PROST: Parallel Robust Online Simple Tracking*

Jakob Santner Christian Leistner Amir Saffari Thomas Pock Horst Bischof
Institute for Computer Graphics and Vision, Graz University of Technology
{santner,leistner,saffari,pock,bischof}@icg.tugraz.at

Abstract

Tracking-by-detection is increasingly popular in order to tackle the visual tracking problem. Existing adaptive methods suffer from the drifting problem, since they rely on self-updates of an on-line learning method. In contrast to previous work that tackled this problem by employing semi-supervised or multiple-instance learning, we show that augmenting an on-line learning method with complementary tracking approaches can lead to more stable results. In particular, we use a simple template model as a non-adaptive and thus stable component, a novel optical-flow-based mean-shift tracker as highly adaptive element and an on-line random forest as moderately adaptive appearance-based learner. We combine these three trackers in a cascade. All of our components run on GPUs or similar multi-core systems, which allows for real-time performance. We show the superiority of our system over current state-of-the-art tracking methods in several experiments on publicly available data.

1. Introduction

Visual object tracking is one of the cardinal problems of computer vision. Although tracking finds many practical applications ranging from robotics, surveillance, augmented reality to human-computer interaction, the state-of-the-art is still far from achieving results comparable to human performance. Trackers have to deal with several difficulties such as background clutter, fast appearance and illumination changes and occlusions. Many different tracking methods have been proposed, from global template-based trackers [2], shape-based methods, probabilistic models using mean-shift [7] or particle filtering [13] to local key-point based trackers [14] or flow-based trackers [19].

Recently, tracking methods based on detection systems have become very popular. These tracking-by-detection systems usually perform one-shot learning of object detectors for the target object at the first frame. Surrounding patches are taken as negative samples [2]. These systems are fast and yield good performance since the classification task is simple: Discriminate the target object from its surrounding background. In order to allow for fast appearance changes, recent works use online learners that perform updating on the target object based on the tracking result (self-updating) [9].

The problem with these approaches is, that the self-updating process may easily cause drifting in case of wrong updates. Even worse, the tracking-by-detection approach suffers also from the fact that usually online counterparts of supervised learning algorithms are used, which are not designed for handling ambiguity of class labels: Despite the fact that boosting is known to by highly susceptible to label noise, it is widely used in self-learning based tracking methods. This severe problem of adaptive tracking-by-detection methods can also be explained by the stability-plasticity dilemma [11]: If the classifier is trained only with the first frame, it is less error-prone to occlusions and can virtually not drift. However, it is not adaptive at all and cannot follow an object undergoing appearance and viewpoint changes. On the other hand, online classifiers that perform self-learning on their own confidence maps are highly adaptive but easily drift in the case of wrong updates.

Grabner *et al*. [10] alleviated this dilemma by formulating tracking-by-detection as a one-shot semi-supervised learning problem using online boosting. Supervised updates are only performed at the first frame and all subsequent patches are exploited as unlabeled data with the help of a non-adaptive prior classifier. Although this method has shown to be less susceptible to drifting and simultaneously more adaptive than an offline learner, it turned out that such an approach is still not adaptive enough to handle fast appearance changes [3].

Another cause of drifting for online learners that perform self-updating is label jitter. The problem of label jitter arises if the bounding boxes of an object are not perfectly aligned with the target, although it is detected correctly. If label jit-

*This work was supported by the Austrian Research Promotion Agency (FFG) within the projects VM-GPU (813396) and Outlier (820923) as well as the Austrian Science Fund (FWF) under the doctoral program Confluence of Vision and Graphics W1209. We also greatly acknowledge Nvidia for their valuable support.

978-1-4244-6985-7/10/$26.00 ©2010 IEEE

ter occurs repeatedly over a tracking sequence, the tracker will most likely start to loose the target object. For offline detectors, Viola *et al*. [22] showed that multiple-instance learning can easily handle such ambiguities of bounding boxes. Therefore, Babenko *et al*. [3] recently used online multiple instance learning to reduce the effect of label jitter during tracking. In their work, the main idea is to take patches lying most likely on the target object as instance for a positive bag and instances further away as negatives. This approach currently yields the best tracking-by-detection results and can be considered as the state-of-the-art.

In this paper, we revisit the stability-plasticity dilemma [11] of online tracking-by-detection methods. In contrast to recent works (*e.g.* [10, 3]), we do not tackle the problem by applying another learning method, but by combining several complimentary trackers operating at different timescales. Recently, Stenger *et al*. [21] investigated in different combinations of tracking methods. Given a particular tracking scenario, they tried to learn which methods are useful and how they can be combined to yield good results. Our work differs from their approach such that our method does not require offline pre-training of possible combinations.

