Comparison of tracking algorithms implemented in OpenCV

Peter Janku[1,a], Karel Koplik[2], Tomas Dulik[1], and Istvan Szabo[3]

[1] *Tomas Bata University in Zlin, Faculty of Applied Informatics, Nad Stranemi 4511, 760 05 Zlin*
[2] *Brno University of Technology, Faculty of Information Technology, Bozetechova 2, 612 66 Brno*
[3] *UNIS, a.s., Jundrovská 33, 624 00 Brno, Czech Republic*

Abstract. Computer vision is very progressive and modern part of computer science. From scientific point of view, theoretical aspects of computer vision algorithms prevail in many papers and publications. The underlying theory is really important, but on the other hand, the final implementation of an algorithm significantly affects its performance and robustness. For this reason, this paper tries to compare real implementation of tracking algorithms (one part of computer vision problem), which can be found in the very popular library OpenCV. Moreover, the possibilities of optimizations are discussed.

1 Introduction

The problem of object tracking is an important area of computer vision. Tracking algorithms can be used for various applications like traffic control and analysis, face and full-body person recognition, image matching etc. Although methods of object tracking has been studied for decades and many algorithms were developed, the current state of the art is far from having a perfect, universal solution for every use case. Because of enormous set of variables, parameters and environments (background, illumination, characteristics of the tracked objects, etc.), it is nearly impossible to develop universal tracking algorithm. [1, 12, 15]

Moreover, the selection of proper algorithm depends not only on the algorithm principle, but also on its implementation. Programming language, compiler and manual optimization can significantly affect the performance and robustness of the algorithm. For this reason, we decided to carry out an analysis of pre-implemented tracking algorithms available in OpenCV library.

The OpenCV is a well known library, which integrates necessary structures and tools for computer vision algorithms; in addition, it integrates large set of different pre-implemented algorithms solving different parts of object tracking problem. Moreover, different optimization methods including parallel programming, GPU computing etc. can be used for tuning the selected algorithm performance. [17]

This paper tries to bring a comparison of tracking algorithms implementations, which are included in the OpenCV library. Moreover, basic principles of presented algorithms and optimizations are discussed.

[a]Corresponding author: janku@fai.utb.cz

2 Algorithms description

While tracking is a very common computer-vision problem and OpenCV is a widely used C++ computer-vision library, sadly, only few algorithms are available in the library. For our testing, we have used three feature detectors, three pure trackers and one complex tracking framework.

2.1 Feature detectors

Feature detectors are not really trackers, they only try to detect a specified object in each frame individually. It works like this: we have two images, one is an image of the object we want to track and the other is the first frame of a video or frame obtained from live stream. We can obtain the first image from the second image by making rectangular selection.

Feature detectors then try to detect some features in the object image and try to find the best mapping of these features in the current frame. This works very well if the object image is large enough and the object itself has detectable features such as edges and texture. The object can be even freely rotated in the frame plane or slightly rotated in other planes while facing approximately the same direction. In OpenCV, we use functions *findHomography()* to find the transform between matched keypoints and function *perspectiveTransform()* to map the points. We have used a slightly improved code from OpenCV documentation example [6] to test feature detectors in the role of tracker.

The reason why we cannot say feature detectors are really trackers is their inconsistency between frames. Trackers follow trajectory, detectors only detect the best match for two images which leads to them being very unstable, especially when mistaking the tracked object for something at the moment more similar to the originally defined object.

The other problem with mere detectors could occur when tracked object is the same as one or more objects in the picture or is a part of repeating structure (e.g. windows, fences, etc.). This can become a problem with tracking in general but it can be helped by focusing on object's location. A this is were the trackers come in.

In OpenCV, there are currently three useful feature detectors: SURF [4], SIFT [2] and ORB [7]. The first two use floating point numbers but are patented. The third one uses integers and is therefore less precise but is faster and has more friendly licence. The basic concepts of SIFT and SURF algorithms and their usage with OpenCV implementation can be found in [5] and [3].

2.2 Pure trackers

For better distinction between this and the following subsection, we will use the term *pure trackers* for trackers which only do tracking and nothing else. Trackers are designed to track the object by following its trajectory and predicting its future locations. This includes correcting an error in the process.

The major problem with *pure trackers* is that they run into difficulties when the camera moves too fast or the object suddenly changes speed or direction. In that case, they may try to detect the object in more predictable location and find something similar instead.

