Shape retrieval and recognition on mobile devices

Levente Kovács

Distributed Events Analysis Research Group
Computer and Automation Research Institute, Hungarian Academy of Sciences
Budapest, Hungary
`levente.kovacs@sztaki.hu`
`http://web.eee.sztaki.hu/~kla`

Abstract. This paper presents a proof-of-concept shape/contour-based visual recognition and retrieval approach with the main goal of lightweight implementation on mobile devices locally, not relying on network connection or server side processing. In such circumstances the focus needs to be on effectiveness and simplicity, while still preserving high level of functionality (i.e. good recognition). Application areas involve offline object recognition and template matching (e.g. for authorization and blind aid applications), and various object categorizations either in pre-processing or in full local processing.

Keywords: mobile vision, shape recognition, indexing, retrieval

1 Introduction

As mobile devices proliferate, and include higher processing capabilities, previously workstation-based solutions spread to these devices, either in fully local processing mode, or combined with partial or full server side processing. Targeted areas range from signal processing to high level visual processing tasks. While server side processing can have its benefits - availability of large datapools, of practically unlimited processing power and storage space, etc. -, local processing can also have positive points: low latency, quick responses, non-dependence on network availability or speed, lower costs associated with data transfers and bandwidth. Also, local processing might serve as a pre-processing step of the server-side computations, reducing communication bandwidth and associated latency and cost.

This work presents the first step towards creating a targeted visual recognition solution with local processing on mobile devices, with network and cost non-dependency. The focus is on creating efficient and fast implementations of algorithms, based on the constrained environment (processing power, memory, battery) of mobile devices. The targeted scenario is when we have some a priori information about the possible recognition tasks (e.g. signs, labels, object types) then we can use offline built indexes, upload them to the device, and perform all recognition steps on the device by using these indexes. As a proof-of-concept, in this work we present a shape-based visual recognition approach which is run locally on a mobile device, and is able to perform queries against a pre-built compact index containing information about an a-priori shape dataset, with the goal of quickly providing a result for that query, which is the type (class, category, label) of the query object. The presented approach uses a synthetic a-priori shape dataset with versatile content, to provide an example of the possibilities of such a solution.

Traditionally, contours/shape descriptors have been extracted and compared with a series of methods, including Hidden Markov Models [19, 2], Scale Invariant Feature points (SIFT) [14], tangent/turning functions [16, 13], curvature maps [5], shock graphs [18], Fourier descriptors [4, 20], and so on. They all have their benefits and drawbacks, regarding computational complexity, precision capabilities, implementation issues, robustness and scalability. See [15] for one of many comparisons performed between some of these methods.

The works in [19, 2] 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 [5] curvature maps are used to compare 3D contours/shapes. In [4, 20] Fourier descriptors are used, as probably the most traditional way of representing contour curves, for comparison purposes. In [20] 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 [16, 13] are also used, mostly for comparison purposes, for it being lightweight and fairly easy to implement. 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.

In this paper we used a modified turning function based contour comparison metric (also used in [12] and presented in detail in [11]). Here, since constraints in mobile computational power, storage space and memory, we use a modified version, presented later.

The novel and interesting elements of the presented approach are the following: providing a proof-of-concept of near-realtime content-based search locally and offline on a mobile device, with local storage, concentrating on speed and effectiveness for low latency, capable of using a number of different a-priori indexes thus providing the possibility of multi-class and multi-feature recognition.

2 Concept, architecture

The idea of local processing on mobile devices is not new. Yet, until recent years, mobile devices lacked the processing capacity to perform any decent computation offline. Thus most of the higher requirement applications have been built around the Web application and/or cloud based services paradigm. This direction has proved its viability, thousands of applications and services use such architecture to provide server-side processing based consumer interaction. Yet, this paradigm has one flaw: dependence on constant, reliable, cheap (also considering data roaming), high bandwidth network connection. One or two of these parameters might be always reachable, but it is very seldom that all of them are.