We show that augmenting a simple online learner with its two extreme counterparts in terms of adaptivity can lead to much better results. In particular, our approach is based on the fact, that different tracking approaches lie on different scales of the adaptivity spectrum (Figure 1): On the one very end are trackers, that are totally non-adaptive such as template-based trackers. On the other end are highly adaptive methods such as optical-flow-based trackers. Tracking-by-detection systems are somewhere in between, depending on their learning method and adaptivity rate.

We propose a system called PROST[1] (Parallel Robust Online Simple Tracking), consisting of three different trackers that are able to cover the entire adaptivity spectrum. We use basic normalized cross correlation template matching to cover the non-adaptive end. Additionally, we introduce a novel highly adaptive optical-flow-based mean-shift tracker. In between, our system consists of an online random forest [17] as adaptive appearance-based classifier. In contrast to previous methods, our system is especially designed to alleviate the drifting problem of appearance based trackers while still being highly adaptive. The core parts have been selected to be easily parallelized and are implemented on the GPU in order to allow for real-time performance.

The adaptivity rate of online trackers can be adjusted by parameter tuning in order to fit to a specific dataset. A particular advantage of our system is, that it is able to perform well on unseen sequences without the need of being adjusted beforehand. Throughout all experiments in this paper, no parameter adjustment has been done - *all results use the identical algorithm and settings*.

[1] *prost* is the german word for *cheers*

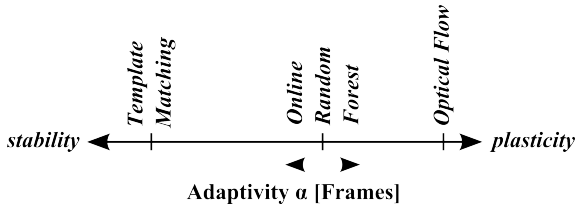


Figure 1. The response characteristics of an online tracker can be defined as the number of frames it needs to adapt to appearance changes. Our complementary algorithms have been chosen from the opposite ends of this spectrum.

In Section 2 and 3, we present our approach and give a detailed overview of its individual parts. In Section 4, we compare our approach to other state-of-the-art methods on benchmark tracking data sets and on own recorded sequences. Finally, in Section 5, we give some conclusions and ideas for future work.

2. Tracking Components

Our goal is to allow an online-tracker to be adaptive to fast appearance changes without being too prone to drifting. In other words, we would like to increase its stability and plasticity at the same time. Therefore, we add complementary algorithms with different adaptivity rate α, where α denotes the number of frames a tracker needs to fully adapt to appearance changes.

We make the following observations: (i) Object detection algorithms do not adapt to appearance changes, yielding an infinite adaptivity rate $\alpha = \infty$. (ii) Frame-to-frame trackers adapt to changing object appearance at every frame thus having $\alpha = 1$. (iii) Online trackers usually can be adjusted to have a certain adaptivity rate $1 \leq \alpha \leq \infty$.

In order to study the most complementary algorithms for an online system, we selected one tracking method from each far end of the adaptivity graph (see Figure 1): An optical flow based tracker and a simple template matching method. All parts are described in more detail below. Note that there are more sophisticated methods performing better than the chosen ones. However, the goal is to demonstrate that an online tracker can be substantially improved by smartly combining it with even simple methods.

2.1. Template Correlation

The static part in our system is based on normalized cross-correlation (*NCC*). We simply use the object which is marked in the first frame by a rectangle as template and match it in every forthcoming frame. The tracking rectangle is moved to the peak of the correlation confidence map. *NCC* does not adapt to any changes but brightness, which renders it useless when the object appearance changes permanently.

2.2. Mean Shift Optical Flow

The estimation of optical flow is one of the essential problems in low-level computer vision. It basically deals with computing the motion between consecutive frames of a sequence. In [12], Horn and Schunk estimated optical flow by minimizing an energy functional of the form

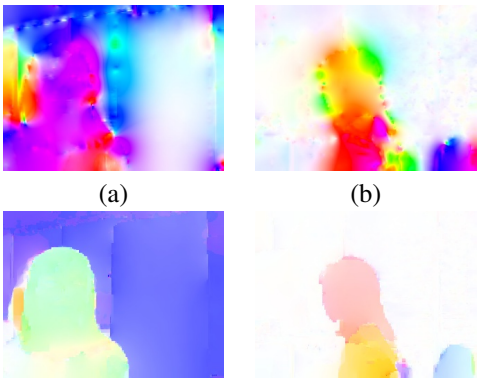
$$\min_u \Big\{ \int_\Omega |\nabla u_1|^2 + |\nabla u_2|^2 \, d\Omega + \lambda \int_\Omega \left(I_1(x + \mathbf{u}(x)) - I_0(x)\right)^2 d\Omega \Big\} \quad (1)$$

