TLM: Tracking-Learning-Matching of Keypoints

Georg Nebehay Roman Pflugfelder
AIT Austrian Institute of Technology
Safety and Security Department
Donau-City-Straße 1
1220 Vienna, Austria
{georg.nebehay.fl, roman.pflugfelder}@ait.ac.at

***Abstract*—In this work, we present an approach for improving the long-term association of keypoints in videos by exploiting the unlabeled parts of the video on-the-fly. Towards this end, we track keypoints from frame to frame by using a method for estimating optic flow. As long as this method is successful (according to certain criteria), we update an ensemble classifier with training data stemming from newly discovered views of the keypoint as well as from false matches. In each frame, we match candidate keypoints to the original keypoint by classifying them and re-initialise the tracking mechanism after failure. Neither a-priori knowledge about the keypoint nor a training stage is required. Our method avoids the use of an expensive sliding-window approach used by a similar method and instead embraces a highly efficient keypoint detection and matching stage, making our method suitable for the use in embedded devices. We show experimentally that our approach is able to provide both accurate and robust results on several sequences.**

I. Introduction

Local features are a fundamental building block for many applications in image and video processing. They are simple and intuitive to use as they can be interpreted as corresponding to interesting parts of objects [24]. While there are various different types of local features, in this work we focus on the robust matching of **keypoints**, arguably the simplest type of a local feature. They have been used in a variety of fields, such as object recognition and tracking, image retrieval and 3D-reconstruction [23].

There are two fundamentally different types of how corresponding keypoints in videos can be identified [23]:
(a) Keypoints in a video frame are **matched** to keypoints in a static database, typically by computing a distance between their **descriptors**. Seminal work in this field was done by Lowe [11]. This type of matching is **robust**, as every match is performed independently of the matching result in previous frames. This allows for instance to deal with occlusions of keypoints. However, the matching process becomes more difficult when elements of the scene change, for instance global or local illumination, camera position, or changes in the appearance of the keypoint itself.
(b) Keypoints are **tracked** from one video frame to the next, for instance by estimating its translation according to the optic flow. Since tracking is typically done by employing a local search, it allows for a greater degree of **adaptivity**, leading to more **accurate** results. These methods fail when the keypoint of interest disappears from the scene, for instance by an occlusion. Another problem is that these methods suffer from the danger of **drift**, which refers to the situation when a method loses track of its original target and adapts to a different target.

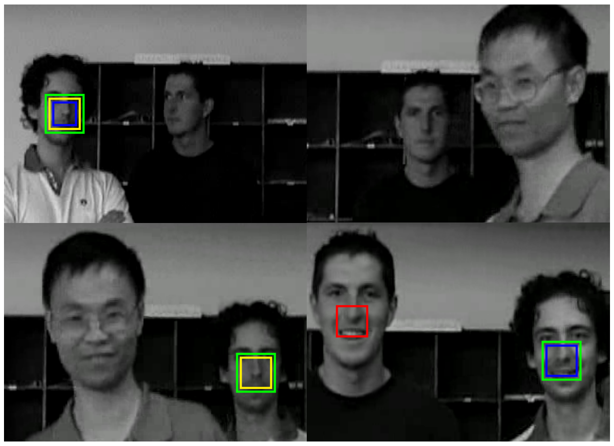


Fig. 1: The main idea of our algorithm. In the top left image, Both the robust matcher (yellow) and the adaptive tracker (blue) agree on the location of the keypoint. The combined result is shown in green. Subsequently, the tracker discovers new views of the keypoint that are used for updating the matcher, until in the top right image, an occlusion of the keypoint occurs, leading to the failure of both methods. In the bottom left image, only the matcher has a result, which is used to re-intialise the tracker. In the bottom right image, the system is able to discover a false match (red), and incorporates it into the matching process. The underlying sequence is from [13].

In this work, we propose a novel way of combining a method for **tracking** keypoints between consecutive frames and a robust **matching** mechanism for keypoints. Keypoint matching is improved in each frame in a **learning** framework. We build on ideas from the recently proposed method for object tracking Tracking-Learning-Detection (TLD [8]). Fig. 1 illustrates the basic idea of our approach. When developing this algorithm, we explicitly envisioned its application on embedded devices such as smart cameras that have to cope with limited resources with respect to computational power.