Thus, as local computation capabilities of mobile devices rise, offline, or combined offline + online service can be a viable alternative, especially when processing would require a high volume of data propagation. E.g. pre-processing images on the device and not sending it to a server (only processed data) can result in real latency and bandwidth reduction (Fig. 1a).

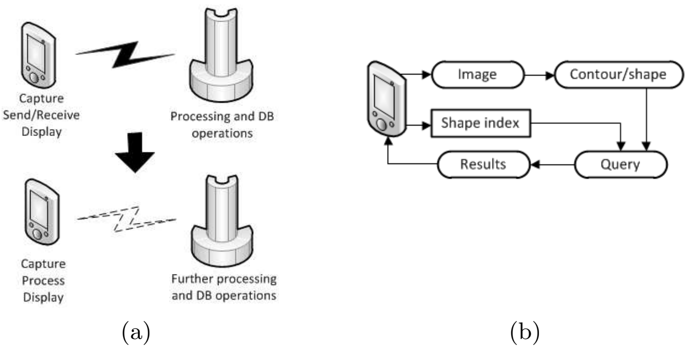


Fig. 1: (a) Top: online: capture data on the device, send, wait for results, then receive and display. Bottom: offline: capture, process and display on the device, and only send upstream if necessary. (b) Preferably all processing is performed on the device, without bandwidth, latency and cost issues.

In our case, we are dealing with visual processing, which has always been a computationally intensive and high data volume field. It is for this reason that most approaches and services work with server side processing [7, 1], and only a few small apps provide local processing [9]. Recently there has been a new interest in combined local and server side processing apps [17].

The motivation behind this work was to find a solution for the above problems, in the field of visual recognition tasks. In the case of visual recognition, large datasets, a-priori training, high computation complexity are hard to overcome, but there might be situations and applications when local offline processing can be used. One of these areas is on-device shape based recognition of objects, and extraction of shape features as a pre-processing step for server side tasks.

The shape-based processing consists of the following steps (Fig. 1b): loading a prepared shape index file onto the device, capturing or selecting a query shape, performing the query and presenting the result. For the implementation we chose the Android platform[6], for the easy access and availability, and the capability of combining Java and native C++ code in the same application. Development was done using an Android v2.2 emulator, and real tests were performed on Samsung Galaxy 3 (i5800) phone (CPU 667MHz single core, 256MB RAM).

Fig. 2 contains example screenshots of the proof-of-concept application. Query images can be picked from a folder on the device or shot with the camera, then the index file is loaded and the search can be performed. The result (the closest match) is displayed in the textual status line at the top of the window.

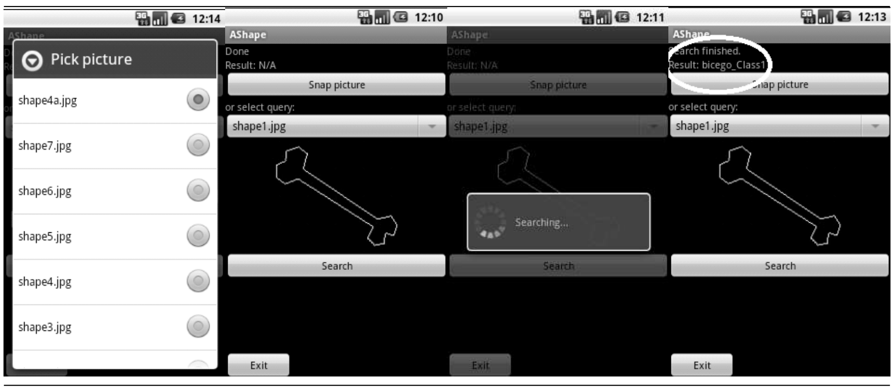


Fig. 2: Screencaps of the proof-of-concept app from the Android device emulator. From left to right: picking a stored image for query; main interface (showing picked or captured query); performing the search; displaying the result.

