Parallel SIFT-detector implementation for images matching

Anton I. Vasilyev, Andrey A. Boguslavskiy, Sergey M. Sokolov
Keldysh Institute of Applied Mathematics of Russian Academy of Sciences, Moscow, Russia
{ahbac, anbg}@mail.ru sokolsm@keldysh.ru

Abstract

This paper describes the parallel SIFT-detector implementation on the basis of the NVIDIA CUDA technology for the images matching. The SIFT-detector implementation was applied for the images matching in the stereo-system mounted on the moving car and for images from the onboard UAV-camera.

Keywords: *real-time computer vision, SIFT, CUDA, images matching.*

1. INTRODUCTION

Programmability of general purpose graphic processor units does them attractive for computer vision tasks in order to accelerate the processing and reduce the computational load of the central processor unit. One of the widespread GPU programming technologies is NVIDIA CUDA. The unified architecture of a set of graphic processor units NVIDIA GeForce/GTX and programming model with application of C-like language [10, 11] lies in its basis.

In the given work the GPU application for the SIFT-detector [1] implementation is described. This algorithm is often applied to detect reliable repeated features (blobs/keypoints). The difficulty of its application in real-time systems in practice is connected with the essential computing time consumption for the image preprocessing and the descriptors generating. Parallel implementation on the basis of CUDA has allowed to apply SIFT for the tie points searching in the stereoscopic visual system mounted on the moving car and for images from the onboard UAV-camera.

2. PARALLEL IMPLEMENTATION OF THE SIFT-DETECTOR WITH THE USE OF CUDA

There is a series of published papers on the subject of the SIFT-detector implementation for GPU, for example, [3-8], including open source on C++ [2] and CUDA [6, 7]. The implementation [7] is not documented and the work is presented without any time evaluations in use. In turn, in work [6] the most high-efficiency time evaluations are showed for realisation with shaders. In work [9] the comparison procedure of n-dimensional vectors on the basis of Euclidean metric with CUDA and the primitives library of computing linear algebra CUBLAS [10, 11] is described. A distinctiveness of the many papers is to evaluate an execution time of a general procedure, whereas in applied tasks it is possible to get an optimization and time gain due to the prior problem knowledge consideration. For example, there is no necessity to calculate the blobs orientations in the pure camera translation mode (without rotation). This speeds up the formation time of the descriptors array and reduces the processing time. Another example: the physical limits consideration allows to reduce an execution time of the descriptors matching, when the comparison of descriptors from the certain image area is performed, instead of the comparison of all descriptors. Thus, at designing of the own SIFT-detector implementation for the images matching, two main purposes were put: the possibility of the detector tuning to take into account features of the concrete applied problem and the possibility of the reuse in modular systems of computer vision of real time. As a whole, it is possible to tell that in the chosen approach to the SIFT-detector algorithm implementation was made the GPU-acceleration process using the GPU memory economically.

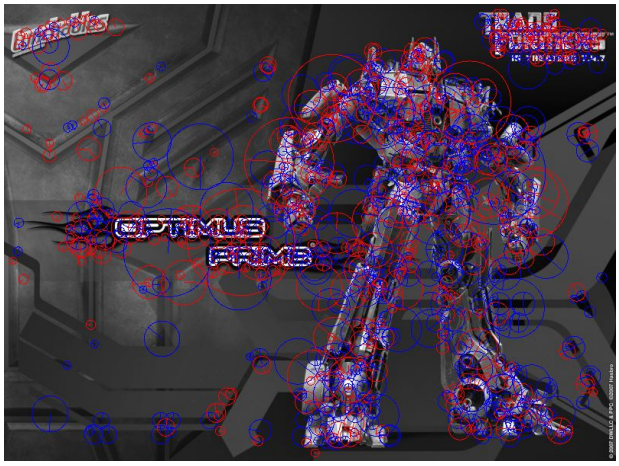


Figure 1: Example of the image processing (800x600, the image is taken from [6] to compare implementations)

2.1 Features of the SIFT-detector implementation

In the SIFT-detector [1] it is possible to emphasize following basic stages:

1) The amount of octaves (and sublevels in the octave) for the image is defined (an amount of the reduced copies with sizes multiple to degrees 2).
2) Formation of the scale-space representation (SSR) for the image: for each octave - the array construction of images, smoothed by Gauss filter (Gss array further), the array size depends of a number of sublevels
3) Formation of the scale-invariant map of edges. Subtraction of the adjacent smoothed images in Gss array is performed for each octave, i.e. outcomes of subtractions are added into the array (DoG array further).
4) The keypoints selecting for each octave: search of local extremes (pixels) for everyone DoG and between adjacent DoGs.
5) Qualification (subpixel position and scale) is fulfilled for each keypoint.