In OpenCV, there are currently three useful pure trackers: MIL [8], Boosting [10] and MedianFlow [9].

2.3 Tracking frameworks

Tracking frameworks are algorithms which attempt to provide the most complex solution for the tracking task at hand. They can be considered as ready-to-use products. They are designed to overcome major tracking problems by constantly adapting to new conditions and correcting all crucial errors. Their disadvantage is bigger memory and processing power consumption, but this is the price to pay for all their advantages.

In OpenCV, there is currently one useful tracking framework: TLD. This algorithm is divided into three mostly independent parts - Tracking - Learning - Detecting. The tracker tries to track a blob of pixels from one frame to another. The detector tries to find similar observed object and correct tracker if necessary, and the learning estimates the detectors errors and updates it in order to avoid errors in future. Thanks to this, TLD can provide stable tracking in long term. Additionally, it is able to adapt to tracked blob changes. [11, 16]

Three basic parts of TLD can run simultaneously, each on separate processor and/or as a separate task. Moreover, each part can be optimized (parallelized) separately due to its character. [11, 16]

Table 1. List of Problems

Abbr.	Name	Description
IV	Illumination Variation	The illumination in the target region is significantly changed.
SV	Scale Variation	The ratio of the bounding boxes of the first frame and the current frame is out of the range.
OCC	Occlusion	The target is partially or fully occluded.
DEF	Deformation	Non-rigid object deformation.
MB	Motion Blur	The target region is blurred due to the motion of target or camera.
FM	Fast Motion	The motion of the ground truth is larger than limit.
IPR	In-Plane Rotation	The target rotates in the image plane.
OPR	Out-of-Plane Rotation	The target rotates out of the image plane.
OV	Out-of-View	Some portion of the target leaves the view.
BC	Background Clutters	The background near the target has the similar color or texture as the target.
LR	Low Resolution	The number of pixels inside the ground-truth bounding box is below limit.

3 Measurement methods

As it was already stated before in similar papers (e.g. [12], [13]), doing a complex comparison of tracking algorithms can be rather tricky. The main reason for that is that tracking itself can be used for many different purposes (tracking faces, people, vehicles etc.) in many different environments and situations (airport, inside a building, roads, nature, fog, day / night, etc.) while the cameras can be of low / high quality, near / far, colour / grey-scale, moving / stabilized and so on. What's more, tracking itself consists of overcoming different types of problems such as rotation or partial occlusion of tracked object, changing light conditions, blurred frame due to a fast camera movement, etc. The above stated means that a very large dataset is needed. Fortunately, Yi Wu et al. already managed to collect most commonly used testing videos and offer it freely with annotations [14]. In [12], they use this dataset to test tracking algorithms which are available with source code. Each video in the dataset has been described by listing major problems which has to be dealt with when implementing a good tracking algorithm. We include their problem definitions in Table 1 [12].

3.1 Success evaluation

In each video we had used the following formula for each frame: $C_{suc}(f) = \frac{|r_t \cap r_g|}{|r_t \cup r_g|}$ where $C_{suc}(f)$ is a success criterion function for frame f; r_t is bounding rectangle returned from tracker and r_g is the bounding rectangle provided by ground truth. We basically take the area of the interception and divide it by the area of the union of above defined rectangles. This will give as the overlapping ratio which is be considered a success when larger than 0.5. [12]

3.2 Precision evaluation

As a measure of algorithm precision, we decided to use the scale of rectangle obtained from the tracker considering the ground truth while the best precision is equal to 1 if we use this formula: $C_{prec}(f) = \frac{|r_t|}{|r_g|}$ where $C_{prec}(f)$ is the precision criterion function for the currently processed frame f.

3.3 Time demands evaluation

To objectively measure time demands of each algorithm, we propose to simply measure time for each frame: $C_{tim}(f) = t$ where $C_{tim}(f)$ is the algorithm time demands criterion function; t is the time it took to process the current frame f.

3.4 Performance evaluation

How did algorithms performed while facing each of the problems defined in Table 1 was evaluated in the same way as with the $C_{suc}(f)$ (see above) but only with relevant videos.

