

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/317803149>

# Evaluation of Visual Tracking Algorithms for Embedded Devices

Conference Paper · May 2017

DOI: 10.1007/978-3-319-59126-1\_8

CITATIONS

3

READS

2,323

4 authors, including:



**François Christophe**

Hämeen ammattikorkeakoulu University of Applied Sciences

40 PUBLICATIONS 195 CITATIONS

[SEE PROFILE](#)



**Tommi Mikkonen**

University of Helsinki

300 PUBLICATIONS 2,350 CITATIONS

[SEE PROFILE](#)

Some of the authors of this publication are also working on these related projects:



Bio-integrated Software Development for Adaptive Sensor Networks [View project](#)



RegOps [View project](#)

# Evaluation of Visual Tracking Algorithms for Embedded Devices

Ville Lehtola, Heikki Huttunen, Francois Christophe, and Tommi Mikkonen

Tampere University of Technology, Tampere, Finland,  
`firstname.lastname@tut.fi`

**Abstract.** Today’s embedded platforms enable executing difficult tasks such as visual tracking. However, such resource-constrained systems are still facing challenges regarding the performance and accuracy in executing these tasks. This paper presents the evaluation of 5 open-source visual tracking implementations available from the contributions branch of the Open Computer Vision (OpenCV) library. This evaluation is performed based on the performance and accuracy of these implementations when embedded in a Raspberry Pi. The algorithms evaluated are On-Line Boosting, Multiple Instance Learning (MIL), Median Flow, Tracking-Learning-Detection (TLD), and Kernelized Correlation Filters (KCF). Even if commercial implementations of these algorithms perform better than their open-source version, the popularity of OpenCV motivates this evaluation. Tests are based on a benchmark of 100 video streams from which the tracking implementations should follow moving objects. The algorithms are evaluated for accuracy using averaged Jaccard indices and for performance by measuring their frame rate. We want to find an open-source implementation that performs well on these two criteria when tested on an embedded platform. Results show Median Flow being the fastest but its accuracy is the lowest. We therefore recommend KCF as it is the second fastest and the most accurate.

**Keywords:** Tracker algorithms, OpenCV

## 1 Introduction

Object detection from still images and object tracking in video streams are important problems in computer vision. During the recent years, there have been many breakthroughs in object detection and localization exploiting the advances in deep learning—often surpassing human level accuracy [6,11]. While it is possible to track objects by re-detecting them in each frame (tracking by detection), there are several drawbacks to this approach: tracking is limited to categories used at training time; the detector may lose track of the object in poor illumination, pose changes, etc.; and most importantly, object detection based on modern deep learning requires heavy computation usually done on a GPU. Therefore, it is often advantageous to approach the problem from the traditional angle of generic object tracking.

Tracking algorithms typically differ from detection in that they learn the changes in object appearance over time. Trackers typically implement two functions: **add** and **update**, with the first initialized every time a new object appears in the scene, and the latter applied subsequently for each frame. The variations between the algorithms reside in the update step, where each method essentially learns the most recent appearance of each target. Moreover, some algorithms may exploit the historical information about motion trajectories of objects, for example in order to limit the search region. Tracking algorithms tend to have a lower computational complexity than a full-scale detection approach would have. Although the accuracy of different tracking algorithms has been widely studied (see, *e.g.*, [13,12]), the computational load has gained less attention. The execution speed varies across algorithms, which easily becomes a limitation when implementing real-time detection in low-resource platforms without any hardware acceleration (such as a GPU).

From the point of view of embedded systems and the increasing interest in the Internet of Things (IoT), the accuracy is often secondary to the use of resources: the burden of both computation and integration tends to dictate the choice of algorithms. Therefore, we consider five tracking algorithms readily implemented in OpenCV library, which is used as a component in many open source projects today. Moreover, we will get an insight into the actual implementations of this widely used library, which are not exactly the same as used in the original papers where the algorithms were first proposed.

This paper is structured as follows. Section 2 introduces the trackers available in the OpenCV library and their original papers. Section 3 introduces the dataset and equipment used in the experiments and defines the methods used to determine the quality of the tested trackers. Section 4 presents the results. Section 5 concludes.