6) An orientation is assigned for each keypoint: the histogram construction of the gradient orientations in a neighborhood of blob (Gss) is fulfilled for corresponding (scale keypoint) smoothed image; the orientation corresponding to maximum bin gets out as a blob orientation.
7) The descriptor is calculated for each blob: the histograms array of gradient orientations is formed round a blob.

Implementation of the described algorithm with the use of the NVIDIA CUDA technology was constructed as (the algorithm stages are presented on figures 2-6):

1) Memory allocation (on GPU):
   - GPU-memory allocation for the image and its copying into the GPU-memory.
   - Memory allocation for SSR on GPU.
2) The octaves and sublevels processing is shown on figures 2-3 (it is realised sequentially for economic use of RAM GPU-memory; green and blue blocks are invoked GPU-memory for current iteration, at that green blocks use the matrix representation of GPU-data):
   - The Gss array generating (it is fulfilled on GPU only).
   - The partial DoG array generating (it is fulfilled on GPU only).
   - The local extremes search and subpixel qualification of them in the matrix representation (it is fulfilled only on GPU).
   - The keypoints redistribution (fig. 3): from the matrix into the linear array (it is fulfilled on GPU and CPU).
3) The orientations assigning for each blob is presented on figures 4-5 (after, all octaves were processed)
   - The orientations calculation for each blob: SSR is used (it is fulfilled on GPU only).
   - If several orientations are defined for one blob, the redistribution is performed in order to construct the array of blobs, having one orientation (it is fulfilled on GPU and CPU).
4) The descriptors generating: SSR is used (it is fulfilled on GPU only).

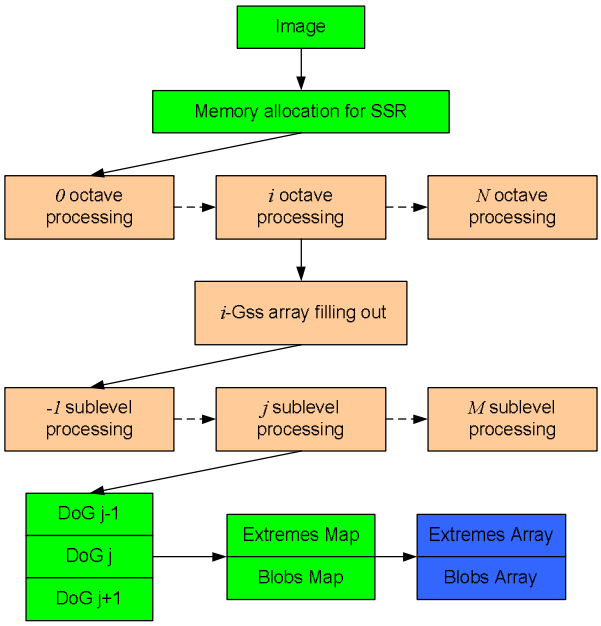


Figure 2: General scheme of the blobs extraction

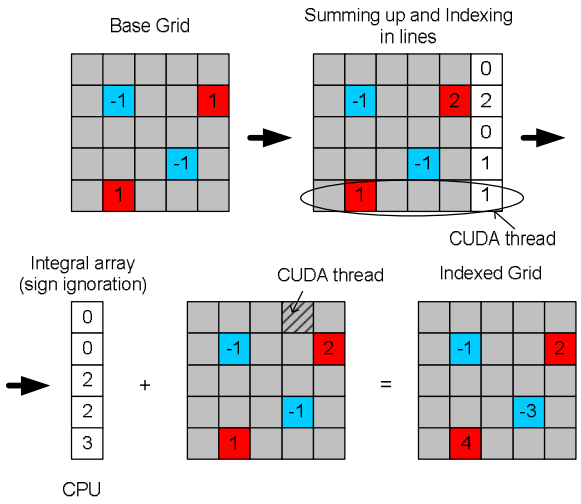


Figure 3: Example of the blobs redistribution (the base grid is extremes map; the indexed grid is used to convert a map to linear array; the integral sum is used to form the integral array)