3.5 Difficulties

A few issues had to be resolved before collecting and processing data. The initial idea was to compare how much do the bounding rectangle provided by the tracker and the ground truth bounding rectangle overlap while also measuring the processing time. The first problem was that sometimes the tracker rejects the object from the ground truth when being initialized or completely looses the object during tracking. Since each frame produces data, even when tracking unsuccessfully, it can then imply the tracker was not able to deal with all tracking problems in the currently processed video. Typically, during benchmarking this is fixed by triggering reinitialization after a few failed frames. We decided to set retrigger threshold at 30 failed frames. This may mean that some trackers will appear more effective (the ground truth should not be available) but, as we supposed, 30 frames were enough to substantially penalize them. Other problems were discovered when trackers on very rare occasions behaved unpredictably and the output data were out of expected boundaries. Sometimes, it took several seconds to process a relatively small frame or the object turned out to supposedly cover almost whole frame while the ground truth bounding rectangle was much smaller. Such output was reclassified as failure and the data were constricted so they could both serve as score penalty and reasonable statistical record.

4 Results

In this section, we are going to discuss all of the benchmarking results. With great advantage, we used box-and-whisker type of charts to compare algorithms with each other, because this type of chart allows us to nicely compare the acquired data. The box represents the range of 50% of results. Inside each box, the horizontal line depicts the middle value, and the bottom-most and top-most lines are minimum and maximum values.

4.1 Success

In each video, we had counted number of frames which had results within the defined limits ($C_{suc}(f) \geq 0.5 \land C_{scale}(f) < 2.0 \land t(f) < 1$) and divided this number by the total number of frames. Surprisingly, TLD did not turn out to be the best algorithm (see Figure 1) despite being so complex.

There may be two reasons for this. The first one may be, that algorithms MIL and Boosting, categorized by us as *pure trackers*, were more successful due to reinitialization from the ground truth. The second reason could be, that TLD is so heavily correcting itself, that the bounding rectangle in each frame shifts a lot and therefore does not follow the ground truth bounding rectangle so nicely (the intersection is often less then 50%). It doesn't mean that TLD looses the object, the opposite is true, but it fails to center the object in its bounding rectangle.

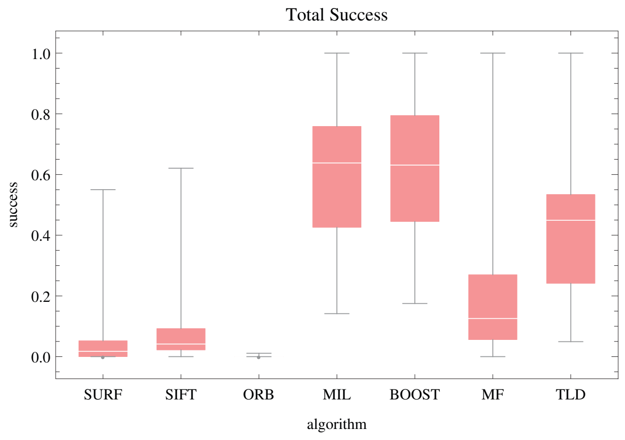


Figure 1. Total success chart

4.2 Precision

To measure precision of each algorithm, we decided to use the scale ratio. If the object is located precisely, the bounding rectangle dimensions should be more similar to the ground truth than if it is not. For this criterion, we had eliminated all the unsuccessful data, which means that the maximal acceptable scale ratio was 2. As you can see at Figure 2, all algorithms had their extreme values bellow this upper limit and above the lower limit, which was, off course, 0. To tell which algorithm did well according to this criterion, we need to look how close is their middle value to 1.0 and how small is the range between their minimal and maximal values.

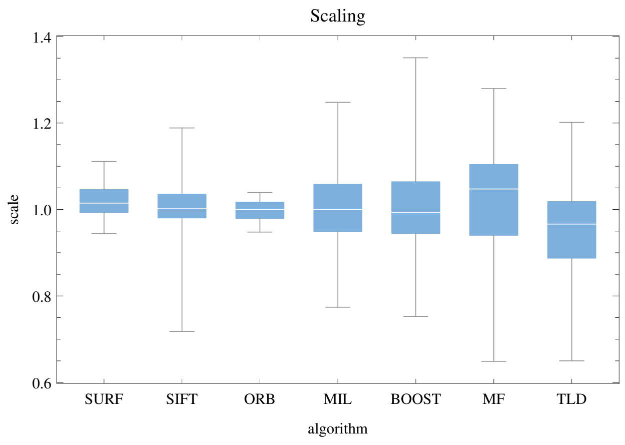


Figure 2. Scale chart

4.3 Time demands

For this criterion, we had measured the time each algorithm needs for processing a single frame. It was necessary to eliminate very short but also very long times because the box-and-whisker chart would be too stretched towards the extremes. You can see at Figure 3 that SIFT and SURF are very slow. That's because they work with floating point values and their inner calculations take more time. The rest of the algorithms are faster, while the slowest one is TLD. Slowness of the TLD algorithm is caused by its relative robustness and lack of optimizations of its OpenCV implementation.