2.1 Implementation issues

During the implementation of the presented algorithmic steps, effectiveness and platform limitations are always an issue to be considered. First, processing capabilities of mobile devices are rising, but there are still quite a number of lower capacity models. Second, on most platforms there are limitations to be considered, e.g. maximum heap/stack sizes, memory capacity, storage space limitations, etc. Some of the computational issues (e.g. Java applications' memory limitations) can be somewhat alleviated in the case of Android, since there is a native part of the SDK available, thus Java code can be combined with C++ in the same application through Java/JNI interfaces. In the case of the proof-of-concept test application, we used the same approach (Fig. 3): we put as much of the algorithms into native code as possible, with the goal of higher speed and effectiveness.

Also, for memory and storage considerations, the dataset used for the building of the index is not transferred to the mobile device, only the index is uploaded, which only contains the shape information of the nodes and an identifier to be able to provide answers to queries.

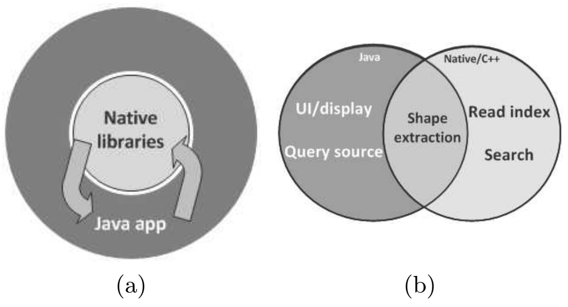


Fig. 3: (a) For speed considerations and memory requirements, most of the algorithms run transparently in native C++ on the Android device, used as external libraries from the Java application. (b) High level grouping of algorithmic elements based on their implementation.

3 Indexing, dataset

The basis of the test dataset used is the one used in [2], extended with other synthetic shape classes[19]. The original number of 673 shapes in 30 classes was reduced to 534 shapes after filtering out very similar shapes during the index building process. Fig. 4 shows some examples of shapes from each of the classes. In-class variations include different distortions of the same shape (Fig. 5). Table 1 shows the number of shapes in each class after the indexing is performed.

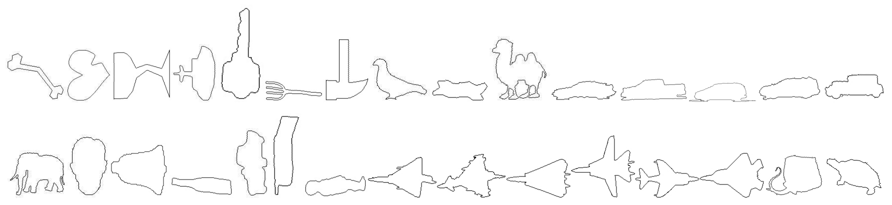


Fig. 4: One example from each of the used synthetic shape classes. Each class contains various versions of each shape.

To facilitate effective searching in the retrieval/recognition phase, an a-priori index structure needs to be built which can be loaded on the device, to be available when the application starts. The index should be able to handle shape features. Obviously, a lot of different tree structures are available for representing relations between all kinds of objects, including classic ones like M, B, R, TV, etc. and complex ones for multidimensional data like BVH trees, KD-trees, and so on. In choosing an index structure, especially in the case of high performance in

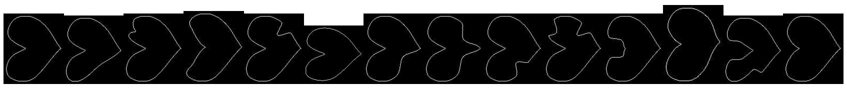


Fig. 5: Example for in-class shape variations.

a low computation capacity environments, easy handling, low memory footprint and fast searching are the main desired parameters, while time to build the index are only marginally important. Our previous works where we needed realtime search capabilities[12] led us to choose a modified BK-tree[3] solution. One of the main benefits of the BK-tree-based approach is that after a proper index building, during the search large parts of the index tree can be dropped even from the first level, and there is no need to revisit other branches on the same level again. This makes searching through the tree a fast process.