## 2 Tracking algorithms

This section presents a brief description of the trackers evaluated in this study. The algorithms evaluated are available from the contributions branch of the Open Computer Vision (OpenCV) library [2].

**On-line boosting** On-line boosting is a tracking algorithm that considers tracking as a binary classification task [4,5]. At each tracking update step, the algorithm updates the object model by using AdaBoost for training a collection of weak classifiers. The training step uses the target region as a positive example and samples patches from the vicinity of the currently tracked object region. The object location in the next frame is then estimated by applying the binary classifier in the next frame, and choosing the most likely location as predicted by the AdaBoost classifier.

**Multiple Instance Learning (MIL)** Similarly to the on-line boosting approach, Multiple Instance Learning algorithm (MILBoost) poses the tracking

problem as a classification task [14]. For classification, the method uses the multiple instance learning approach, which considers *bags* of objects by grouping similar samples into bags, where each bag is considered an overall positive sample if at least one of the individual samples it contains is positive, and negative otherwise. This attempts to avoid confusing the classifier with sub-optimal samples being labeled as positive, instead giving the classifier a more vague understanding of what positive samples should be like. More recently, an on-line modification of the learning method was proposed [1], which is more suitable for tracking due to lower computational load.

**Median Flow** Median Flow is presented in [9]. The authors present a tracking error measure, where points of each object are tracked both forward and backward in time and the resulting trajectories are compared. By the assumption that a correct tracking method would yield the same but opposite trajectory when running on time reversed input, any divergence of the forward and backward tracking trajectories indicates a tracking error. The authors use this error measure to propose a tracking method where points inside a bounding box are tracked and measured for error, then classified to inliers and outliers by the result. Outliers are filtered out and the bounding box motion is estimated based on the inliers.

**Tracking-Learning-Detection (TLD)** Tracking-Learning-Detection is a method for long-term tracking tasks [10]. The authors describe it as a framework rather than a tracking method as the different stages, tracking, learning and detection, are performed by separate components of the overall system. The goal of the method is to improve the robustness of tracking by disabling the on-line learning if the object is out of frame or completely occluded by other objects, thus avoiding learning from misinformation. The detection component also allows the method to re-detect the object, should it reappear in the video later.

**Kernelized Correlation Filter (KCF)** The Kernelized Correlation Filter method [8] employs the shift invariance property of Fourier transform for designing a fast algorithm for correlation filter based matching. The Fourier transform simplifies the correlation computation to make it extremely fast. Further, the method generalizes by applying the kernel trick to allow nonlinear correlation measures. The authors note that one of the main challenges in tracking is the inability of using a large enough number of training data available from each frame of input due to high computational load. This problem is avoided in KCF due to its lightweight implementation.

### 3 Evaluation method

The primary interest of evaluation was not only to find out how accurate the trackers are, but also to measure the viability of the OpenCV implementations as

real-time tracking solutions on hardware with severe memory and performance constraints. Each algorithm was initialized with the ground-truth bounding box from the first frame of a sequence and only the default parameters for the tracker were used.

**Dataset used for evaluation** The dataset of 100 video sequences used in the evaluation is from the visual tracking benchmark by Wu and Lim, who originally ran the benchmark on 29 trackers in [13]. From those trackers, three — MIL, TLD and Boosting (OAB in original benchmark) — are implemented in OpenCV and the other two provide completely new benchmarks on the full data, although the authors of KCF have run their implementation of the algorithm on a 50 sequence subset of the dataset [8].

**Description of material and experiment** A Raspberry Pi 3 B v1.2 was used as the hardware. The board has a 64 bit CPU with 4 cores clocked at 1.2 GHz, and 1 GB RAM memory, with a default of 100 MB of virtual memory. The RAM is shared with the GPU, leaving 862 MB for general use by the operating system. The board was installed with a Raspbian GUI-based OS. This was necessary as the board was intended to be used in various tasks and it also helped with debugging the tracking programs by allowing the user to see the input and output of the software.

OpenCV was compiled without OpenMP [3] installed on the Raspberry Pi and as such the algorithms were tested on single core executions.