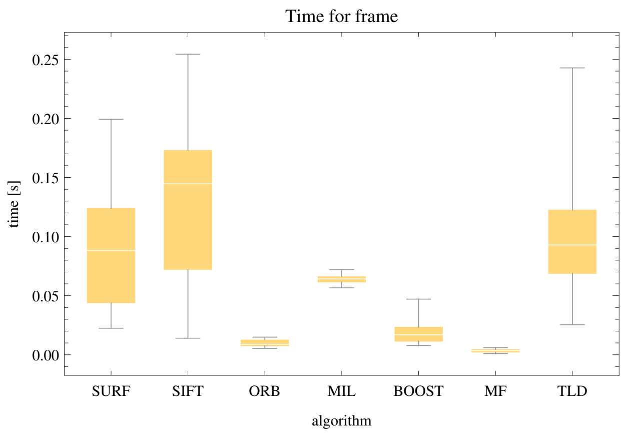


Figure 3. Time demands chart

4.4 Performance

In this section, we present results for each algorithm's performance when dealing with major tracking problems. As shown at Figure 4, the algorithms' success was not so different from their total success. To have a better overview, the Table 2 also shows the average success for each algorithm and problem.

Table 2. Problems' success table

	SURF	SIFT	ORB	MIL	BOOST	MF	TLD
IV	0.057	0.065	0.000	0.644	0.596	0.138	0.372
SV	0.076	0.089	0.001	0.549	0.559	0.218	0.409
OCC	0.051	0.068	0.000	0.572	0.586	0.185	0.400
DEF	0.023	0.044	0.000	0.616	0.596	0.146	0.388
MB	0.099	0.075	0.001	0.521	0.560	0.114	0.425
FM	0.083	0.086	0.001	0.506	0.521	0.165	0.394
IPR	0.064	0.079	0.001	0.500	0.567	0.206	0.366
OPR	0.053	0.060	0.001	0.562	0.585	0.231	0.372
OV	0.158	0.094	0.002	0.468	0.486	0.124	0.293
BC	0.066	0.076	0.001	0.692	0.661	0.153	0.374
LR	0.100	0.081	0.002	0.488	0.545	0.312	0.373

5 Algorithm optimizations

Because this paper deals with specific implementations of algorithms included in OpenCV, it has to be mentioned, that the final speed of algorithms depend not only on its principle, but also on the style of implementation and/or on used optimizations.

The OpenCV library contains one important method for parallelization of operations in computer vision algorithms. It is a set of virtual functions and classes, by which the performed operation, usually programmed using *for* loop, is defined as parallelisable operation. Thanks to this, when the OpenCV library is compiled with some parallelization framework, the marked section of algorithm is automatically distributed into parallel execution.