with $\mathbf{u} = (u_1, u_2)^T$, consisting of two terms: A regularization term $\int_\Omega |\nabla u_1|^2 + |\nabla u_2|^2$ smoothing the flow field and a data term $\int_\Omega \left(I_1(x + \mathbf{u}(x)) - I_0(x)\right)^2$. λ is a parameter steering between data term and regularization term, I_0 and I_1 represent the sequential frames and $\mathbf{u}$ is the two-dimensional flow field. This model uses quadratic penalizers and therefore is not suited for estimating flow fields with sharp discontinuities. Using an L_1 norm on the data term and Total Variation (TV) regularization [20] leads to the following energy:

$$\min_u \Big\{ \int_\Omega |\nabla u_1| + |\nabla u_2| \, d\Omega + \lambda \int_\Omega \left|I_1(x + \mathbf{u}(x)) - I_0(x)\right| d\Omega \Big\} \quad (2)$$

Zach *et al*. [24] achieved realtime performance by minimizing this energy on the GPU. The TV regularization favors sharp discontinuities, but also leads to a so-called staircase effect, where the flow field exhibits piecewise constant levels. In recent work, Werlberger *et al*. [23] replaced the TV norm by a Huber norm to tackle this problem: Below a certain threshold the penalty is quadratic, leading to smooth flow fields for small displacements. Above that threshold, the penalty becomes linear allowing for sharp discontinuities. With the additional incorporation of a diffusion tensor for anisotropic regularization, their method (Huber - L_1) is currently one of the most accurate optical flow algorithms according to the Middlebury evaluation website [4]. Figure 2 shows the difference between the method of Werlberger *et al*. and the algorithm of Horn and Schunk.

In order to use the dense flow field as input to a tracker, we estimate the object's translation from the flow vectors. We use a mean-shift procedure in the two-dimensional translation space, taking into account every flow-vector within our tracking rectangle. In contrast to averaging the displacement vectors, mean shift allows to handle occlusions more robustly. For simplicity, we estimate only translation of our object throughout this work; however, note that other motion models incorporating e.g., rotation, scale, affine motion, etc. could be estimated from the flow field.



(a) (b) (c) (d)

Figure 2. Optical flow estimation using the algorithm of Horn and Schunk [12] (a,b) and Werlberger *et al*. [23] (c,d). These flow-fields are printed in color-coded representation, where hue encodes direction and intensity encodes magnitude of the flow vectors. As can easily be seen, (c) and (d) are less noisy than (a) and (b) while preserving sharp motion boundaries.

This mean shift optical flow tracker (*FLOW*) is fast and accurately adapts to appearance changes. However, it may loose the object in presence of large occlusions, fast illumination changes and vast 3D-motion such as out-of-plane rotations. Furthermore, once it fails, it is not able to recover.

2.3. Online Random Forest

Complementary to *FLOW* and *NCC*, we employ an adaptive appearance-based tracker based on online random forests (*ORF*). Random Forests [6] are ensembles of N recursively trained decision trees in form of $f(\mathbf{x}) : \mathcal{X} \rightarrow \mathcal{Y}$. For a forest $\mathcal{F} = \{f_1, \cdots, f_N\}$, a decision is made by simply taking the maximum over all individual probabilities of the trees for a class k with $C(x) = \arg\max_{k \in \mathcal{Y}} \frac{1}{N} \sum_{n=1}^{N} p_n(k|\mathbf{x})$, where $p_n(k|\mathbf{x})$ is the estimated density of class labels of the leaf of the n^{th} tree. In order to decrease the correlation of the trees, each tree is provided with a slightly different subset of training data by sub sampling with replacement from the entire training set, *a.k.a* bagging. During training, each split node randomly selects binary tests from the feature vector and selects the best according to an impurity measurement. The information gain after node splitting is usually measured with

$$\Delta H = -\frac{|I_l|}{|I_l| + |I_r|} H(I_l) - \frac{|I_r|}{|I_l| + |I_r|} H(I_r), \quad (3)$$

where I_l and I_r are the left and right subsets of the training data. $H(I) = -\sum_{i=1}^{K} p_i^j log(p_i^j)$ is the entropy of the classes in the node and p_i^j is the label density of class i in node j. The recursive training continues until a maximum depth is reached or no further information gain is possible.

Random Forests have several advantages that make them particularly interesting for computer vision applications, *i.e.*, they are fast in both training and evaluation and yield state-of-the-art classification results while being less noise-sensitive compared to other classifiers (*e.g.*, boosting). Additionally, RFs are inherently multi-class and allow, due to their parallel structure, for multi-core and GPU [18] implementations.