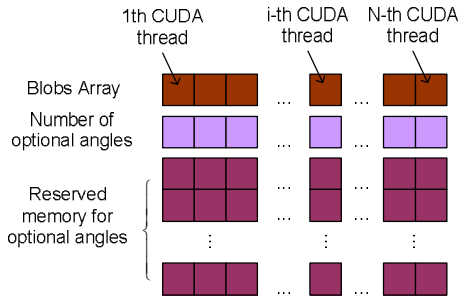


Figure 4: General scheme of the orientations calculation

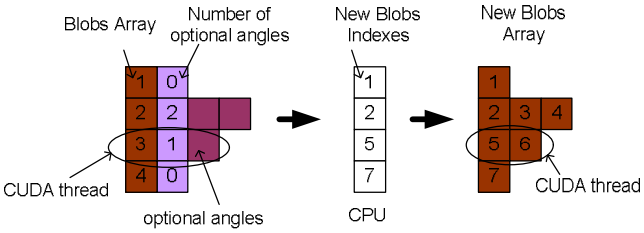


Figure 5: Example of the orientations calculation and reindexation (The integral sum using for the formation of new indexes array)

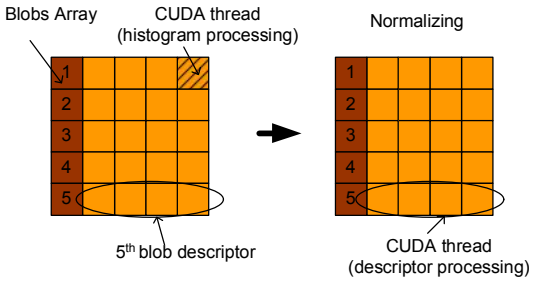


Figure 6: Example of the descriptors generating

Essential differences of our implementation from work [6]:

1) Texture memory is not used for the images arrays storage, as a consequence, there are losses in the access speed but indexation on the image SSR becomes simpler.
2) The octaves and sublevels processing is realised sequentially (economic use of RAM GPU-memory).
3) The data redistribution (reindexation of array elements and the transform from the sparse matrix representation into the linear representation also) has been realised with the use of GPU and CPU.
4) The orientations calculation is performed in parallel for all blobs on the SSR image, GPU and CPU are used for reindexation of the blobs array
5) The Descriptors generating is performed in parallel for all blobs on the SSR image, and the each histogram of descriptor is formed by the separate CUDA thread.
6) The octave index and extreme type are used as blob parameters
7) The detector tuning includes next parameters:
   - Start octave index (down-sampling or up-sampling for the base octave)
   - the amount of octaves and sublevels
   - the maximum quantity of multi orientations for blobs (inclusive of nothing)
   - the SIFT-descriptor size (a number of histograms and their size)

2.2 Features of the matching implementation

The matching procedure (fig. 7) contains the following operations:

1) The physical limits matrix (Boolean matrix) is prepared on the basis of the discovered blobs
2) The calculation of Euclidean distances matrix is performed with the use of the physical limits matrix

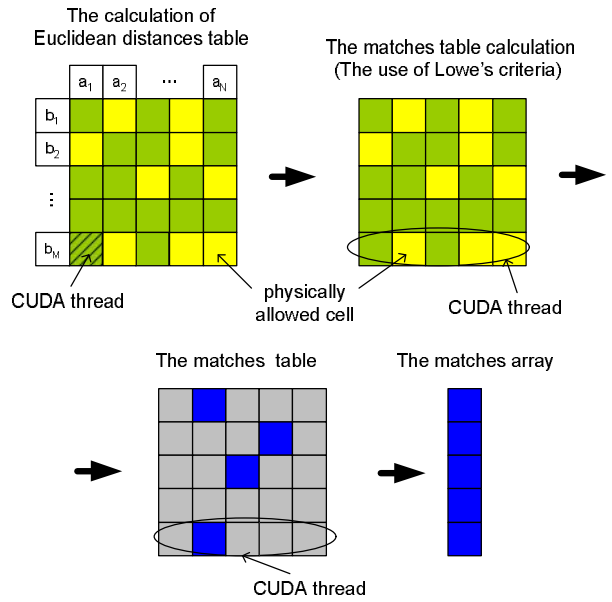


Figure 7: General scheme of the matching procedure (green cells are not admissible physically, blue cells are matches)

