Embedded Real-Time Visual Search with Visual Distance Estimation

Marco Paracchini[1](✉), Emanuele Plebani[2], Mehdi Ben Iche[2], Danilo Pietro Pau[2], and Marco Marcon[1]

[1] DEIB - Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan, Italy
{marcobrando.paracchini,marco.marcon}@polimi.it
[2] Advanced System Technology, STMicroelectronics, Agrate Brianza, Italy
{emanuele.plebani1,mehdi.beniche-ext,danilo.pau}@st.com

Abstract. Visual Search algorithms are a class of methods that retrieve images by their content. In particular, given a database of reference images and a query image the goal is to find an image among the database that depicts the same object as in the query, if any. Moreover, in many different real case applications more than one object of interest could be viewed in the query image. Furthermore, in this kind of situations, often, it is not sufficient to identify the object depicted on a query image but its precise localization inside the scene viewed by the camera is also requested. In this paper we propose to couple a Visual Search system, which can retrieve multiple objects from the same query image, with an additional Distance Estimation module that exploits the localization information already computed inside the Visual Search stage to estimate localization of the object in three dimensions. In this work we implement the complete image retrieval and spatial localization pipeline (including relative distance estimation) on two different embedded devices, exploiting also their GPU in order to get near real time performances on low-power devices. Lastly, the accuracy of the proposed distance estimation is evaluated on a dataset of annotated query-reference pairs ad-hoc created.

Keywords: Visual Search · CBIR · LoG · SIFT · Distance estimation

1 Introduction

Visual Search (VS), also known as Content Base Image Recognition (CBIR), is generally defined as a system, based on computer vision methods, that performs image retrieval based on the pixel content alone. In particular, given a query image depicting a (usually rigid and not reflective) object and a reference database of images containing the query object among many others, the system has to return the correct image. This task is generally different from image classification, since in VS the goal is to identify a single object instance in the image (e.g. a specific book or a building) instead of assigning a class to the object (e.g. a generic book or a generic building). VS systems can be modeled as a large Cyber

Physical System (CPS) [1] in which many mobile imaging nodes (i.e. the users' smartphones) acquire images and contact the central node to retrieve some information about the objects depicted in each image. Such systems can be used in a large amount of domains, such as mobile e-commerce applications, interactive museum guides and many others. In general, VS applications may potentially substitute QR codes in all the fields they are used in, essentially replacing a printed code attached to the object with the object itself and thus preserving the original object appearance.

In this paper we consider scenarios in which the query object must not only be identified but also precisely localized in the physical 3D space in front of the camera. An application of this kind could be implemented in many different scenarios, such as robots picking up different objects from an unordered shelf scaffold (e.g. in a kitchen or a warehouse) or a drone that has to reach a specific landing pad. In these particular scenarios the VS application could run entirely on the local device and on a small reference database, without the need of a central node. On the other hand, the application must be also able to identify and localize multiple objects depicted in a single query image and estimate the distance between them and the observing camera. The only assumptions are that the camera is already calibrated and the objects retrieved are rigid, non reflective, locally planar and of known size.

An application able to solve the aforementioned task is described in Sect. 3 and its implementation and testing on two different ARM devices is reported in Sect. 4. Finally in Sect. 5 the real time viability of this kind of implementation on embedded boards is discussed.

2 Related Work

Algorithms for VS/CBIR firstly arose in the mid '90s with the goal of organize photo collections and analyze the image content of a rapidly growing Internet. The first applications created to solve these kind of problems were Query By Image Content (QBIC) from IBM [2] and the Photobook system from MIT [3], among many others.

The rapid growth of the smartphone market in the late 2000 s and the introduction of high resolution cameras, coupled with increased CPU/GPU computational power, moved the first personal computer (PC) and web-based Visual Search experiments towards mobile Visual Search. The growing processing power of mobile CPUs showed that sending to the server an entire image, even compressed, is unnecessary and image analysis such as feature extraction could be performed directly on the smartphone, following the "analyze then compress" paradigm to minimize the loss of information. Furthermore, other factors such as unstable or limited bandwidth and network latency empathize the importance of features compression in mobile VS applications [4].