Recently, Saffari *et al*. [17] proposed an online version of RFs which allows to use them as online classifiers in tracking-by-detection systems. Since recursive training of decision trees is hard to do in online learning, they propose a tree-growing procedure similar to evolving-trees [15]. The algorithm starts with trees consisting only of root nodes and randomly selected node tests f_i and thresholds θ_i. Each node estimates an impurity measure based on the Gini index ($G_i = \sum_{i=1}^{K} p_i^j(1-p_i^j)$) online, where p_i^j is the label density of class i in node K. Then, after each online update the possible information gain ΔG during a potential node split is measured. If ΔG exceeds a given threshold β, the node becomes a split node, *i.e.*, is not updated any more and generates two child leaf nodes. The growing proceeds until a maximum depth is reached. Even when the tree has grown to its full size, all leaf nodes are further updated online.

The method is simple to implement and has shown to converge fast to its offline counterpart. Additionally, Saffari *et al*. [17] showed that the classifier is faster and more noise-robust compared to boosting, which makes it an ideal candidate for our tracking system.

3. Tracker Combination

A tracker has to incorporate two conflicting properties: It has to (i) adapt to fast object appearance changes while (ii) being able to recover in case of drifting. In other words, we need an highly adaptive tracker that is corrected by system components that are more inertial. Therefore, we combine the three different tracking approaches discussed before in a simple fall-back cascade (see also Figure 3): In order to allow for fast changes, *FLOW* forms the main tracker. This implies that *FLOW* can also easily lose the target, hence, it can be overruled by *ORF*. *NCC* is employed to prevent *ORF* from making too many wrong updates. Our cascade can be summarized with the following simple rules:

1. *FLOW* is overruled by *ORF* if they are (i) not overlapping and (ii) *ORF* has a confidence above a given threshold.
2. *ORF* is updated only if it overlaps with *NCC* or *FLOW*.

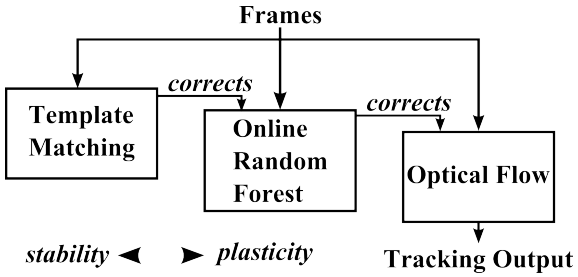


Figure 3. Highly-flexible parts of our system take care of tracking, while the conservative parts correct the flexible ones when they have drifted away.

4. Experiments

During the experiments, we compare our algorithm to current state of the art methods on publicly available datasets. We also created several new challenging video sequences, which are available on our website together with ground truth annotation and results [2]. The major conclusion from the experiments is that our algorithm is more adaptive and stable at the same time compared to other tracking-by-detection systems. Please note that we *always use the same parameters* throughout the experiments in this section.

4.1. Implementation

For *FLOW*, we employ the GPU-based implementation of Werlberger *et al*. [23], which is available online. *NCC* is based on the *cvMatchTemplate()* function implemented in the *OpenCV* library, *ORF* is based on the code of Saffari *et al*. [17], which is also publicly available. We achieve real-time performance with our system, however, *NCC* and especially *ORF* could benefit largely from being implemented on the GPU.

4.2. Quality Score

To evaluate the performance of their tracker, Babenko *et al*. [3] use a score representing the mean center location error in pixels. This is not a good choice, as the ground truth rectangles are fixed in size and axis-aligned whereas the sequences exhibit scale and rotational changes. Furthermore, their score does not take into account the different size of the objects in different sequences.

To overcome these problems, we additionally use a score based on the PASCAL challenge [8] object detection score: Given the detected bounding box ROI_D and the ground truth bounding box ROI_{GT}, the overlap score evaluates as

$$score = \frac{area(ROI_D \cap ROI_{GT})}{area(ROI_D \cup ROI_{GT})}.$$

[2] www.gpu4vision.org

By interpreting a frame as true positive when this score exceeds 0.5, we can give a percentage of correctly tracked frames for each sequence.

4.3. Sequences

Throughout the experiments, we use ten challenging sequences (Table 1) featuring e.g. moving cameras, cluttered background, occlusions, 3-D motion or illumination changes. The video data, ground truth and results of other methods for the first six sequences have been taken from Babenko *et al*. [3]. The other four videos (see Figure 7) have been created and annotated by ourselves.

Sequence	Frames	Main Challenges
Girl [5]	502	3D-motion, moving camera
David [16]	462	moving camera, varying illumination
Sylvester[16]	1344	3D-motion, varying illumination
Faceocc1 [1]	886	moving camera, occlusions
Faceocc2 [3]	812	occlusions, heavy appearance change
Tiger1 [3]	354	fast motion, heavy appearance change
Board	698	3D-motion
Box	1161	fast 3D-motion, occlusions
Lemming	1336	heavy scale changes, motion blur
Liquor	1741	motion blur, occlusions

Table 1. The tracking sequences used in our experiments. The last four videos are available together with ground-truth annotations on our website.