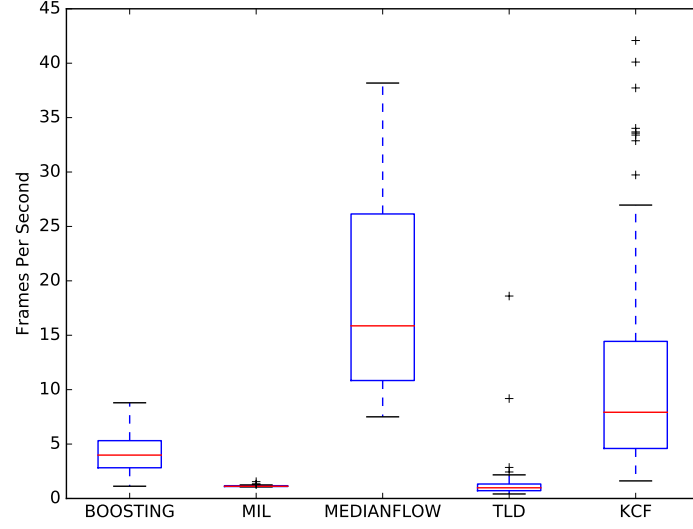
**Evaluation of performance** The performance of the algorithms was determined by measuring the processor time spent in the methods that update the bounding box of the tracker. The time spent initializing the tracker was not measured. The number of frames in a sequence was divided by the total time spent on all frames, producing a measure for frames per second, FPS.

**Evaluation of accuracy with the Jaccard index of similarity** The Jaccard index is a common measure of similarity between two sets  $A$  and  $B$ . This index is expressed as follows:

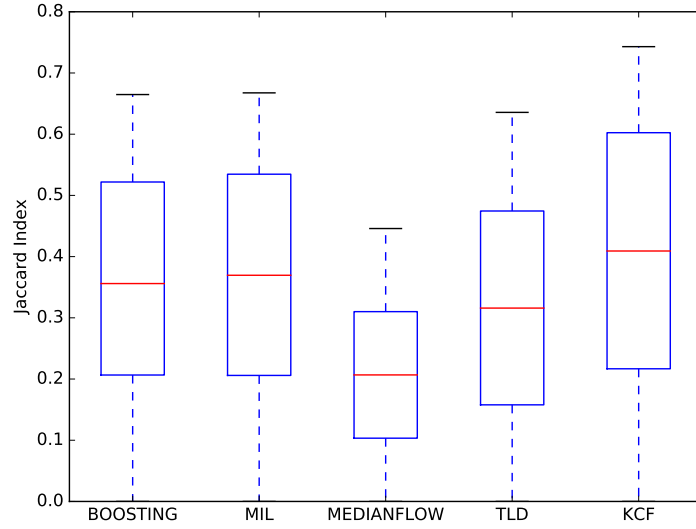
$$J(A, B) = \frac{A \cap B}{A \cup B} \quad (1)$$

When following an object, the Jaccard index presents a good measure of accuracy when comparing bounded boxes of the object followed drawn by the tested algorithm with bounded boxes of reference. When considering  $n$  frames of a video,  $R$  as the set of ground-truth bounding boxes in each frame and  $A$  the set of bounding boxes drawn by the tested algorithm, we derive a global similarity between  $A$  and  $R$  from equation 1 as follows:

$$J_R(A) = \frac{\sum_{i=1}^n J(A_i, R_i)}{n} \quad (2)$$



**Fig. 1.** Box plot illustrating averages and distributions of frame rates. Four (4) outliers above 50 FPS between Median Flow and KCF are missing to make the y axis scale better.



**Fig. 2.** Box plot illustrating averages and distributions of Jaccard indices.

## 4 Results

This section presents a visualization of results obtained from the benchmark. The code for the benchmark and the table of results are available from github<sup>1</sup>.

The tests showed that from the implemented trackers only Median Flow and TLD were able to adapt to scale changes in the tracked object, that is to the change of the size and shape of the detected bounding box.

Figure 1 illustrates the typical performance of the trackers in terms of processed frames per second. Median Flow is typically the fastest and KCF the second fastest, although KCF reaches very high performance on a few sequences. The Boosting algorithm sits between the fastest and slowest, and MIL and TLD are the slowest ones.