The index structure is a variation of BK-trees introduced earlier[12, 11] (Fig. 6). Essentially the index trees are representations of point distributions in discrete metric spaces. For classical string matching purposes, the tree is built so as to have each subtree contain sets of strings that are at the same distance from the subtree's root, i.e. for all e leaves below sub-root r the $d(e,r) = \varepsilon$ is constant. In our case, the modifications include a). using a tree structure where the nodes contain not only an identifier, but also the descriptor data associated to the node, which makes searching and result display a fast process, and b). using a structure that contains nodes that can have an arbitrary number of children (N), where the leaves below each child contain elements for which the distance d falls in a difference interval: $d(e,r) \in [\varepsilon_i; \varepsilon_{i+1})$ (where $i \in [0,N] \cap \mathbb{N}$). The distance intervals in the child nodes (denoted by $\varepsilon_i, \varepsilon_{i+1}$ above) depend on the maximum error E_{max} that the distance metric can have, more specifically, $\|\varepsilon_{i+1} - \varepsilon_i\| = E_{max}/N$, thus the intervals are linearly divided buckets.

The result of the indexing process is a binary file that contains a compact representation of the built index tree structure. When searching the index, this binary file is loaded, from which the tree structure can be replicated.

Table 1: Number of elements in each of the used shape classes.

class number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
nr. of elements	20	20	20	40	20	20	20	12	11	11	30	30	30	30	12
class number	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
nr. of elements	12	8	20	20	20	20	20	17	2	22	7	3	20	7	10

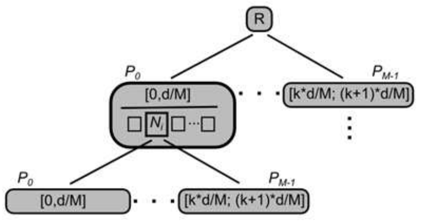


Fig. 6: Structure of an index tree.

4 Retrieval, recognition

For the dataset contour point data is already available, thus they are used (after a noise filtering step to produce smoother contours) during the index building process.

To localize objects in the query image, we use a contour point detection method [10] based on the Harris corner detection [8] as a starting point. The traditional approach only emphasizes corners in the image, but for contour point detection we need both corner and edge points to be extracted, therefore we need to modify the original characteristic function.

The eigenvalues of the so called Harris matrix[8] M (denoted by λ_1 and λ_2) will be proportional to the principal curvature of the local autocorrelation function and separate three kinds of regions: both of them are large in corner regions, only one of them is large in edge regions and both of them are small in homogeneous (flat) regions. In the modified detector points are then detected as the local maxima of $L = \max(\lambda_1, \lambda_2)$ around pixel (x_i, y_i):

$$(x_i^\star, y_i^\star) = \underset{(x_i,y_i)\in b}{\operatorname{argmax}} \{L(x_i, y_i)\} \ , \tag{1}$$

and the set of detected $(x_i^\star, y_i^\star)$ points will be used as corner/contour points. Fig. 7 shows examples to compare the Harris detector with the modified detector, while Fig. 8 shows an example contour detection.

As a metric for comparing different extracted contours/shapes, we used a turning function approach[12, 11]. The reason for choosing such an approach is the lightweight nature of the method, and its tolerance against rotation, scale and noise/changes in the shapes. Fig. 9 shows an example for internal representation of a shape.

In recognizing shapes, the result of a query is the best match. Yet, to evaluate the metric and the comparison capabilities of the approach, we included the results of retrieval tests, and the first 5 results for 7 different queries are displayed as examples in Fig. 12. The results show the viability of the approach.

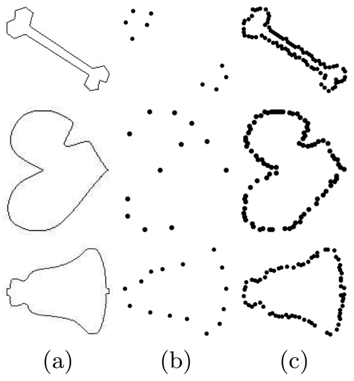


Fig. 7: Contour point detection. (a): samples; (b): original Harris corner detector; (c): used detector[10].