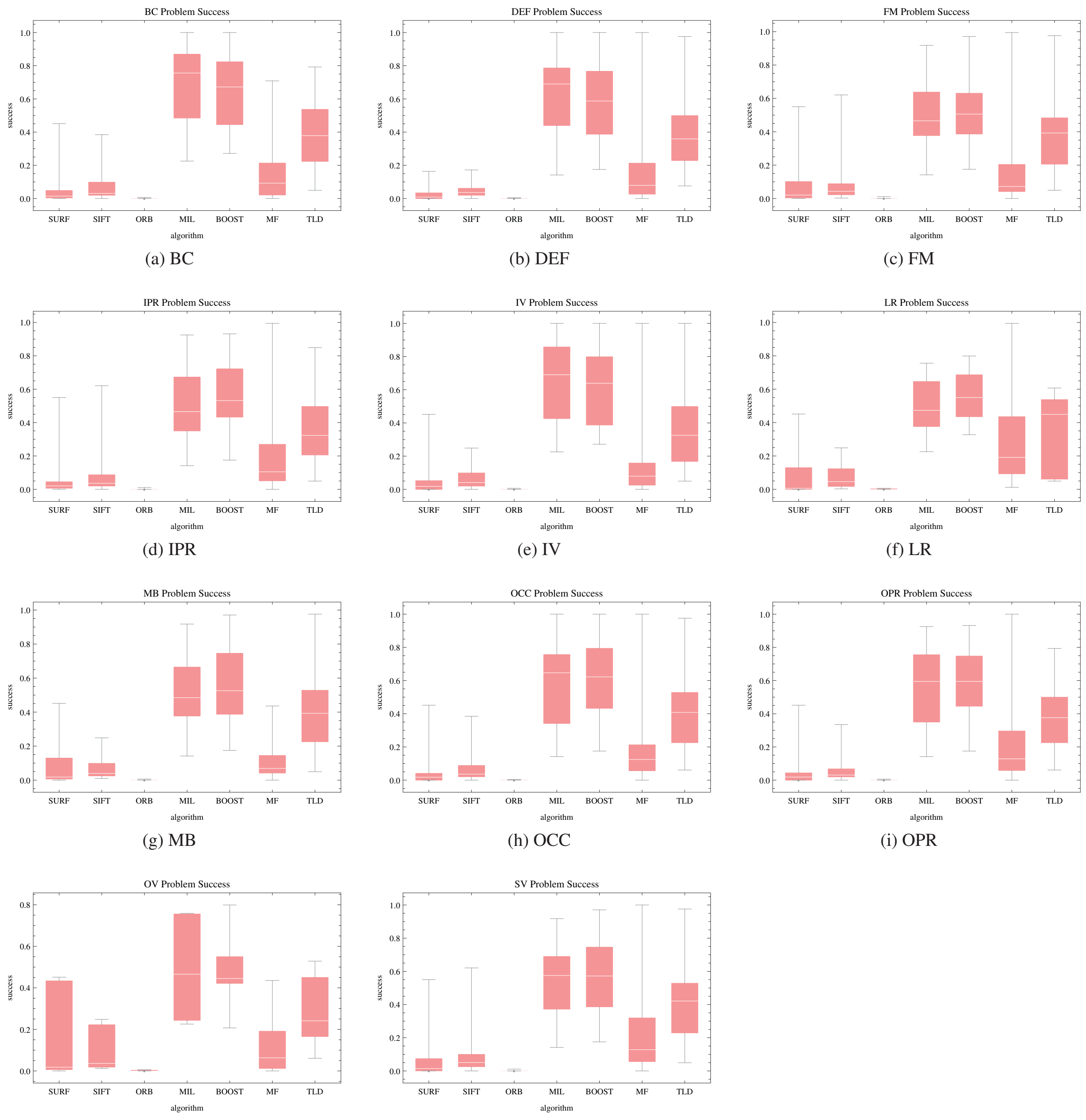
5.1 Available optimizations in OpenCV

The set of available optimization frameworks, which can be used as OpenCV background slightly depends on the platform. All optimizations can be divided into two basic categories. The first category of optimizations works with CPU. Inside this category are the multimedia instructions (SSE, NEON) together with multi-core parallelism methods (OpenMP, TTB). By using multimedia instruction, the one simple mathematical operation can be performed on multiple data concatenated into one processor word. For example if we have a processor with 64 bit wide registers, we can calculate a sum either in one instruction with eight numbers 1 byte wide or with numbers 2 byte wide or 2 numbers 4 byte wide. On the other hand, the multi-core parallelism methods are suitable when we have a CPU with multiple cores. In that way, the algorithm is divided into proper number of independent execution queues.

The second category of optimization uses GPUs as a computing devices. In OpenCV, the support for CUDA (NVidia graphics cards) and OpenCL(Intel and ATI graphic cards) is implemented. Moreover, the Vulcan (universal graphical card computing framework) will be available soon. The principle of this optimization lies in moving the graphical calculation on the GPU cores placed on the graphic card, where the amount of cores is significantly higher than in CPU.

6 Conclusion

The real aspects of implementation of tracking algorithms were discussed in this paper. Selected algorithms were



(a) BC (b) DEF (c) FM
(d) IPR (e) IV (f) LR
(g) MB (h) OCC (i) OPR
(j) OV (k) SV

Figure 4. Problems chart

tested and compared in different conditions and from different points of view.

Already the first criterion - *success* brings unexpected results. The TLD algorithm, which can be considered as the best one from the previous research base, was not as good as expected. It was partly caused by its complexity (it was not able to aim the target blob correctly because of adaptations), and partly by its insufficient implementation in OpenCV.