3) The matches table is calculated based on the physical limits matrix and Euclidean distances matrix by means of Lowe's criteria [1]
4) The matches table is converted from the matrix representation into the linear array

3. TIME EVALUATIONS AND RESULTS IN PRACTICE

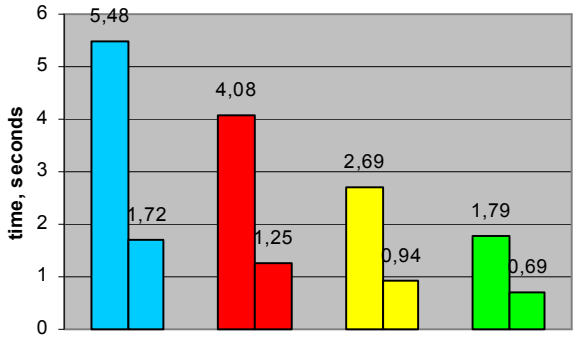
The processing example of the separate image (800x600 pixels, all outcomes are shown for GeForce GTX 260) is represented in tab. 1 for the execution time evaluation of our implementation. Settings were set: 0 start octave index (no up- or down-sampling), 4 octaves with 5 sublevels, no more than two orientations for one blob, 128 elements (4x4x8) in the SIFT-descriptor. On fig. 1 the processing result is shown (blobs/extremes: red - minima, dark blue – maxima). The picture (fig. 1) is taken from work [6] for the purpose of comparison with the published implementation [6].

Table 1: Time evaluations

Keypoints extraction (subpixel position and scale, the linear array formation)			37.5 ms/974 blobs	
The orientations assigning and regrouping			5.1 ms/1139 blobs	
The descriptors generating			43.89 ms	
Total time			87.2 ms	
Key points extraction				
Octave	0	1	2	3
The Generating of Gss-arrays pyramid, ms	13.25	3.13	1.18	0.68
Transform from sparse matrix into linear array (indexing), ms	2.36	1.08	0.67	0.475
Blobs searching (including indexing), ms	10.47	4.48	2.71	2.32
A number of the discovered blobs	612	256	89	17

The received time evaluations concede the implementation (based on shaders) presented in work [6], for example, the image processing on fig. 1 is about 53.9 ms/1052 blobs (in the multi orientation mode). However in our implementation the octaves processing is performed in a consecutive way instead of a parallel way. Thereby resources of RAM GPU-memory are used more economically, as it's important for the UAV-shots processing. For example, the pair of shots (4016x2672 pixels everyone) is formed simultaneously for twin-lens UAV-camera. In turn, the processing of one such shot took about two seconds, whereas it's not possible to process such shot without down-sampling in [6] (the application was downloaded from the site).

The physical limits consideration for the two UAV-shots matching (2008x1336 in size, figures 9-11 in supplementary materials) is presented on the plot 1. The processing time of each of pictures is about 500 ms, at that size of blobs array is about 10.000 elements. The histograms are shown for two cases, when descriptor contains 128 and 32 elements (big and small bins).



Plot 1: The matching time evaluations for UAV-shots with the physical limits consideration (128- and 32- dimensional descriptor): cyan bins – general procedure (no limits), red bins consider the motion without rotation (no orientations), yellow bins – the motion without rotation and the consideration of similar extremes types, green bins – the motion without rotation, the consideration of similar extremes types and octaves indexes

The processing of two shots (nonsequential shots of video, 720x576 pixels in size, figures 12-13 in supplementary materials) took about 300 ms: image 1 – 96 ms/1377 blobs, image 2 – 97 ms/1437 blobs, matching (without any physical limits) – 108 ms/67 pairs. If we use the physical limits (without rotation, similar extremes types, 32 elements in the descriptor) then the processing time is about 108 ms: image 1 – 44 ms/1251 blobs, image 2 – 45 ms/1341 blobs, matching – 19 ms/120 pairs.