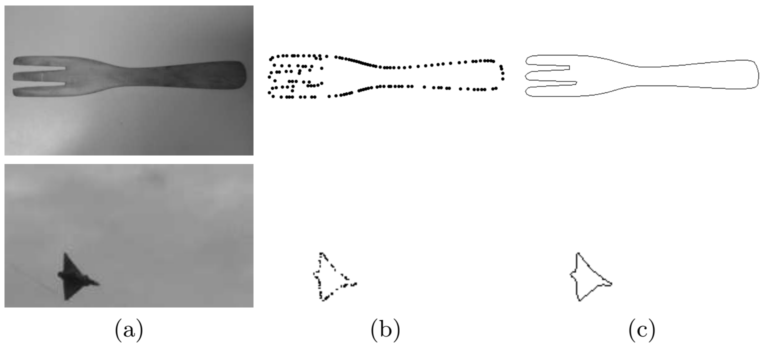


Fig. 8: Sample input: (a) image; (b) detected points; (c) produced contour.

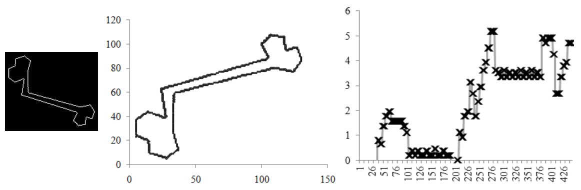


Fig. 9: Sample for input image (left), extracted contour (middle), internal turning representation (right).

4.1 Evaluation

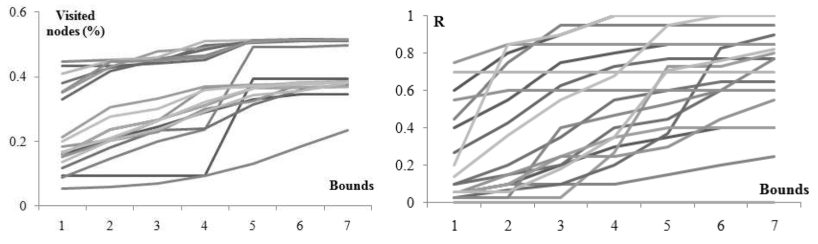
We investigated the visual and numerical performance of the above presented approach on the outlines dataset. First, recognition/retrieval capabilities were investigated for numerical results, which are presented in Fig. 10.

For all queries and retrieval tests, the first result was always inline with the query, which means we received a recognition rate of 100% (of course in the case when queries were belonging to one of the indexed classes).

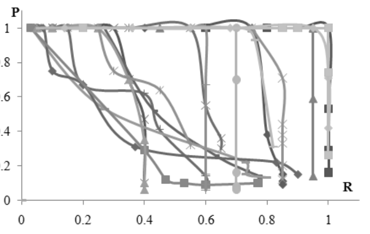
We investigated the relation between retrieval threshold bounds (affecting the number of returned results) and the number (percentage) of visited nodes in the index tree (Fig. 10a), which directly affects time performance. The expected result was that allowing a higher number of results causes large increase in the number of visited nodes (and required time). We also investigated how the change in the number of visited nodes affects recall rates (Fig. 10b). Here, the expected outcome - which was also the produced result - was that an increase in visited nodes results in an increase of recall values (note: recall is the number of relevant results with relation to the total number of possible relevant dataset elements).

Fig. 10c shows precision-recall (P/R) curves related to the above tests (20 queries with 7 retrieval bounds) with the intent of investigating the approach from a content retrieval point of view. Overall the results turned out promising: the first returns (at low retrieval bounds) which are used for recognition are always $P = 1$, and recall rates are also good, ranging from 40 to 100%. Additionally, we included in Fig. 12 some visual retrieval results to extend Fig. 10c.