4.4. Performance of the building blocks

In the first experiment, we investigate the behavior of our three building blocks separately on two sequences, *Sylvester* and *David*. The average pixel error is given in Figure 4.

- *NCC* works well when the appearance of the object is close to the learned template. In the sequence *David*, this holds for the first 100 frames, then the object has changed such that *NCC* is not able to distinguish it from background. For *Sylvester*, the *NCC* works also well on the initial frames and, although loosing the object later, it is able to find it again several times throughout the sequence.

- *ORF* clearly indicates the fundamental problem of online tracking algorithms on these two sequences: With identical parameters, it is stable enough for *Sylvester* but looses *David* completely after 150 frames.

- *FLOW* tracks the object correctly for the first 150 frames on *David* and 400 frames of *Sylvester*, but then starts to drift away, accumulating errors from frame to frame. In *David*, it gets back to the object by chance around frame 200, but then looses it again at frame 400. In general, the more frames tracked, the less accurate *FLOW* gets.

With these experiments we show, that the different algorithms can complement one another. *FLOW* is a good high dynamic tracker, but needs correction from time to time to get rid of cumulating errors. *ORF* could do that, but needs a supervisor preventing it from doing too many wrong updates. *NCC* is not suited to track on a per-frame basis but gives strong cues when the object reappears similarly to the initial template.

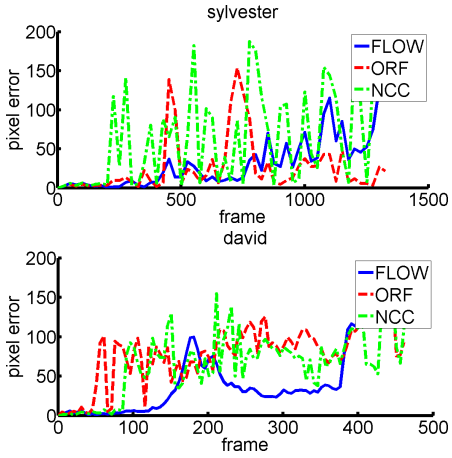


Figure 4. Separate evaluation of the building blocks of our system.

4.5. Benchmarks

4.5.1 Standard Datasets

In this experiment, we would like to benchmark our approach on the following sequences: *Girl*, *David Indoor*, *Sylvester*, *Occluded Face*, *Occluded Face 2* and *Tiger 1*. Recently, Babenko *et al*. [3] showed superior results comparing their method (*MILTrack*) to *AdaBoost* of Grabner *et al*. [9] and *FragTrack* of Adam *et al*. [1]. We benchmark against their results, details on the parametrization of the different algorithms are given in their paper. For their own method, they provide results for five different runs: As their algorithm depends on random features, the performance varies between subsequent runs. This difference is most of the times substantial, however, we give our own algorithm a handicap by *comparing our results to their best run* in each sequence according to the PASCAL score.

Table 2 and Figure 5 depict the results based on the mean pixel error: In 3 of 6 sequences, our method yields the best scores, in the other sequences it is the second best. Table 3 depicts the percentage of frames tracked correctly over all six sequences based on the PASCAL score: Our algorithm's correct frames average to 83.8% followed by *MILTrack* (80.0%), *FragTrack* (59.8%) and *AdaBoost* (41.0%).