Our implementation work analysis by means of the CUDA visual profiler is presented on fig. 8 (in supplementary materials). From drawing it can be seen, that the greatest time is spent for the SIFT-descriptors generating and convolution operation, and the summarized contribution for copying operation of the data between CPU↔GPU makes less than 5 %.

4. CONCLUSION

In the given work the images matching based on the SIFT-detector is shown for two various applied problems: the video processing for stereoscopic measurements and the UAV-shots processing for mosaic formation. The serial-parallel scheme of the blobs extraction, the possibility of the detector tuning and the physical limits consideration allowed to increase the processing speed without essential losses of the used GPU-memory. The next step of development for this implementation will be an inclusion the CUFFT library in order to form the image scale-space representation and the small-size octaves processing in parallel.

5. REFERENCES

[1] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” International Journal of Computer Vision, vol. 60, pp. 91–110, 2004

[2] A. Vedaldi. (2011) Sift ++ source code and documentation. http://www.vlfeat.org/

[3] Warn S., Emeneker W., Cothren J., Apon A. “Accelerating SIFT on Parallel Architectures”, Cluster Computing and Workshops, 2009

[4] S. Sinha, J.-M. Frahm, M. Pollefeys, and Y. Genc, “Feature tracking and matching in video using programmable graphics hardware,” Machine Vision and Applications, March 2007

[5] S. Heymann, K. Muller, A. Smolic, B. Frohlich, and T. Wiegand. SIFT Implementation and Optimization for General-Purpose GPU, in WSCG ’ 07, 2007

[6] C. Wu (2011) "SIFTGPU: A GPU implemenatation of scale invariant feature transform (SIFT)", http://www.cs.unc.edu/~ccwu/siftgpu/#lowesift

[7] Bjorkman M., A CUDA implementation of SIFT http://www.csc.kth.se/~celle/

[8] Kayombya G.-R., "Implementation and Optimization of SIFT on an OpenCL GPU", 2010, http://beowulf.csail.mit.edu/18.337/projects/reports/Kayombya_report.pdf

[9] V. Garcia, E. Debreuve, F. Nielsen, M. Barlaud "KNN Search: fast GPU-based implementations and application to high-dimensional feature matching", ICIP, 2010

[10] NVIDIA, “CUDA technology,” 2011, http://www.nvidia.com/object/cuda_home_new.html

[11] Borescov A.V., Harlamov A.A. “CUDA basics”. – Moscow, DMK-Press, 2010 (In Russian).

About the author

Anton Vasilyev, Keldysh Institute of Applied Mathematics of Russian Academy of Sciences, postgraduate student

Andrey Boguslavskiy, Keldysh Institute of Applied Mathematics of Russian Academy of Sciences, senior scientist

Sergey Sokolov, Keldysh Institute of Applied Mathematics of Russian Academy of Sciences, leading scientist

SUPPLEMENTARY MATERIALS

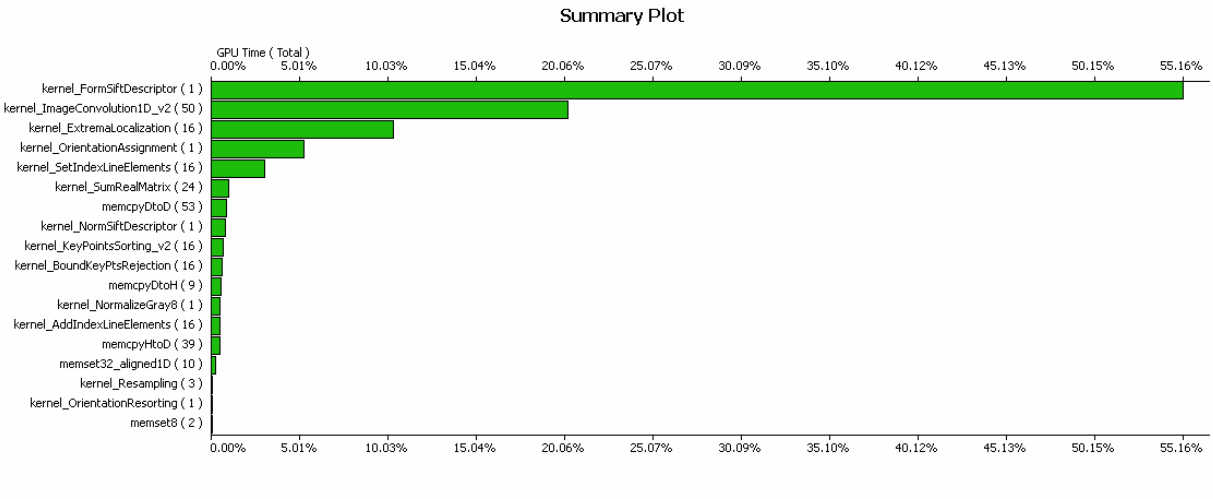


Figure 8: The program work analysis outcomes by means of the CUDA visual profiler