We also ran measurements to gather performance data regarding the time required to produce recognition results (Fig. 11). We compared run times of the same algorithm running natively (C++) on an Intel 1.3GHz Core2 laptop processor, in an Android emulator running on the same processor (using Java and C++ combination), and the same code running natively on a Samsung Galaxy 3 Android phone (667MHz ARM architecture processor). Fig. 11a shows running times for all the above queries on a laptop. The default bound for real life tests is bound nr. 2, which means that generally results are provided below 500ms. Fig. 11b shows average running times for all the queries on all 3 environments. As it is shown, there is an order of magnitude difference between the PC and the 667MHz mobile device results. What this means in practice is, that for a lower number of objects (below 500), and a lower number of categories (practically below 20) close-to-realtime results can be achieved. Application areas in such scenarios include face recognition, iris recognition, fingerprint recognition (for identification and authorization purposes), or blind aid applications (currency/coin recognition, street sign recognition, etc.), all working reliably without constant network connection. Also, today we already have mobile devices (phones and tablets) with processors up to 1GHz in clock speed (some with multiple cores), which can cause significant increase in processing speed. Finally, we can conclude that local offline processing on mobile devices is a practically viable approach and good results can be expected.

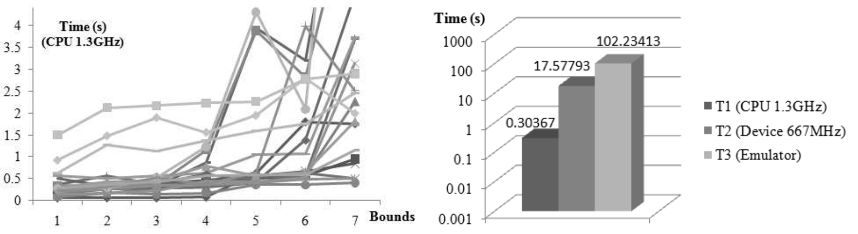


(a) (b)



(c)

Fig. 10: Evaluation of retrieval (performance). (a): Visited nodes of the index tree (in %) for 20 different query curves based on 7 decreasing retrieval bounds (horizontal axis). (b): Recall rates associated to the same queries and retrieval bounds as in (a). Meaning: as more nodes are visited in the tree, Recall rates increase accordingly. (c): Precision/Recall curves for 20 different queries, each for 7 different retrieval bounds.



(a) (b)

Fig. 11: Evaluation of retrieval (time). (a) Retrieval time curves for the same 20 queries and 7 increasing retrieval bounds (default is bound nr. 2) related to Fig. 10 - when run on a notebook Core2 CPU 1.3GHz. (b) Average time comparisons for the same set of runs on a 1.3GHz Core2 CPU, a Samsung Galaxy 3 (667MHz) and an Android Emulator (running on the same 1.3GHz CPU).

5 Conclusions

In this paper we presented an approach for a simple and fast shape-based retrieval and recognition solution for local offline processing on mobile devices. The purpose of such methods is to provide a basis for reliable, non-network-dependent solutions for visual recognition tasks. These in turn can be the basis of local authorization/login/authentication solutions, blind aid applications (sign recognition, cash classification, etc.). Currently we are in the process of providing a larger set of content based descriptors that can be used realtime on mobile devices, with the purpose of local feature extraction and categorization of a small set of objects for targeted applications (product and sign recognition).

Acknowledgements This work has been partially supported by the Hungarian Scientific Research Fund (OTKA) grant nr. PD83438.