The contributions of this work are threefold: (1) the application of the TLD principle to the problem of keypoint matching, (2) the replacement of the expensive sliding-window stage for detection in TLD by a highly efficient keypoint detection, description and matching stage, and (3) an experimental evaluation of this approach. This work is structured as follows. In Section II we discuss related work. Section III contains a detailed description of our proposed method. In Section IV we show experimental results. Section V concludes this work.

II. Related Work

There exists a substantial amount of work in the area of machine learning concerned with the question of how to improve supervised classification methods by additional unlabeled data, a discipline known as semi-supervised learning [2]. The key idea here is that the intrinsic structure of the data is exploited to automatically infer a labeling , which is then used to refine the prediction of a classifier. We will now briefly describe selected methods that make use either of domain-agnostic semi-supervised learning methods or of domain-specific knowledge in order to extract labels from videos with the aim of improving performance.

There is some work on the improvement of keypoint matching by using unlabeled data. Meltzer et al. [14] track keypoints in a training phase where small displacements are assumed using the method of Lucas and Kanade [12] and learn the variations in keypoint appearances using kernel principal component analysis. Building on the work of [10], Özuysal et al. [17] "harvest" features in a training stage, where the object of interest is assumed to move slowly. A classifier is trained that is used to establish correspondences of keypoints that appear later in the training phase. These corresponding keypoints are then used to update the classifier. Grabner et al. [5] present an approach that does not rely on a training stage. Instead, they match keypoints by an online boosting approach and estimate a homography based on the successful matches. This homography is used in order to infer a labeling of the keypoints and to update the boosting mechanism.

Unlabeled data has also been used in the context of object tracking. In [3], online boosting is used in order to predict the location of an object. New training data is acquired by performing self-learning, meaning that the prediction itself is used to perform an update of the boosting mechanism. This type of acquiring new labels is known to suffer from the problem of error accumulation. The approach was extended in order to circumvent this problem in [4], where only the first appearance of the object leads to an update and only the classification is improved using unlabeled data. If the prior on the first appearance is too strong, all the other appearances of the object are considered too different and hence will not be found. If the prior is too weak, then clutter in the background will easily be interpreted as the object. In [22] this problem is partially overcome by an adaptive prior.

Recently, Kalal et al. [8] proposed a method called Tracking-Learning-Detection, where a frame-to-frame tracker based on the estimation of optic flow is used to estimate the position of the object in the next frame. This position is used to update the detector in order to account for the new view of the object as well as for false positives. The detector in turn is used to re-initialise the frame-to-frame tracker after failure. This mechanism was shown to be self-stabilising. Our method differs from TLD in the following aspects. First, TLD deals with the problem of object tracking, while we focus on the re-association of keypoints, a problem that is not necessarily tied to the application of object tracking. Second, in TLD a sliding-window based classifier is used in order to perform a search for an object. By replacing this computationally expensive global search of TLD by a keypoint detection and matching stage, we are able to embrace existing methods that are able to efficiently identify interesting image regions.

III. Proposed Method

In this section we give a detailed description of our proposed method. First, we track the keypoint of interest from the previous frame to the current frame (Sec. (III-A). Then we detect a set of keypoints (Sec. III-B) and compute a descriptor for each keypoint (Sec. III-C). We then proceed with the matching of these descriptors (Sec. III-D). Based on the results of the tracker and the matcher we compute a fused result (Sec. III-E) and use it to perform an update of the matcher (Sec. III-F).

A. Frame-to-Frame Tracking

The aim of the frame-to-frame tracker is to provide an estimate o_t of the position of the keypoint of interest, where the subscript t refers to the current frame. To this end, we compute the optic flow from the video frames I_{t-1} to I_t in the point p_{t-1}, which refers to the position of the keypoint in the previous frame. For computing optic flow, we employ a pyramidal implementation of the well-known method of Lucas and Kanade [12], here denoted by LK.