For this reason, starting from 2010, the Moving Picture Experts Group (MPEG) standardized content based access to multimedia data in the MPEG-7 standard by defining a set of descriptors for images, namely Compact Descriptor for Visual Search (CDVS) [4]. The standard proposal also includes a reference CDVS implementation in a *Test Model* and the formal definition of a bit stream which encodes in compressed form the information required to perform a server-side search. The information encoded consist of a global descriptor, which is a digest computed from SIFT [5] descriptors extracted from the image and compressed SIFT descriptors with the coordinates of the associated keypoints.

Regarding the distance estimation part, many different methods exist in literature, as, for example, Structure from Motion (SfM) [6] or Visual Simultaneous Localization And Mapping (VSLAM) [7] in which images acquired by a set of cameras (or a singular moving camera) depicting the same scene is used to 3d reconstruct the viewed scene via triangulation and bundle adjustment. The method used in this paper is instead based on homography decomposition [8].

The Visual Search application proposed in this paper is based on the work presented in [9], without the global descriptor stage, which is unnecessary on small databases such as the ones used to store the reference targets, and with the addition of the Distance Estimation module.

3 Proposed Pipeline

The Visual Search and Distance Estimation pipeline proposed in this paper can be split into four main blocks: three belongs the Visual Search part, the Image Analyzer (IA), the Retrieval Stage (RS) and the Multiple Results Loop (MRL), plus one additional module for the Distance Estimation (DE). The flowchart of the proposed method is depicted in Fig. 1.

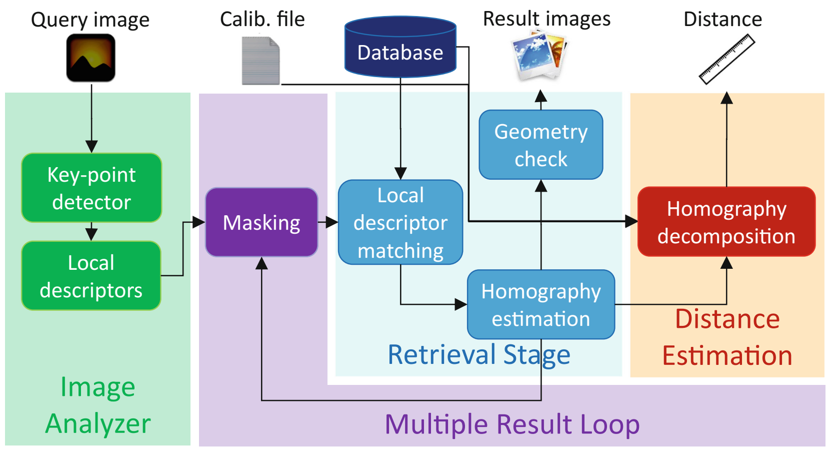


Fig. 1. The complete Visual Search and Distance Estimation pipeline proposed in this paper. The different blocks that make up the pipeline are highlighted in different colors. The query image, the reference database and the camera calibration file are the only inputs required. The outputs generated by the application are the list of objects found in the query image and their estimated distance from the observing camera. (Color figure online)

The first stage of the pipeline, IA (in green in Fig. 1), processes the image and extracts relevant visual descriptors from it whilst the second stage, RS (in blue in Fig. 1), compares the query and database image descriptors to determine if the query image shows the same object, or scene, of an image in the reference database. The MRL stage (in purple in Fig. 1) is implemented as a feedback module that is able to efficiently re-run the previous block (i.e. RS) if more than one object is depicted in the query image. The DE (the red part in Fig. 1) module is used to extract geometrical information for each of the results found by the MRL.