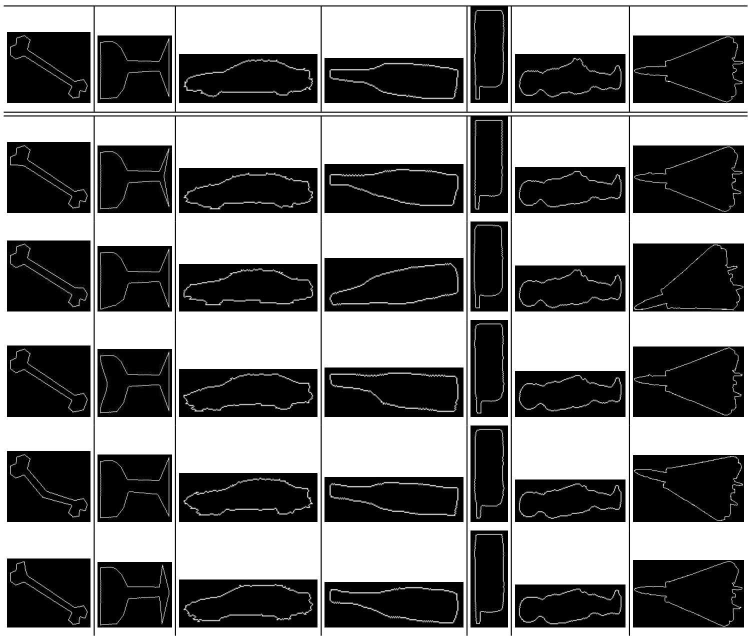


Fig. 12: Samples for retrievals for different query images (top row) and the first 5 results (in columns).

References

1. Aurasma: Aurasma Augmented Reality Platform. `http://www.aurasma.com`

2. Bicego, M., Murino, V.: Investigating Hidden Markov Models' capabilities in 2D shape classification. IEEE Tr. on Pattern Recognition and Machine Intelligence 26(2), 281–286 (2004)
3. Burkhard, W., Keller, R.: Some approaches to best-match file searching. Communications of the ACM 16, 230–236 (1973)
4. Frejlichowski, D.: An algorithm for binary contour objects representation and recognition. LNCS Image Analysis and Recognition 5112, 537–546 (2008)
5. Gatzke, T., Garland, M.: Curvature maps for local shape comparison. In: Proc. of Shape Modeling and Applications. pp. 244–253 (2005)
6. Google: Google Android SDK. `http://developer.android.com`
7. Google: Google Goggles. `http://http://www.google.com/mobile/goggles`
8. Harris, C., Stephens, M.: A combined corner and edge detector. In: Proc. of the 4th Alvey Vision Conference. pp. 147–151 (1988)
9. Ipplex: LookTel Money Reader. `http://www.looktel.com/products`
10. Kovács, A., Szirányi, T.: High definition feature map for GVF snake by using Harris function. In: Proc. of Advanced Concepts for Intelligent Vision Systems, Lecture Notes in Computer Science. vol. 6474, pp. 163–172. Springer (2010)
11. Kovács, L.: Contour based shape retrieval. In: Proc. of Intl. Symposium on Visual Computing (ISVC), Lecture Notes in Computer Science. vol. 6455, pp. 59–68. Springer (2010)
12. Kovács, L., Utasi, A.: Shape and motion fused multiple flying target recognition and tracking. In: Proc. of Automatic Target Recognition XX, SPIE Defense, Security and Sensing. vol. 7696, pp. 769605–1–12 (2010)
13. Latecki, L.J., Lakamper, R.: Application of planar shape comparison to object retrieval in image databases. Pattern Recognition 35(1), 15–29 (2002)
14. Lowe, D.G.: Object recognition from local scale-invariant features. In: ICCV. pp. 1150–1157 (1999)
15. Rosenhahn, B., Brox, T., Cremers, D., Seidel, H.: A comparison of shape matching methods for contour based pose estimation. LNCS Combinatorial Image Analysis 4040, 263–276 (2006)
16. Scassellati, B., Alexopoulos, S., Flickner, M.: Retrieving images by 2D shape: a comparison of computation methods with perceptual judgements. In: Proc. of SPIE Storage and Retrieval for Image and Video Databases II. vol. 2185, pp. 2–14 (1994)
17. Schroth, G.and Huitl, R., Chen, D., Abu-Alqumsan, M., Al-Nuaimi, A., Steinbach, E.: Mobile visual location recognition. IEEE Signal Processing Magazine 28(4), 77–89 (2011)
18. Sebastian, T., Klein, P.N., Kimia, B.B.: Recognition of shapes by editing their shock graphs 26(5), 550–571 (2004)
19. Thakoor, N., Gao, J., Jung, S.: Hidden Markov Model-based weighted likelihood discriminant for 2D shape classification. IEEE Tr. on Image Processing 16(11), 2707–2719 (2007)
20. 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(8), 1878–1891 (2007)