It is desirable to detect errors of the frame-to-frame tracker, as wrong results might lead to an incorrect update of the classifier used for matching. In order to increase the robustness of the LK-method, we compute optic flow bidirectionally, a technique used by many authors, e.g. [7]. The idea here is that the estimation of the optic flow is considered not to be reliable if the estimation is not reversible. To this end, we employ the method of Lucas and Kanade in forward and backward direction

$$o_t = LK(I_{t-1}, I_t, p_{t-1}) \tag{1}$$
$$o'_{t-1} = LK(I_t, I_{t-1}, o_t) \tag{2}$$

and compute a forward-backward error measure FB as in [7] using the Euclidean distance

$$FB = \|p_{t-1} - o'_{t-1}\|. \tag{3}$$

We discard the result of the frame-to-frame tracker when at least one of the following conditions is true: a) LK fails to compute the optic flow, b) p_{t-1} is undefined, c) FB is larger than 20 pixels, d) o_t is outside the image boundaries.

B. Keypoint Detection

The detection of keypoints typically is a very early processing step. Often keypoints are used simply as building blocks in computer vision applications, without attaching any semantic meaning to them. A variety of keypoint detectors exists, differing with respect to various quality measures and computational demands. For an exhaustive survey, see [24]. While in principle any keypoint detector can be used in our method, For our experiments we chose to use the FAST keypoint detector by Rosten et al. [19] for its very low computational demands and its state-of-the-art performance.

The idea of the FAST detector is depicted in Fig. 2. A circle of sixteen pixels is considered around the corner candidate. If there exists a set of 10 contiguous pixels in the circle which are all brighter than the candidate pixel $I(p)$ plus a threshold, or all darker than $I(p)$ minus a threshold, then the candidate is added to the list of corners. Rosten devised a mechanism

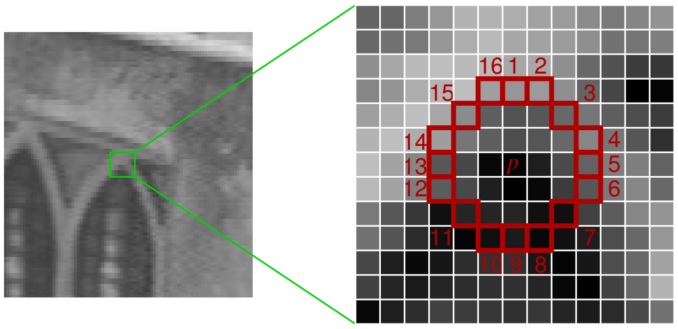


Fig. 2: The FAST corner detection algorithm responds to candidate pixels p if a set of continuous pixels in a circle around p are either brighter or darker than p. Image is from [19].

that evaluates those pixels first that are most likely to result in an information gain. This way, many candidates can be rejected very quickly. Additionally, non-maximal suppression is performed in order to filter out corners that closely lie together. We use a threshold value of 10 for our experiments.

C. Keypoint Description

As wide as the choice of keypoint detectors is the choice of their descriptors. Much work has been done on evaluating the performance of local descriptors (e.g. [15]). SIFT-based descriptors typically rank among the top of these performance evaluations. However, we are interested not only in good matching performance, but also in an efficient implementations. For our experiments, we use the BRIEF descriptor [1] which we will now describe.

The BRIEF descriptor on a keypoint k is based on n tests of the form

$$\tau = \begin{cases} 1 & \text{if } I(p_1) > I(p_2) \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

The locations p_1 and p_2 are relative coordinates and are chosen randomly before the start of the application. The concatenation of all the binary values then yields the descriptor x. We currently use 256 tests. Following [1], in order to achieve robustness against noise, the image is smoothed using a box filter prior to feature computation, which is a simple approximation to the Gaussian filter. We employ an efficient implementation based on integral images [21].

A natural and very efficient way of computing the distance between two BRIEF descriptors x_a and x_b is to employ the Hamming distance

$$d_{HAMM}(x_a, x_b) = \sum_{i=1}^{n} XOR(x_{a_i}, x_{b_i}) \tag{5}$$

which for binary descriptors is equal to the L1-norm.