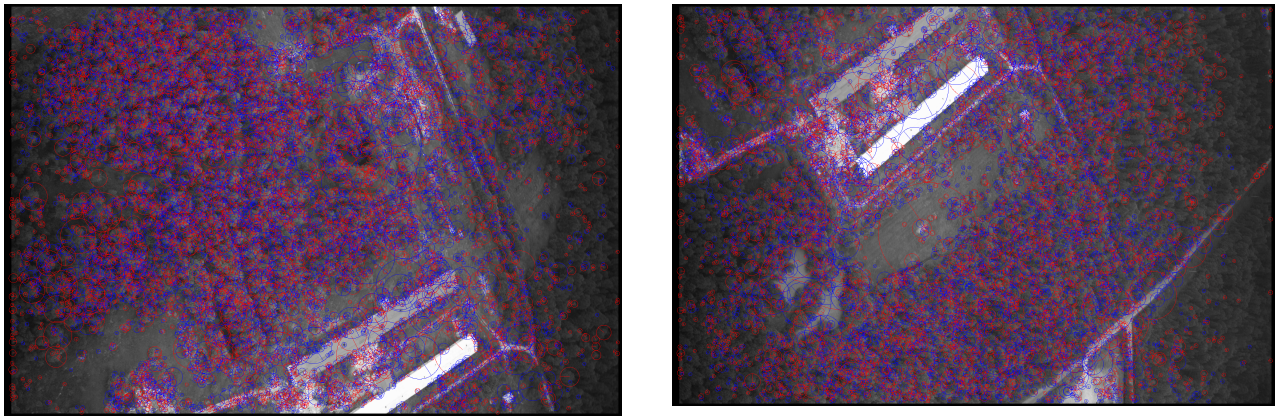


Figure 9: The UAV-shots processed by the SIFT-detector (7 octaves, 5 sublevels, 2 orientations): 539 ms/11058 blobs and 539 ms/9258 blobs

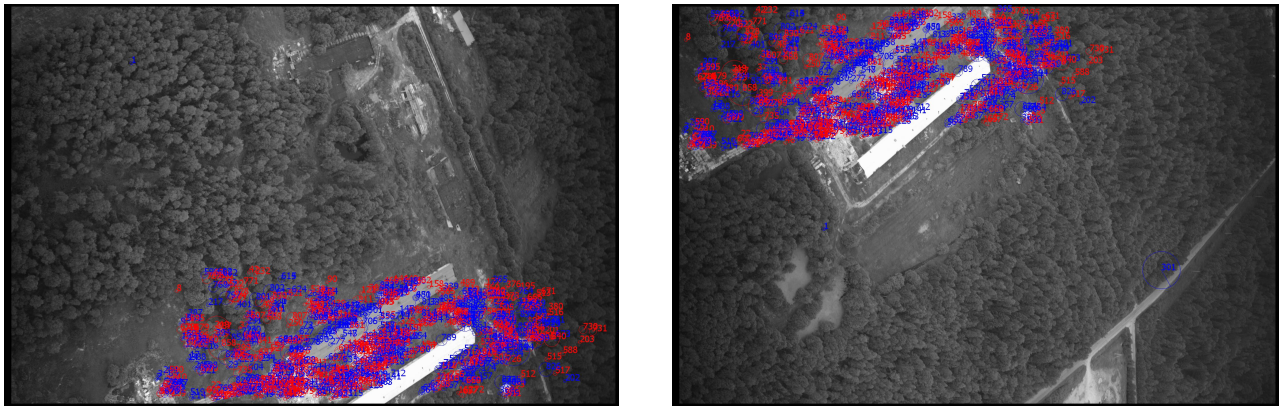


Figure 10: The UAV-shots matching (128-dimensional descriptors, without RANSAC filtering): 5.47 s/865 pairs