3.1 Image Analyzer

The Visual Search application proposed in this paper is based on local features, so the first step of the IA is to select a set of pixel coordinates $\{[u, v]^\intercal\}_k$ where relevant information about the image will be extracted. This step is implemented with a Laplacian of Gaussian (LoG) [10,11] image pyramid. The LoG filter acts as a corner detector, emphasizing regions of rapid intensity change thanks to its Laplacian component and de-noising the final result with its Gaussian component. As an alternative filter DoG (Difference of Gaussian) could be used, as in [5], which approximates the LoG response. LoG has been chosen mainly because DoG requires the computation of an additional filtered image, due to the fact that DoG is based on differences. The LoG filter is applied on a set of many down-sampled versions of the query image to detect interesting points (i.e. keypoints) on multiple scales. The set of obtained key-points is then ranked using a simple model that predict their importance and only a subset of the most relevant ones is kept.

The next step of IA is the computation of a local descriptor for each keypoint selected: the well-known SIFT [5] features have been chosen for their robustness to image changes. The keypoint detection and SIFT computation steps have been implemented in both CPU and GPU (with OpenGL); a performances analysis is reported in Sect. 4.2.

The final step of the IA stage is a scalar quantization of the SIFT features in order to reduce their size in the reference database and the query descriptor. In order to achieve the compression, for each selected keypoint each element of the relative feature vector is binarized simply comparing its value to a constant threshold given by the SIFT elements average on a large training database.

All the operations described so far are executed on-line for the query image and in a batch fashion for the compressed descriptors of the reference database images.

3.2 Retrieval Stage

The RS takes as inputs the descriptors generated by the IA and use them to determine if the query image depicts the same object as an image in the reference database. For each key-point selected and for each image in the database a local descriptor matching is performed. If I is a reference image in the database, the

k-th local descriptor of the query image matches with the j-th local feature of I if s_k^j is above a threshold, where the score s_k^j is defined as:

$$s_k^j = \cos\left(\frac{\pi}{2}\frac{h_k^j}{h_k^i}\right) \tag{1}$$

In this formula, h_k^j is the Hamming distance between the k-th query descriptor and the j-th reference descriptor, while h_k^i is the distance from the second nearest feature in the image I. This particular score function has been chosen in order to assign high values (i.e. 1) when the best match distance is much smaller in respect to the second best one and low values (i.e. 0) when the two distances are very similar. If a significant number of matched features is found, the image I is selected as a candidate match for the query image.

The described procedure generates many outlier matches between features, i.e. matches between visually similar area of the two images but geometrically inconsistent. For this reason, the matches are verified through a geometry check based on homography estimation, which assumes locally planar and rigid objects. The homography ($\boldsymbol{H}$) is rapidly computed using the widely used combination between the Direct Linear Transform (DLT) [8] algorithm and RANdom SAmple Consensus (RANSAC) [8] method in order to remove outlier matches. The list of candidate matches is then ranked using the geometry check score and the highest ranked image is chosen as the retrieval result.

3.3 Multiple Results Loop

The MRL retrieves multiple results from the database if a query picture depicts more than one object. As is shown on Fig. 1, the MRL is a feedback loop that performs cyclically the RS steps. After each RS iteration the top match is added to a multiple results list and the RS is performed again on a subset of the local features from the previous iteration. This subset is determined by mapping the bounding box of the previous result image I on the query one using the estimated homography $\boldsymbol{H}$ (adding also some padding) and computing an occupancy mask. This mask localizes accurately the region of the query image in which the retrieved object is located: then, all the features having coordinates inside the mask are removed for the next iteration of MRL. At each iteration the current mask is obtained from the union of all the previous masks and the new re-projected bounding box. This features removal operation are described in Fig. 2.

The MRL is performed until either the remaining non-masked surface in the query image is sufficiently small or the scores of the RS results drop below a selected threshold. Figure 1 also shows how the IA is not part of the MRL, so the query image is analyzed only once, independently from the number of objects depicted in it. This is particularly important if the Visual Search application is implemented on a distributed system, e.g. a Cyber-Physical System (CPS), with a central node and multiple imaging nodes, in which each leaf node can perform the IA stage independently.