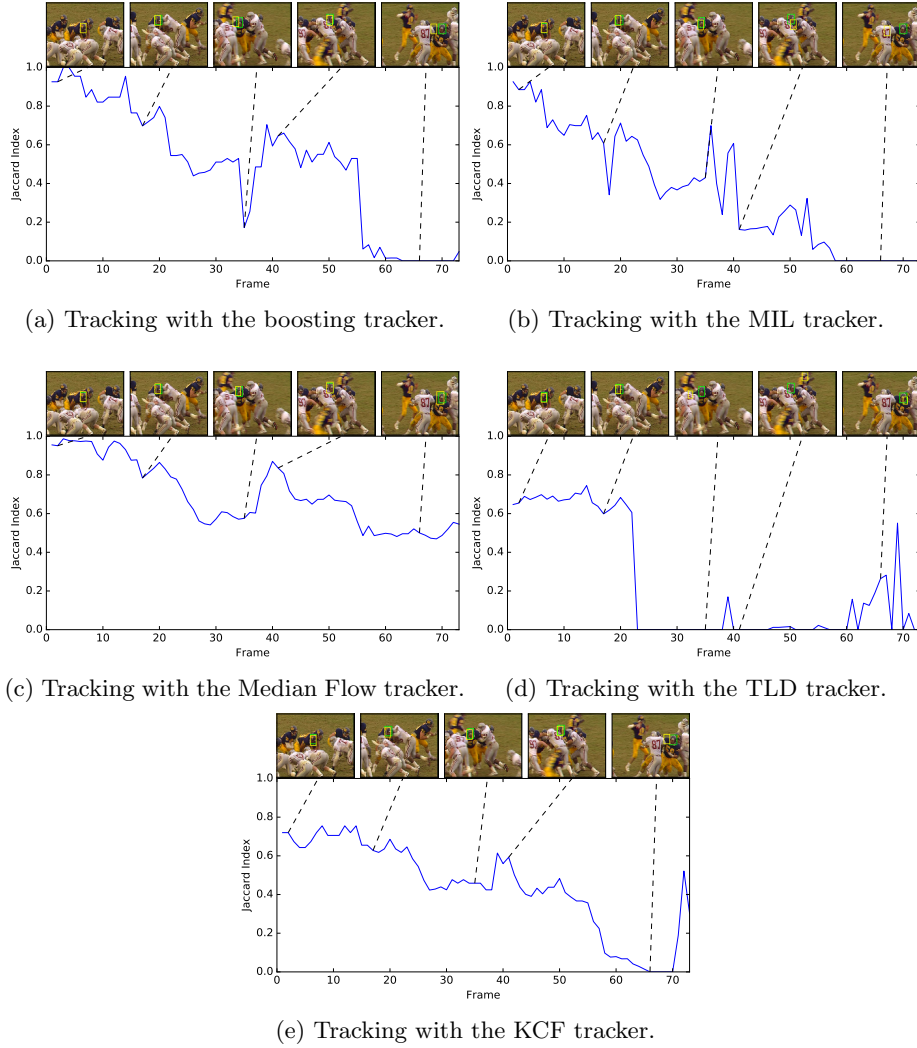
Only Median Flow and KCF trackers are able to perform at speeds that could be considered as real-time. The performance of any of these trackers in a practical application can be increased by reducing the resolution of the video feed, so the result can also be interpreted such that Median Flow and KCF allow for the largest resolution while maintaining real-time performance.

Figure 2 compares the statistics of the trackers with respect to the measured Jaccard indices. The Median Flow algorithm has a notably low accuracy. The rest share similar average accuracies, but KCF has the best average and can reach better accuracy than the others in best cases.

Figures 3a to 3e show the Jaccard index as a function of video frame for the sequence 'Football1'. The sequence shows players clashing in a game of American football, with the tracking target as one of the players faces. The images above the charts show selected frames from the sequence with the same frames shown for every tracker. The green rectangle in the images is the ground truth and the yellow rectangle is the tracking result.