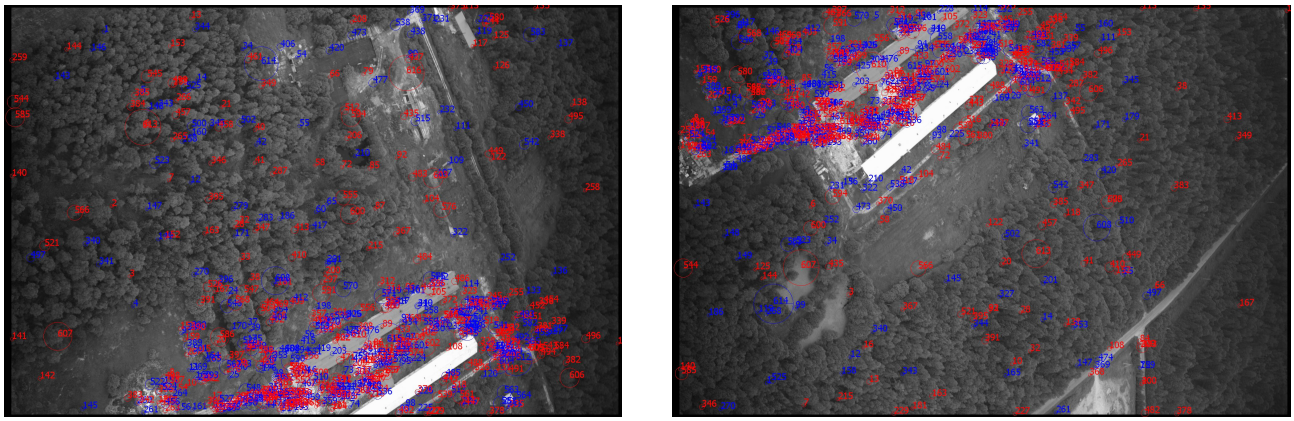


Figure 11: The UAV-shots matching (4 octaves, 5 sublevels, 0 orientations, 32-dimensional descriptors, the consideration of similar extremes types and octaves indexes): 201 ms/9749 blobs, 201 ms/8104 blobs, 695 ms/616 pairs

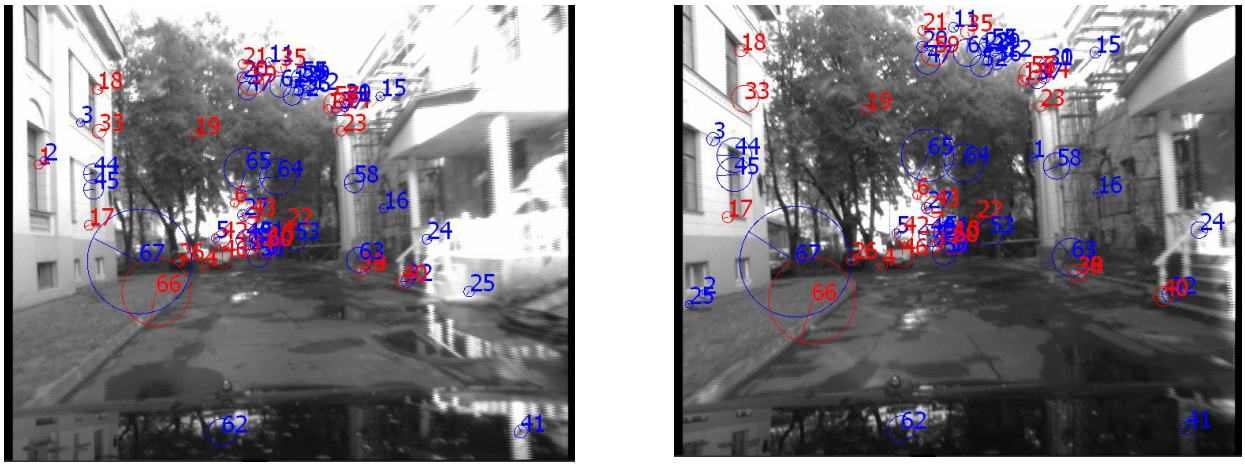


Figure 12: The shots matching from the camera mounted on the moving car (4 octaves, 5 sublevels, 2 orientations, 128-dimensional descriptors, no limits): 96 ms/1377 blobs, 97 ms/1437 blobs, 108 ms/67 pairs