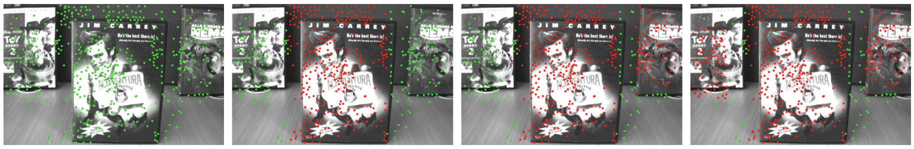

(a) 1st iteration (b) 2nd iteration (c) 3rd iteration (d) 4th iteration

Fig. 2. These figures show the MRL operation removal performed in each iteration of the loop when these method is applied on a query image depicting 3 different objects. The green circles represent the features considered in each iteration while the red ones are the removed features. (Color figure online)

3.4 Distance Estimation

The Distance Estimation module can estimate the relative position between the camera and the retrieved object, expressed as a rotation matrix $\boldsymbol{R}$ and a translation vector $\boldsymbol{t}$, by decomposing the homography $\boldsymbol{H}$ computed in the RS. The inputs needed by the module are the camera calibration parameters (for simplicity only the intrinsic parameters matrix $\boldsymbol{K}$ of a pinhole camera model) and the annotated object size, either the width w_o or the height h_o.

Having defined $\left[u\ v\ 1\right]$ as a homogeneous pixel coordinate on the query image, $\left[u^*\ v^*\ 1\right]^\mathsf{T}$ as the corresponding homogeneous pixel coordinate on the result image and $\boldsymbol{H}$ as the estimated homography, if λ is a positive scaling factor the formula:

$$\left[u\ v\ 1\right]^\mathsf{T} = \lambda \boldsymbol{H} \left[u^*\ v^*\ 1\right]^\mathsf{T} \tag{2}$$

follows from the homography definition. On the other hand, if $\left[x\ y\ 0\right]$ is a physical point lying on the observed planar object, where we assumed for simplicity a plane with equation $z = 0$, from the definition of the camera projection matrix the corresponding point on the query image $\left[u\ v\ 1\right]$ is expressed as:

$$\begin{aligned}\left[u\ v\ 1\right]^\mathsf{T} &= \gamma \boldsymbol{K} \left[\boldsymbol{R} \mid \boldsymbol{t}\right] \left[x\ y\ 0\ 1\right]^\mathsf{T} \\ &= \gamma \boldsymbol{K} \left[\boldsymbol{r_1}\ \boldsymbol{r_2}\ \boldsymbol{t}\right] \left[x\ y\ 1\right]^\mathsf{T} \\ &= \gamma \boldsymbol{K} \left[\boldsymbol{r_1}\ \boldsymbol{r_2}\ \boldsymbol{t}\right] \boldsymbol{S} \left[u^*\ v^*\ 1\right]^\mathsf{T}\end{aligned} \tag{3}$$

where γ is another positive scaling factor, $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$ are the first two columns of $\boldsymbol{R}$. $\boldsymbol{S}$ is a scaling matrix transforming 3D points on the plane to pixel coordinates $[u^*, v^*, 1]^\mathsf{T}$ defined as:

$$\boldsymbol{S} = \begin{bmatrix} \frac{w_p}{w_o} & 0 & 0 \\ 0 & \frac{h_p}{h_o} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

having defined w_o and h_o as the width and height of the physical planar object and w_p and h_p as the pixel width and height of the associated image in the database. From Eqs. 2 and 3 follows that:

$$\left[\boldsymbol{r_1}\ \boldsymbol{r_2}\ \boldsymbol{t}\right] = \eta \boldsymbol{K^{-1}} \boldsymbol{H} \boldsymbol{S^{-1}} \tag{5}$$

where η is such that $||\boldsymbol{r_1}||_2 = ||\boldsymbol{r_2}||_2 = 1$ and $\boldsymbol{r_3} = \boldsymbol{r_1} \times \boldsymbol{r_2}$; $\boldsymbol{R}$ is then orthogonalized with SVD to make the estimation more robust. The estimated distance between the object and the camera is simply $d = ||\boldsymbol{t}||_2$ where $\boldsymbol{t}$ is the estimated translation vector. The factor η combined with the scaling matrix $\boldsymbol{S}$ ensures that the translation has the right value in metric units.

