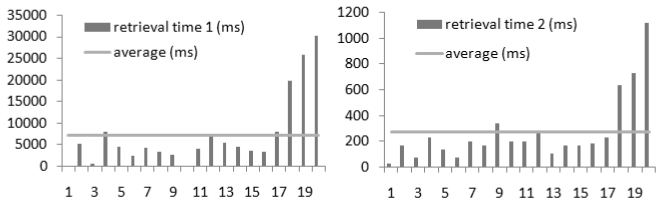

Fig. 4. Comparing retrieval times without (left) and with (right) the smoothing step in the distance metric

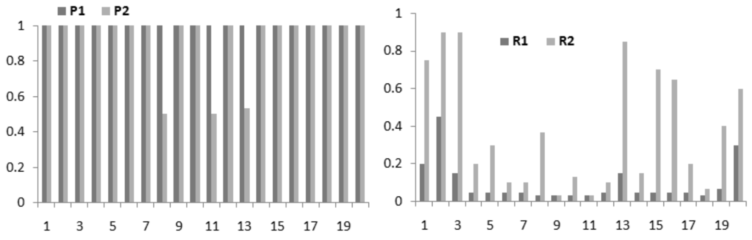

Fig. 5. Comparing recognition rates without (P1, R1) and with (P2, R2) the smoothing step in the distance metric. Left: precision values. Right: recall values.

with smoothing, containing times of 20 different queries (each query was picked from a different class), and their averages. The average run time for the raw retrieval over the 590 dataset is 7230 ms, which - by including the smoothing step - drops down to 275 ms.

We also compared retrieval performance changes caused by the inclusion of the smoothing step. Fig. 5 shows precision and recall values for the two cases. As the experiments show, comparison with smoothing resulted in almost similar precision values, while it caused considerable improvement in recall rates. Precision and recall are defined by

$$P = \frac{\text{nr. of relevant retrieved items}}{\text{nr. retrieved items}}; \quad R = \frac{\text{nr. of relevant retrieved items}}{\text{nr. all relevant items}} \tag{7}$$

2.4 Evaluation on Synthetic Data

In this section we present some retrieval evaluation details, concerning recognition rate evaluation, and presenting precision-recall values for a series of retrieval experiments. The retrievals were performed on the above presented datapool of 590 shapes belonging to 23 classes. We used 20 different query shapes, each belonging to a different class, and for each query we ran 6 different retrievals with differing retrieval thresholds (influencing the number of retrieved results), leading to a total of 120 retrieval runs.

The shape retrieval literature - e.g. [2,9] - contains extensive recognition rate evaluation data, generally varying between 60-100% precision rate. Our

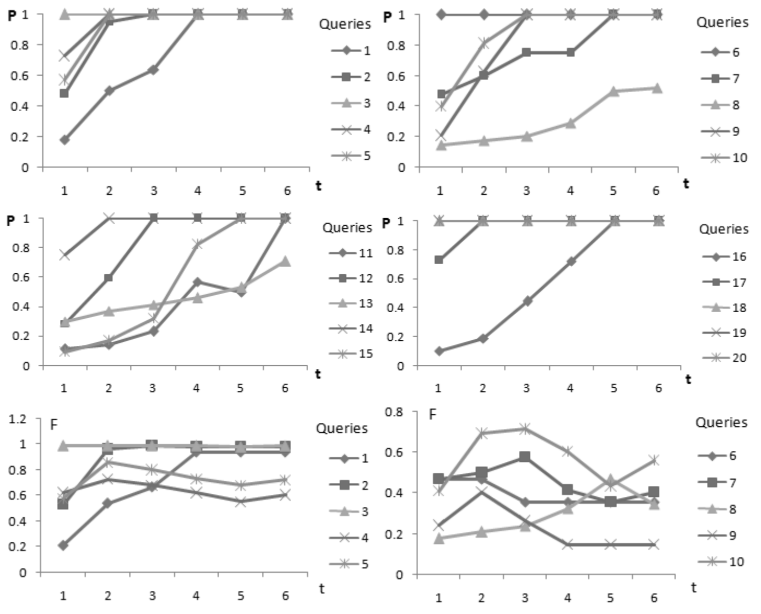

Fig. 6. Precision values of retrievals for 20 query shapes in 6 different retrieval threshold cases (top and middle row) and F0.5 score values for the first 10 queries (bottom row) showing combined P-R values

experiments showed that the retrieval method presented in this paper can produce 100% recognition rates in 90% of retrievals (to be precise: the method can produce these results with the presently used shape database).