Most of the trackers yield good results on this sequence and surpass their average performance. The boosting tracker (Fig. 3a) comes close to losing the target around the middle of the sequence, but is able to recover before finally losing the target at the end. The MIL tracker (Fig. 3b) shows a gradual degradation of tracking accuracy with some sudden jumps and finally loses the target. Despite being the lowest accuracy tracker, the Median Flow tracker (Fig. 3c) performs exceptionally well on this sequence; in fact better than any of the others. The low overall accuracy of this tracker seems to be caused by its tendency to completely lose its target and when this occurs, it moves its bounding box to the upper left corner of the frame and resizes it to zero, giving it no chance to recover. It seems to be a powerful tracker, if only it could recover from losing its target by e.g. re-detection. The TLD tracker (Fig. 3d) performs somewhat poorly on the sequence. The tracking bounding box jumps around erratically in the middle part of the sequence implying that this implementation of the tracker is relying mostly on tracking by detection rather than moving the bounding box in small increments. The KCF tracker (Fig. 3e) shows a mostly reliable tracking

<sup>1</sup> <https://github.com/lehtolav/tracker-benchmark>



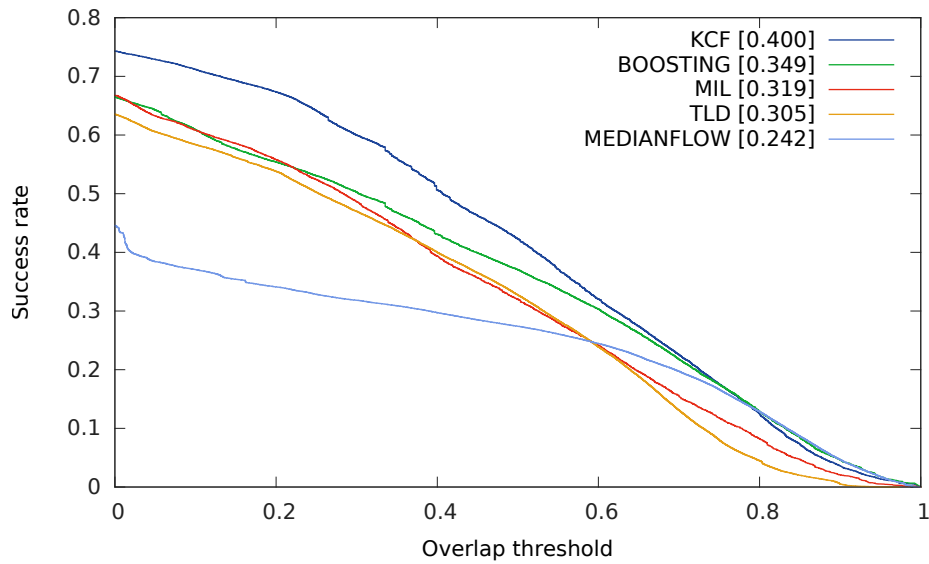
**Fig. 3.** Examples of the evolution of Jaccard index with the 5 implementations on the *football11* video stream

accuracy with some degradation over time until in the end it finally loses the target and suddenly finds it again.

The success rate is represented as a function of overlap threshold in Figure 4 similarly to the benchmark in [13]. The overlap threshold is the minimum Jaccard index required per frame. The success rate is the amount of frames that reached more than the threshold for all sequences. For example, 51% of the frames tracked with KCF have a Jaccard index higher than 0.4. The number in square brackets



is the area under curve (AUC) of the tracker. We see that the MIL and TLD implementations in OpenCV are inferior to those used in the benchmark, with MIL scoring 0.319 versus 0.362 and TLD scoring 0.305 versus 0.437. Notably, even the best of the OpenCV trackers, KCF, falls short of the performance of the TLD tracker of the benchmark, although TLD was only the third best scoring method there, with Struck scoring 0.473 and SCM scoring 0.499. While running the benchmarks, it became clear that the TLD implementation does not allow the object detection to fail, but forced it to detect the most likely candidate from the frame. This led to many false positives that confused the method.



**Fig. 4.** Success plots of the five tested trackers showing the fraction of frames successfully tracked as a function of required Jaccard index.

## 5 Conclusion

In this paper, we presented an evaluation of 5 tracker implementations. These 5 trackers are Median Flow, MIL, On-line Boosting, TLD, and KCF all of which are available from the contributions branch of the OpenCV library. This evaluation is based on two performance criteria. The first criterion is related to the rapidity of execution of implementations when embedded on a Raspberry Pi. The second criterion evaluates the accuracy in tracking with an averaged Jaccard index. According to this criterion, KCF scores highest. This quick evaluation method by averaged Jaccard indices (Fig. 2) gives the same ranking as the area under

curve (Fig. 4) so it can be used as a fast approximation for selecting a tracking implementation.