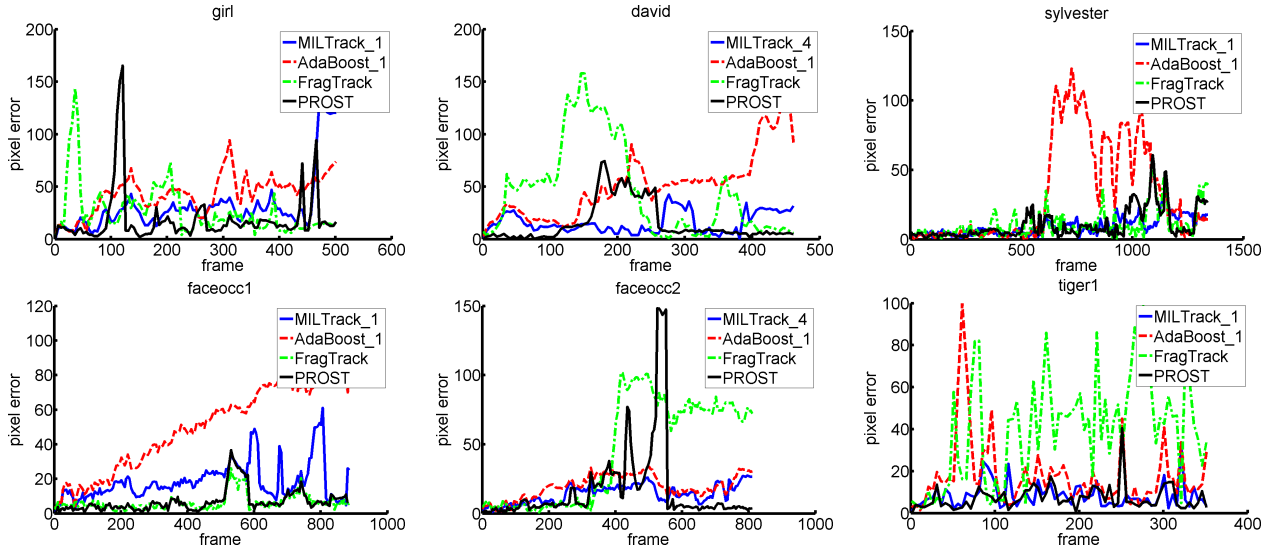


Figure 5. Tracking results for standard tracking sequences

Sequence	*Adaboost*	*FragTrack*	*MILTrack*	*PROST*
Girl	43.3	26.5	31.6	**19.0**
David	51.0	46.0	15.6	**15.3**
Sylvester	32.9	11.2	**9.4**	10.6
Faceocc	49.0	**6.5**	18.4	7.0
Faceocc2	19.6	45.1	**14.3**	17.2
Tiger1	17.8	39.6	8.4	**7.2**

Table 2. Mean distance of the tracking rectangle to annotated ground truth, the best result is printed in bold faced letters, the second best result is underlined.

Sequence	*Adaboost*	*FragTrack*	*MILTrack*	*PROST*
Girl	24	70	70	**89**
David	23	47	70	**80**
Sylvester	51	**74**	**74**	73
Faceocc	35	**100**	93	**100**
Faceocc2	75	48	**96**	82
Tiger1	38	20	77	**79**

Table 3. Percentage of frames tracked correctly.

4.5.2 PROST dataset

To further demonstrate the capabilities of our system, we compare on newly created sequences. Besides our own method (parametrized identically to the previous experiments), we benchmark the following algorithms:

- *ORF* with 100 trees of maximum depth 5 and a search region factor of 1.0. Similar to *MILTrack* [3], we use Haar-like features. This is exactly the online part of our tracker, thus this experiment directly shows the benefit of the complementary methods.
- *FragTrack* [1] with 16 bins and a search window half size of 25 to cope with the larger frame size.
- *MILTrack* [3], as provided on their webpage with search window size increased to 50. Similar to the previous experiment, we use the best out of 5 differently initialized runs.

The average pixel error for each method is given in table 4, the PASCAL based score in table 5. Our approach yields the best score in three sequences, tracking correctly an average of 79.5% over all four sequences, followed by *FragTrack* (65.3%), *MILTrack* (48.5%) and *ORF* (27.3%). Looking at the pixel error graph in Figure 6 directly shows the benefits of our combined system over the online tracker *ORF* it is based on:

- *ORF* looses the object in every sequence after at least 400 frames. With the high-dynamic optical flow tracker increasing plasticity, our system looses the object far less often.
- When *ORF* has lost the track, it performs wrong updates until eventually totally drifting away from the object. This happens in the sequences *board*, *box* and *lemming*. In *liquor*, it is able to recover the object three times. Although far less often, our system also looses the track several times, but is, except for the last frames of *liquor*, always able to recover the object.

Sequence	*MILTrack*	*ORF*	*FragTrack*	*PROST*
Board	51.2	154.5	90.1	**37.0**
Box	104.6	145.4	57.4	**12.1**
Lemming	**14.9**	166.3	82.8	25.4
Liquor	165.1	67.3	30.7	**21.6**

Table 4. Mean distance error to the ground truth.