4 Experiments and Results

The Visual Search and Distance Estimation pipeline described in Sect. 3 has been implemented on two different systems based on two different ARM based devices: the Hardkernel Odroid XU3[1] and the STMicroelectronics B2260[2]. The Odroid, the most powerful of the two, is equipped with a Samsung Exynos5422 system-on-a-chip (SoC) including a Cortex-A15 2.0 GHz quad core, Cortex-A7 quad core CPU and an ARM Mali-T628 MP6 GPU. The B2260 uses a ST Cannes2-STiH410-EJB SoC including a dual ARMcortex-A9 at 1.5 GHz and an ARM Mali-400 MP2 GPU. Both systems are connected to an USB IDS Ueye UI-1221LE-M-GL[3] camera to acquire images. The setup of the Visual Search and Distance Estimation application running on the Odroid XU3 is shown in Fig. 3. For both systems the camera is mounted on the sliding horizontal headpiece of a stadiometer facing the ground; the object to be retrieved (e.g. an A4 print of

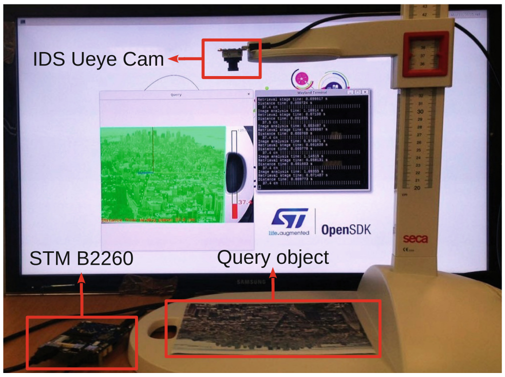


Fig. 3. Setup of the Visual Search and Distance Estimation application running on the STMicroelectronics B2260. The IDS Ueye camera is mounted on the sliding headpiece of the stadiometer, facing the ground, and it is connected with an USB cable to the B2260. The query object is a printed sheet depicting an aerial view of a city and its placed on the bottom platform of the stadiometer. The result of the application is shown on the monitor.

[1] http://www.hardkernel.com/main/products/prdt_info.php?g_code=g140448267127.
[2] http://www.96boards.org/documentation/ConsumerEdition/B2260/README.md.
[3] https://en.ids-imaging.com/store/ui-1221le.html.

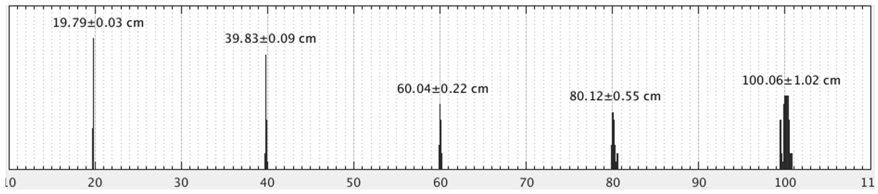


Fig. 4. Histograms of distance estimation results on 5 different set of images acquired at 5 difference distances (20, 40, 60, 80, 100 cm). The intervals represent $\pm 3\sigma$ and they are all below 1% of the estimated distances.

a building aerial view as in Fig. 3) has been placed on the bottom platform of the stadiometer in order to easily record the ground truth distance.

Regarding the quantitative evaluation of the the proposed method's retrieval accuracy, some results could be found in [9].