Fig. 6 presents recognition rate performance data (precision and F score values) over the 20 queries with 6 retrieval thresholds (separated into 4 graphs for easier viewing). The graphs show that with the appropriate retrieval thresholds we can achieve 100% precision in 90% of the cases. Table 3a presents actual numerical values for the recognition rates for 20 queries, while Table 3b shows average recognition rate values from other works for a quick comparison.

Fig. 7 presents results for 7 sample queries (from the 20 query shapes) and the first 7 retrieved results. As this example also shows, even when recall values become low at certain retrievals, the recognition always remains stable, and classification of a shape into the different shape classes remains high in all cases.

2.5 Evaluation on Real Data

To evaluate the performance of the method in real circumstances, we performed recognition tests on a dataset of real plane shapes (over 8000 shapes in 24 classes), automatically segmented from from real video footage - no synthetic data. Each plane type is treated as a separate class, and during recognition a retrieval only counts as positive when the correct plane type is recognized. Given the index tree generated with the described method, the retrieval of shapes can

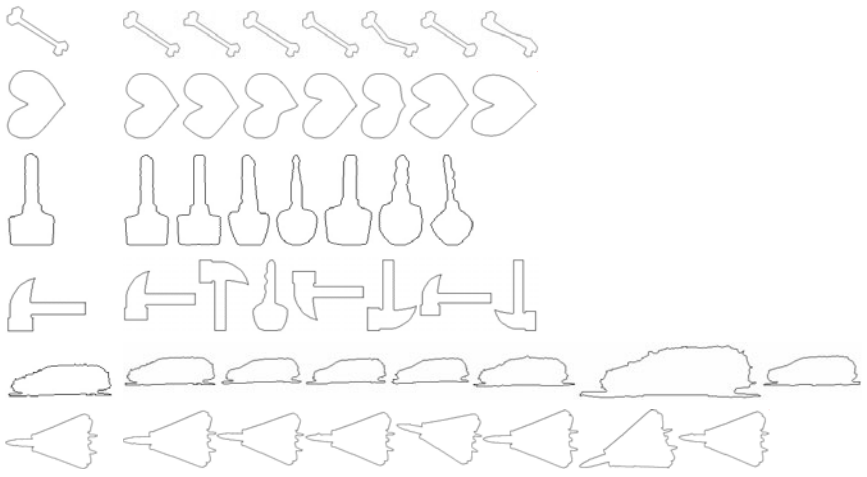

Fig. 7. Example retrieval results for 6 queries (left column) and first 7 retrieved results (on the right)

Table 1. The used shape classes, the number of shape variations of the classes and a sample mask from them

class id.	1	2	3	4	5	6	7	8	9	10	11	12
shapes in class	221	207	665	836	544	597	300	572	87	211	79	184
sample												
class id.	13	14	15	16	17	18	19	20	21	22	23	24
shapes in class	104	224	445	124	11	340	638	392	90	74	285	501
sample												

Table 2. Recognition rate data for 120 queries, belonging to 7 different classes

test nr.	1	2	3	4	5	6	7	avg.
nr. of queries	12	7	10	10	40	25	16	120
recognition rate	1	0.86	0.89	1	0.93	0.65	0.59	0.85

be performed. The query of the retrieval is a shape image similar in content to the ones already indexed. Table 1 shows how many different shape variations each class had in itself.

Testing of the retrieval performance is done as follows: a video of a plane type that has a class in the dataset is taken as a test video, and for each frame of the video a query is performed against the indexed shape database. If plane type A was always recognized as belonging to class B, then according to the retrievals source A is 100% of class B. Table 2 contains some examples for average recognition rates of the retrieval (1 means 100%). Generally, the performance will

Table 3. (a) Recognition data for the 20 query shapes, belonging to different shape classes: set1 - Bicego et al. [3], set2 - cars [2], set3 - MPEG-7 shapes [2], set4 - plane shapes [2]. (b) Average recognition rates from other works.