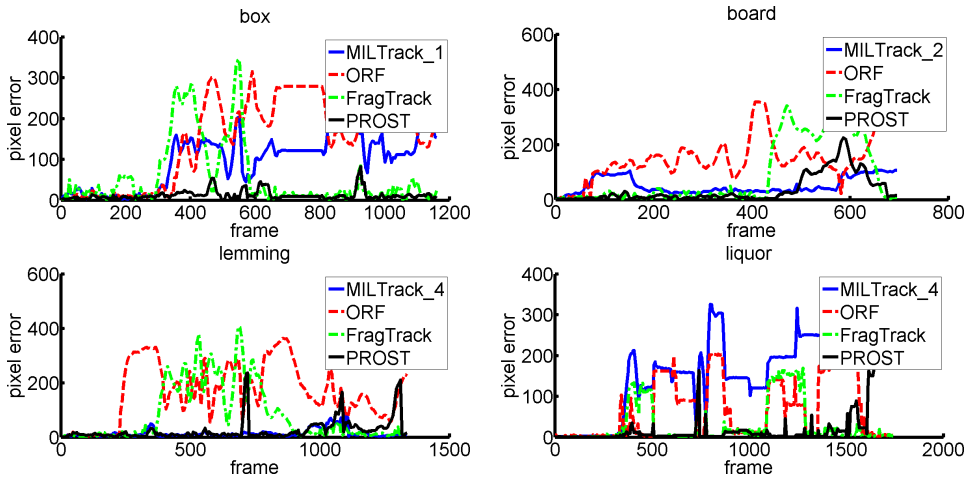


Figure 6. Tracking results for the PROST dataset

Sequence	*MILTrack*	*ORF*	*FragTrack*	*PROST*
Board	67.9	10.0	67.9	**75.0**
Box	24.5	28.3	61.4	**91.4**
Lemming	**83.6**	17.2	54.9	70.5
Liquor	20.6	53.6	79.9	**83.7**

Table 5. Percentage of frames tracked correctly.

5. Conclusion

In this paper, we addressed the robustness and adaptivity of on-line appearance-based tracking. In order to increase the stability and plasticity of an on-line tracker at the same time, we proposed to combine it with both a static and a highly dynamic element. In particular, we combined an on-line random forest with a simple correlation-based template tracker and a novel optical-flow-based mean shift tracker as most adaptive part. The three elements are combined in a cascade-style.

In the experimental part, we compared our method with state-of-the-art appearance-based methods on both tracking benchmark data sets and on own recorded sequences. We demonstrated superior performance in sequences that demand more conservative tracking behavior as well as sequences with rapid appearance changes with constant parameter settings.

Our approach suggests several extensions: First, we used simple methods in our combined tracker. As each individual part of our system can be exchanged easily, employing more powerful trackers could increase the performance of the overall system. Second, the tracker is currently restricted to axis-aligned fixed-size rectangles. One can increase the power of the system by extending it to handle rotation, scale change or affine motion and by giving pixel-wise segmentations of the object.

References

[1] A. Adam, E. Rivlin, and I. Shimshoni. Robust fragments-based tracking using the integral histogram. In *CVPR*, 2006. 5, 6

[2] S. Avidan. Ensemble tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(2):261–271, 2007. 1

[3] B. Babenko, M.-H. Yang, and S. Belongie. Visual Tracking with Online Multiple Instance Learning. In *CVPR*, 2009. 1, 2, 4, 5, 6

[4] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. Black, and R. Szeliski. A database and evaluation methodology for optical flow. In *ICCV*, 2007. http://vision.middlebury.edu/flow/. 3

[5] S. Birchfield. Elliptical head tracking using intensity gradients and color histograms. In *CVPR*, 1998. 5

[6] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001. 3

[7] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *CVPR*, 2000. 1

[8] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *Int. J. Comput. Vision*, 88(2):303–308, 2009. 4