4.1 Distance Estimation Accuracy

To test the distance estimation accuracy of the proposed method, an ad-hoc test set of 250 images has been created. All the images were acquired by the same camera and all of them depicts the one object acquired at 5 different ground-truthed camera-object distances: 20, 40, 60, 80 and 100 cm. Each image contains a printed sheet depicting an aerial view of a building to mimic a drone application. The proposed method was applied on the described dataset while having a small reference database of 10 images containing the correct image. The object depicted in each query image was always successfully retrieved and in Fig. 4 the histograms of the estimated distances are reported for each true distance. As expected, the standard deviation of the measurements increase with the true distance: the standard deviation reaches up to 0.34 cm at 100 cm, producing a 99% confidence interval of ± 1.02 cm, assuming Gaussian observations; this is equivalent to say that in the 99% of all cases the maximum measurement error is below 1% of the real distance.

Figure 4 also shows that the average measured distance is slightly shifted from the ground-truth one (less than 30 mm in all cases), with a negative bias for small distances and positive bias for large distances. This bias is probably due to some small errors in the camera calibration, such as ignoring small lens distortions.

4.2 Performances

To test the performance of the Visual Search application proposed in this paper and comparing its computational speed on two different systems the same dataset described in Sect. 4.1 has been used. The average time for each module is reported in Table 1. As the queries show only one object, the MRL was not performed, but it would contribute linearly to RS and DE whereas the IA does not depend on

Table 1. Comparison of execution times for each module of the Visual Search and Distance Estimation application obtained on Odroid XU3 and B2260, either exploiting or not the on-board GPU for the IA stage. Only one object is depicted in the query image, so the MRL is never executed. The reference database used has size 10.

		IA	RS	DE	fps
CPU	Odroid XU3	416.9 ms	102.9 ms	0.3 ms	1.9
	B2260	1207.9 ms	159.0 ms	0.3 ms	0.7
GPU	Odroid XU3	105.9 ms	103.7 ms	0.3 ms	4.8
	B2260	157 ms	233 ms	0.3 ms	2.6

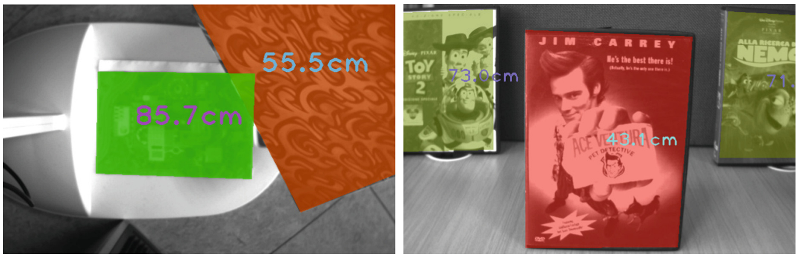
it, as described in Sect. 3.3; moreover, the reference database contains 10 images (not to be confused with the test dataset which, on the other hand, contains 250 images).

As it can be noticed from Table 1 the measured average speed of the CPU implementation is 1.9 fps on the Odroid XU3 and 0.7 fps on the B2260. This difference is mainly due to the inequality between the CPUs of the two devices. Moreover, as can be observed, the heavier computational stage of the VS is the IA that is about 4 times slower than the RS part (while the impact of the DE time is negligible). On the other hand, the RS scales linearly with the dimension of the reference database while the IA is independent from it. The RS time is also indirectly influenced by the use of the GPU in the IA because the descriptors are not computed in the exact same way as in the CPU implementation. For this reason the RS time measured could be slightly different between the GPU and CPU implementation of the VS, as it is in the case of the B2260. The Visual Search and Distance Estimation overall frame rate exploiting the system GPU is 4.8 fps on the Odroid XU3 and 2.6 fps on the B2260.

5 Conclusions

In this work we described and implemented a reliable Visual Search pipeline. In order to be able to work in a real case scenario the proposed application is able to retrieve multiple objects from the same query image. Moreover, due to the necessity in many real case scenarios to also spatially localize the retrieve object the proposed VS application is coupled with an accurate distance estimation module.

In particular, the Visual Search and Distance Estimation application proposed in this paper, takes as input a query image depicting one or more objects of interest, extracts some relevant features, searches inside a database of images the most similar one(s) and finally estimates the relative position between the camera and the pictured object(s). The distance estimation is efficiently performed exploiting the homography already estimated in the VS pipeline, in particular the Retrieval Stage, Sect. 3.2.