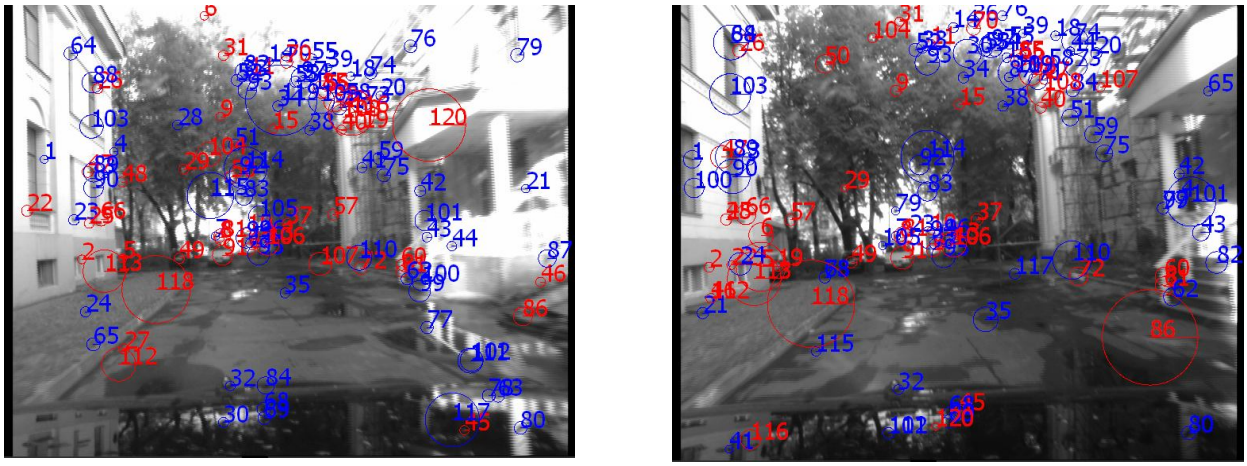


Figure 13: The shots matching from the camera mounted on the moving car (4 octaves, 5 sublevels, 0 orientations, 32-dimensional descriptors, the similar extremes types): 44 ms/1251 blobs, 45 ms/1341 blobs, 19 ms/120 pairs

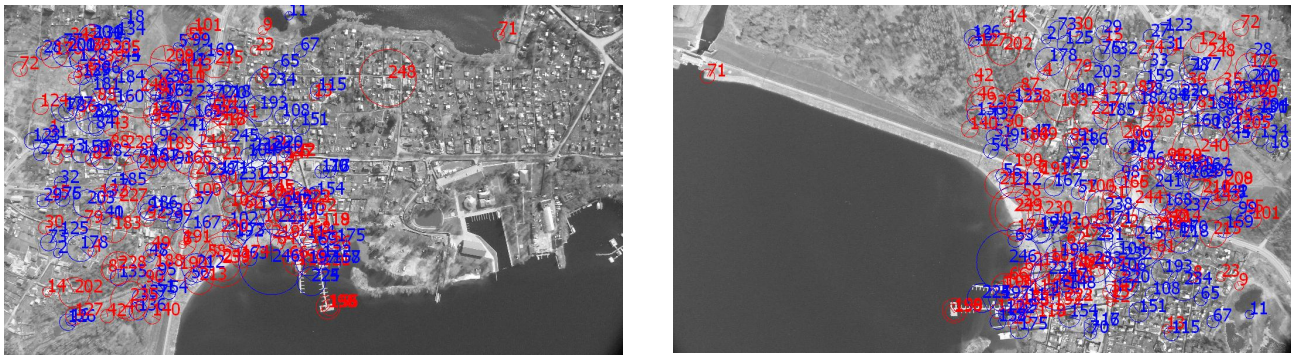


Figure 14: The UAV-shots matching (4224x2376 in size, start octave index 2, 7 octaves, 5 sublevels, 2 orientations, 128-dimensional descriptors, the consideration of similar extremes types and octaves indexes): 102 ms/1019 blobs, 91 ms/695 blobs, 15 ms/248 pairs