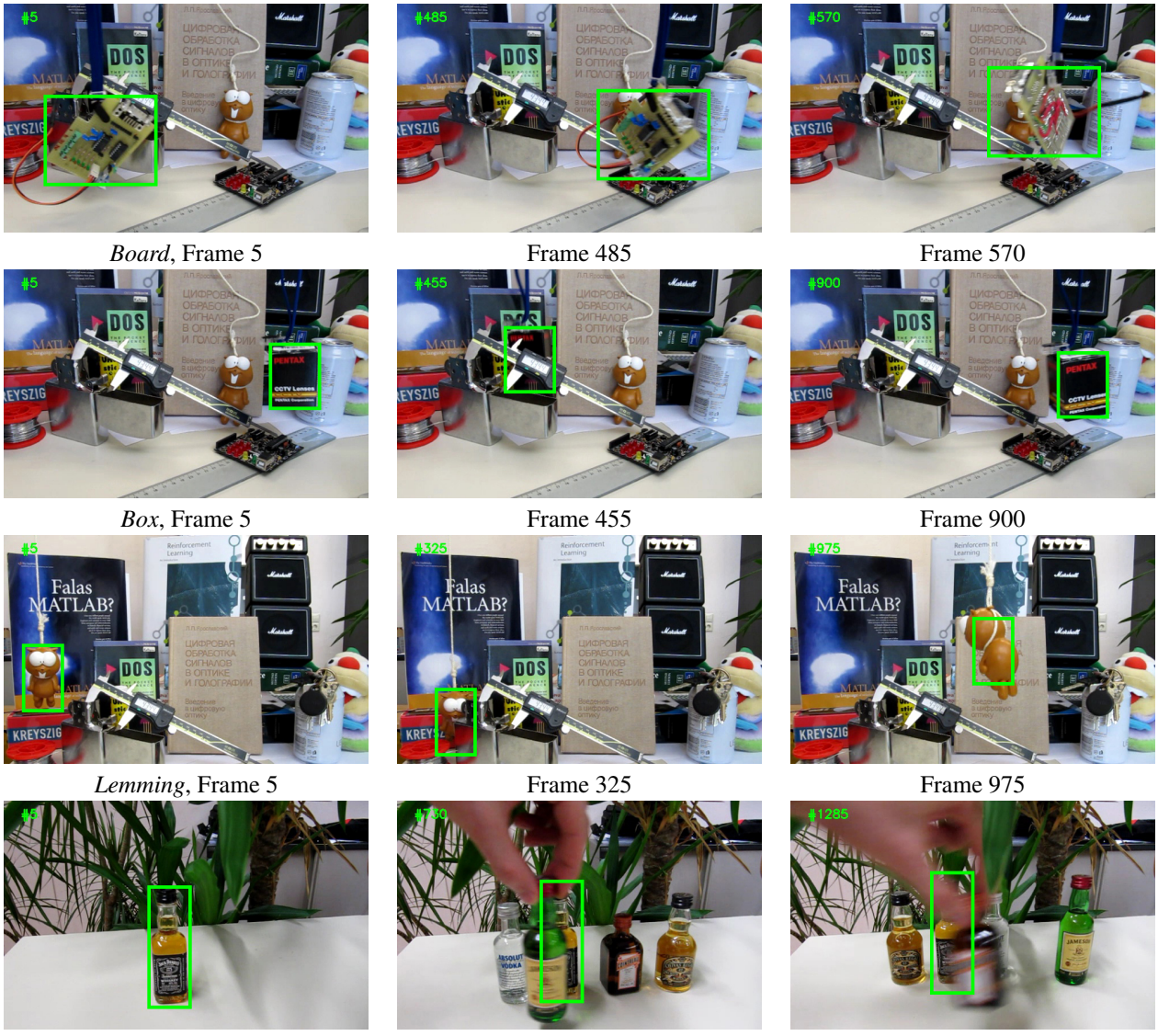
[9] H. Grabner, M. Grabner, and H. Bischof. Real-time tracking via on-line boosting. In *Proceedings British Machine Vision Conference*, 2006. 1, 5

[10] H. Grabner, C. Leistner, and H. Bischof. Semi-supervised on-line boosting for robust tracking. In *ECCV*, 2008. 1, 2

[11] S. Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Neural networks and natural intelligence*, pages 213–250, 1998. 1, 2

[12] B. K. P. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence, 17*, pages 185–203, 1981. 3

[13] Y. Li, H. Ai, T. Yamashita, S. Lao, and M. Kawade. Tracking in low frame rate video: A cascade particle filter with discriminative observers of different lifespans. In *CVPR*, 2007. 1



Board, Frame 5 | Frame 485 | Frame 570

Box, Frame 5 | Frame 455 | Frame 900

Lemming, Frame 5 | Frame 325 | Frame 975

Liquor, Frame 5 | Frame 730 | Frame 1285

Figure 7. Exemplar frames of the PROST dataset, the rectangle represents the ground truth.

[14] M. Özuysal, P. Fua, and V. Lepetit. Fast keypoint recognition in ten lines of code. In *CVPR*, 2007. 1

[15] J. Pakkanen, J. Iivarinen, and E. Oja. The evolving tree—a novel self-organizing network for data analysis. *Neural Process. Lett.*, 20(3):199–211, 2004. 4

[16] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang. Incremental learning for robust visual tracking. *Int. J. Comput. Vision*, 77(1-3):125–141, 2008. 5

[17] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof. On-line random forests. In *3rd IEEE ICCV Workshop on On-line Comput. Vision*, 2009. 2, 4

[18] T. Sharp. Implementing decision trees and forests on a GPU. In *ECCV*, 2008. 4

[19] J. Shi and C. Tomasi. Good features to track. In *CVPR*, 1994. 1

[20] D. Shulman and J.-Y. Hervé. Regularization of discontinuous flow fields. In *Proceedings Workshop on Visual Motion*, 1989. 3

[21] B. Stenger, T. Woodley, and R. Cipolla. Learning to track with multiple observers. In *CVPR*, 2009. 2

[22] P. Viola, J. Platt, and C. Zhang. Multiple instance boosting for object detection. In *Advances in Neural Information Processing Systems*, volume 18, pages 1417–1424. MIT Press, 2006. 2

[23] M. Werlberger, W. Trobin, T. Pock, A. Wedel, D. Cremers, and H. Bischof. Anisotropic Huber-L1 optical flow. In *Proc. of the British Machine Vision Conf.*, 2009. 3, 4

[24] C. Zach, T. Pock, and H. Bischof. A duality based approach for realtime tv-l1 optical flow. In *Pattern Recognition (Proc. DAGM)*, 2007. 3