(a)

query nr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
sets	set1							set2				set3						set4		
best recog.	1	1	1	1	1	1	1	0.5	1	1	1	1	0.7	1	1	1	1	1	1	1
avg. recog.	0.7	0.9	1	0.95	0.9	1	0.8	0.3	0.8	0.9	0.4	0.8	0.5	0.96	0.6	0.6	0.95	1	1	1

(b)

[3] - avg. precisions for:		[10] - avg. precisions for:		[9] - avg. precisions for diff. features:	
different noise levels:	0.92	99 shapes	0.9	MI	0.69
30 degree tilt for 9 slant levels:	0.8	216 shapes	0.97	FD	0.57
60 degree tilt for 9 slant levels:	0.75	1045 shapes	0.84	UNL	0.71
90 degree tilt for 9 slant levels:	0.69	24 shape classes	0.9	UNL-F	0.98

usually get higher if the analysis is performed on a longer feed/image sequence (thus more queries are performed) since the inter-class variance of different shape types is very high, and also, a high number of plane shapes are very similar from a profile perspective (which can lower recognition rates).

3 Conclusions

In this paper we have presented an implementation of a shape indexing and retrieval method, with the main goals of being lightweight and fast, and with high precision. The final approach serves as one of the low level descriptors in content-based retrieval, which higher level semantic interpretations build upon. Future work will concentrate on applications for realtime recognition tasks, e.g. object recognition and tracking, identification tasks for unmanned aerial vehicles.

Acknowledgments. This work has been partially supported by the MEDUSA project of the EDA.

References

1. Szlávik, Z., Kovács, L., Havasi, L., Benedek, C., Petrás, I., Utasi, A., Licsár, A., Czúni, L., Szirányi, T.: Behavior and event detection for annotation and surveillance. In: International Workshop on Content-Based Multimedia Indexing, pp. 117–124 (2008)
2. Thakoor, N., Gao, J., Jung, S.: Hidden markov model-based weighted likelihood discriminant for 2d shape classification. IEEE Tr. on Image Processing 16, 2707–2719 (2007)

3. Bicego, M., Murino, V.: Investigating hidden markov models' capabilities in 2d shape classification. IEEE Tr. on Pattern Recognition and Machine Intelligence 26, 281–286 (2004)
4. Lowe, D.G.: Object recognition from local scale-invariant features. In: ICCV, pp. 1150–1157 (1999)
5. Scassellati, B., Alexopoulos, S., Flickner, M.: Retrieving images by 2d shape: a comparison of computation methods with perceptual judgments. In: SPIE Storage and Retrieval for Image and Video Databases II, vol. 2185, pp. 2–14 (1994)
6. Latecki, L.J., Lakamper, R.: Application of planar shape comparison to object retrieval in image databases. Pattern Recognition 35, 15–29 (2002)
7. Gatzke, T., Garland, M.: Curvature maps for local shape comparison. In: Shape Modeling and Applications, pp. 244–253 (2005)
8. Sebastian, T., Klein, P.N., Kimia, B.B.: Recognition of shapes by editing their shock graphs, vol. 26, pp. 550–571 (2004)
9. Frejlichowski, D.: An algorithm for binary contour objects representation and recognition. In: Campilho, A., Kamel, M.S. (eds.) ICIAR 2008. LNCS, vol. 5112, pp. 537–546. Springer, Heidelberg (2008)
10. Wong, W.T., Shih, F.Y., Liu, J.: Shape-based image retrieval using support vector machines, fourier descriptors and self-organizing maps. Intl. Journal of Information Sciences 177, 1878–1891 (2007)
11. Rosenhahn, B., Brox, T., Cremers, D., Seidel, H.: A comparison of shape matching methods for contour based pose estimation. In: Reulke, R., Eckardt, U., Flach, B., Knauer, U., Polthier, K. (eds.) IWCIA 2006. LNCS, vol. 4040, pp. 263–276. Springer, Heidelberg (2006)
12. Burkhard, W., Keller, R.: Some approaches to best-match file searching. Communications of the ACM 16, 230–236 (1973)