It appears that Median Flow presents a high frame rate and is the fastest of the 5 implementations. Despite this rapidity of execution, it shows a low accuracy scoring the lowest averaged Jaccard index when considering the mean accuracy score on all the video sequences. In fact, this method has often high tracking scores but is not able to detect the target again once lost. Therefore, KCF presents in our opinion a better compromise for visual tracking on embedded devices as it has a more robust tracking accuracy while keeping the second highest frame rate.

The trackers were benchmarked using only one core. The parallelized versions of these trackers could lead up to four times speedup on the used hardware (4 cores). This acceleration on TLD and MIL would still not beat the sequential performance of KCF and Median Flow.

The GOTURN Tracker [7] was recently introduced to the library. The authors results show the tracker running at 165 FPS on a GPU, but only 2.7 FPS on CPU. Thus it is very unlikely to be a candidate for use on low performance hardware. Moreover the tracker runs out of memory during initialization on the Raspberry Pi. Adding it to this benchmark in future would require adjusting the memory allocated for the GPU or adding extra virtual memory. For future work it would be interesting to benchmark the trackers on video from the Raspberry Pi camera either on a live feed or a pre-recorded video. Also, as an addition to the results in this paper it would be possible to measure the accuracy and performance of the trackers as a function of video resolution by using several downsized versions of the sequences.

## Acknowledgement

This research is funded by the Academy of Finland under project named “Bio-integrated Software Development for Adaptive Sensor Networks”, project number 278882. The work presented in this article is part of Ville Lehtola’s Master thesis. The authors also thank Jani Boutellier for his valuable comments to this article.

## References

1. Babenko, B., Belongie, S.: Visual tracking with online Multiple Instance Learning. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition. pp. 983–990 (2009)
2. Bradski, G.: The OpenCV Library. Dr. Dobb’s Journal of Software Tools (2000)
3. Dagum, L., Menon, R.: OpenMP: an industry standard API for shared-memory programming. IEEE Computational Science and Engineering 5(1), 46–55 (1998)
4. Grabner, H., Bischof, H.: On-line boosting and vision. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. vol. 1, pp. 260–267 (2006)

5. Grabner, H., Grabner, M., Bischof, H.: Real-Time Tracking via On-line Boosting. In: Proceedings of the British Machine Vision Conference. vol. 1, pp. 1–10 (2006)
6. He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. Arxiv.Org 7(3), 171–180 (2015), <http://arxiv.org/pdf/1512.03385v1.pdf>
7. Held, D., Thrun, S., Savarese, S.: Learning to track at 100 FPS with deep regression networks. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 9905 LNCS, pp. 749–765 (2016)
8. Henriques, J.F., Caseiro, R., Martins, P., Batista, J.: High-speed tracking with kernelized correlation filters. IEEE Transactions on Pattern Analysis and Machine Intelligence 37(3), 583–596 (2015)
9. Kalal, Z., Mikolajczyk, K., Matas, J.: Forward-backward error: Automatic detection of tracking failures. In: Proceedings - International Conference on Pattern Recognition. pp. 2756–2759 (2010)
10. Kalal, Z., Mikolajczyk, K., Matas, J.: Tracking-Learning-Detection. IEEE transactions on pattern analysis and machine intelligence 34(7), 1409–1422 (2012)
11. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You Only Look Once: Unified, Real-Time Object Detection. In: CVPR 2016. pp. 779–788 (2016)
12. Smeulders, A.W.M., Chu, D.M., Cucchiara, R., Calderara, S., Dehghan, A., Shah, M.: Visual tracking: An experimental survey. IEEE Transactions on Pattern Analysis and Machine Intelligence 36(7), 1442–1468 (2014)
13. Wu, Y., Lim, J., Yang, M.H.: Online object tracking: A benchmark. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. pp. 2411–2418 (2013)
14. Zhang, C., Platt, J.C., Viola, P.a.: Multiple instance boosting for object detection. In: Neural Information Processing Systems. vol. 74, pp. 1769–1775 (2005), <http://papers.nips.cc/paper/2926-multiple-instance-boosting-for-object-detection.pdf>