The second criterion - *precision*, brings constant results from the mean value point of view. At the other hand significant differences can be found on minimum and maximum levels of this measurement. These results can be marked as predictable on the previous research base.

As was described in previous sections, the time comparison of defined algorithms shows that the slowest algorithm was SIRF, directly followed by SURF algorithm. Surprisingly, the TLD algorithm was nearly as slow as the previous two. After deep analysis, it was recognized that the speed of TLD algorithm is limited by its complexity and/or its imperfect implementation, which does not use parallel loops available in OpenCV for speed optimization.

Because of the uncovered imperfections and limitations in algorithm implementation, the more detailed analysis will be created. The future work on this research will be focused of detailed analysis of implementation of these algorithms, including used programming techniques and memory and computing efficiency.

Acknowledgement

This paper is supported by Technology Agency of the Czech Republic (TA ČR) within the Visual Computing Competence Center - V3C project No. TE01020415, by Ministry of Education, Youth and Sports of the Czech Republic within the National Sustainability Programme project No. LO1303 (MSMT-7778/2014) and also by the European Regional Development Fund under the project CEBIA-Tech No. CZ.1.05/2.1.00/03.0089 and the Internal Grant Agency at TBU in Zlin, Project No. IGA/FAI/2016/036.

References

[1] D. Li, B. Liang, a W. Zhang, „Real-time moving vehicle detection, tracking, and counting system implemented with OpenCV", in 2014 4th IEEE International Conference on Information Science and Technology, 2014, s. 631–634.

[2] Lowe, D. G., "Distinctive Image Features from Scale-Invariant Keypoints", International Journal of Computer Vision, 60, 2, pp. 91-110, 2004.

[3] "OpenCV: Introduction to SIFT (Scale-Invariant Feature Transform)", Retrieved May 24, 2016, from http://docs.opencv.org/3.1.0/da/df5/tutorial_py_sift_intro.html

[4] Bay, H. and Tuytelaars, T. and Van Gool, L., "SURF: Speeded Up Robust Features", 9th European Conference on Computer Vision, 2006.

[5] "Introduction to SURF (Speeded-Up Robust Features)", Retrieved May 24, 2016, from http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html

[6] "Features2D + Homography to find a known object", Retrieved May 24, 2016, from http://docs.opencv.org/2.4/doc/tutorials/features2d/feature_homography/feature_homography.html

[7] Ethan Rublee, Vincent Rabaud, Kurt Konolige, Gary R. Bradski: ORB: An efficient alternative to SIFT or SURF. ICCV 2011: 2564-2571.

[8] B. Babenko, M-H. Yang, and S. Belongie, "Visual Tracking with Online Multiple Instance Learning", In CVPR, 2009.

[9] Z. Kalal, K. Mikolajczyk, and J. Matas, "Forward-Backward Error: Automatic Detection of Tracking Failures", International Conference on Pattern Recognition, 2010, pp. 23-26.

[10] Friedman, J. H., Hastie, T. and Tibshirani, R., "Additive Logistic Regression: a Statistical View of Boosting.", Technical Report, Dept. of Statistics, Stanford University, 1998.

[11] Z. Kalal, K. Mikolajczyk, and J. Matas, "Tracking-Learning-Detection," Pattern Analysis and Machine Intelligence 2011.

[12] Wu, Yi et al., "Object Tracking Benchmark", IEEE Trans. Pattern Anal. Mach. Intell. 37 (2015): 1834-1848.

[13] Wu, Ye et al., "Online Object Tracking: A Benchmark", The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2013, pp. 2411-2418.

[14] "Visual Tracker Benchmark. (n.d.)", Retrieved May 23, 2016, from http://cvlab.hanyang.ac.kr/tracker_benchmark/datasets.html.

[15] Z. Kalal, K. Mikolajczyk, a J. Matas, „Face-TLD: Tracking-Learning-Detection applied to faces", in 2010 IEEE International Conference on Image Processing, 2010, s. 3789–3792.

[16] Z. Kalal, K. Mikolajczyk, a J. Matas, „Tracking-Learning-Detection", IEEE Transactions on Pattern Analysis and Machine Intelligence, roč. 34, č. 7, s. 1409–1422, čvc. 2012.

[17] I. Culjak, D. Abram, T. Pribanic, H. Dzapo, a M. Cifrek, „A brief introduction to OpenCV", in 2012 Proceedings of the 35th International Convention MIPRO, 2012, s. 1725–1730.