(a) Results on 2 objects (b) Results on 3 objects

Fig. 5. The Visual Search and Distance Estimation output when the camera views more than one object. Objects highlighted in red are estimated to be closer to the camera while more distant ones are colored in green. (Color figure online)

As reported in Sect. 4.1, this method has proved to be very accurate when estimating object distances between 20 cm and 100 cm with a 99% confidence interval of just 1 cm. The complete pipeline has also been implemented on two different embedded devices (STMicroelectronics B2260 and Hardkernel Odroid XU3) with different computation capabilities, both connected to an USB camera. Furthermore, some qualitative results are also reported in Fig. 5. In particular the proposed method running on a B2260 is applied at two different query images. All the objects depicted in the two images are localized and their distance from the camera is displayed.

Two different implementations have been tested for each device, one exploiting the on-board GPU and the other only using the CPU, and, as shown in Sect. 4.2 multiple frames could be processed in each second achieving near real-time performances for the GPU implementations for both devices. Moreover, in the Visual Search and Distance Estimation method described in this paper, the object identification procedure is performed for every frame acquired by the camera but this is not necessary; in fact, this method could be easily coupled with a fast object tracking module performed only after retrieving the depicted object(s), as shown in [12], leading to an increment of the overall performances.

Finally, the proposed method could also be implemented on a distributed system, in a similar way as in [9], in which the embedded device(s), connected to a server, executes only the Image Analyzer part leaving the RS and the distance estimation to be executed on the central node. This would grant the application to run in real time (the IA spends 105.9 ms in the Odroid XU3 GPU implementation, 9.4 fps) also working on a big database (located in the server).

Acknowledgments. This work was supported in part by the H2020 European project COSSIM.

References

1. Gill, H.: Cyber-physical systems: beyond ES, SNs, SCADA. In: Presentation in the Trusted Computing in Embedded Systems (TCES) Workshop (2010)

2. Niblack, W.C., Barber, R., Equitz, W., Flickner, M.D., Glasman, E.H., Petkovic, D., Yanker, P., Faloutsos, C., Taubin, G.: The QBIC project: Querying images by content, using color, texture, and shape. In: Storage and Retrieval for Image and Video Databases (SPIE) (1994)
3. Pentland, A., Picard, R.W., Sclaroff, S.: Photobook: content-based manipulation of image databases. Int. J. Comput. Vis. **18**(3), 233–254 (1996)
4. The moving picture experts group website. http://mpeg.chiariglione.org/standards/mpeg-7/compact-escriptors-visual-search
5. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vis. **60**(2), 91–110 (2004)
6. Pagani, A., Stricker, D.: Structure from motion using full spherical panoramic cameras. In: ICCV Workshops, pp. 375–382. IEEE (2011)
7. Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I.D., Leonard, J.J.: Simultaneous localization and mapping: present, future, and the robust-perception age. CoRR, vol. abs/1606.05830 (2016)
8. Hartley, R., Zisserman, A.: Multiple View Geometry in Computer Vision, 2nd edn. Cambridge University Press, New York (2003)
9. Paracchini, M., Marcon, M., Plebani, E., Pau, D.P.: Visual search of multiple objects from a single query. In: 2016 IEEE 6th International Conference on Consumer Electronics-Berlin (ICCE-Berlin), Berlin, Germany, pp. 41–45. IEEE (2016)
10. Haralick, R.M., Shapiro, L.G.: Computer and Robot Vision. Addison-Wesley Longman Publishing Co. Inc, Boston (1992)
11. Canny, J.: A computational approach to edge detection. IEEE Trans. Pattern Anal. Mach. Intell. **8**(6), 679–698 (1986)
12. Plebani, E., Buzzella, A., Pau, D.P., Marcon, M.: Mixing retrieval and tracking using compact visual descriptors. In: 2013 IEEE 3rd International Conference on Consumer Electronics-Berlin (ICCE-Berlin), Berlin, Germany. IEEE (2013)