D. Keypoint Matching

The aim of the keypoint matching stage is to perform an association between positive examples of the keypoint of interest and candidate keypoints found in the image. Additionally, negative examples from other keypoints can be used in order to improve the association. While in principle a nearest neighbour

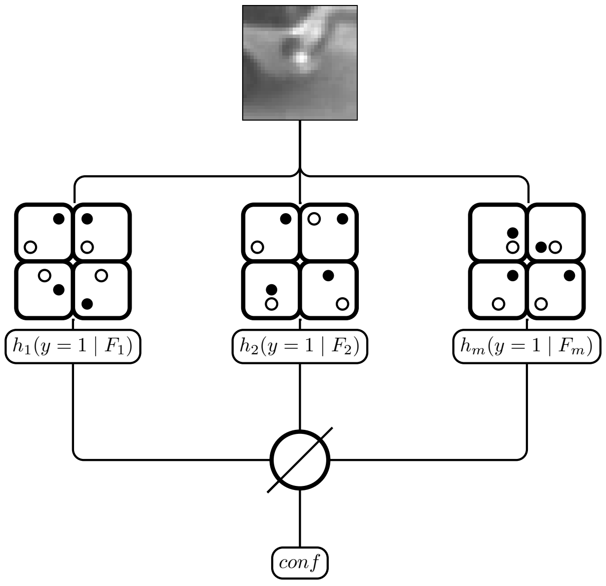


Fig. 3: BRIEF descriptor and random fern classification of a keypoint. The descriptor is split into vectors $F_1 \ldots F_m$ of equal size. Each vector is then converted to a decimal number and used as an index for looking up the posterior probability. A classification result is obtained by averaging all posterior probabilities and performing a thresholding.

search could be used, it has been shown that by employing machine learning techniques, highly efficient keypoint matching can be performed [16]. We formulate the matching of a single keypoint as a binary classification problem with classes $y = 1$ (positive class) and $y = -1$ (negative class) and turn to the technique proposed in [16] known as random fern classification for solving it. For an illustration, see Fig. 3. The key idea is to partition the feature vector x into m subvectors of equal size s, here denoted by $F_1 \ldots F_m$.

$$x = \overbrace{x_1 \ldots x_s}^{F_1}, \overbrace{x_{s+1} \ldots x_{2s}}^{F_2}, \ldots, \overbrace{x_{(m-1)s+1} \ldots x_{ms}}^{F_m}. \tag{6}$$

Instead of classifying the feature vector x as a whole, m classifiers are used, where the classifier h_k operates on the subvector F_k. We follow [8] in the modelling of the classification decision function $h_k(y = 1|F_k)$ of each classifier as

$$h_k(y = 1|F_k) = \frac{p_{F_k}}{p_{F_k} + \max(n_{F_k}, 1)}. \tag{7}$$

The term p_{F_k} refers to the number of times that the feature vector F_k was used as a positive training example and vice versa for n_{F_k}. If no training examples have been seen yet for a subvector F_k, then we assume that it belongs to the negative class, as the distributions of the classes are highly imbalanced towards the negative class. This idea is reflected in Eq. 7 by the term $\max(n_{F_k}, 1)$. Since F_k can be interpreted as a binary digit, it can serve as the index to an array where the $h_k(y = 1|F_k)$ are stored, making this calculation extremely efficient.

We combine the individual responses by

$$conf = \frac{1}{m} \sum_{i=1}^{m} h_i(y = 1|F_i), \tag{8}$$

$$\hat{y} = \text{sgn}\left(conf - \theta\right), \tag{9}$$

which essentially is an estimate of the label of the descriptor. In every frame, we estimate the label of all keypoints in I_t. Note that it is possible for multiple keypoints to be classified as positive, a situation that will be dealt with in the next section. The threshold θ can be increased in order to allow for a more conservative matching. For our experiments we set it to 0.8. We also set the fern size s to 8 and the number of ferns m to 32.

E. Fusion

We use the following heuristic for fusing the results of the tracking and matching stage into the final result p_t. If exactly one candidate keypoint was matched to the positive class, we set p_t to the location of this keypoint. if this is not the case, meaning that either multiple keypoints or no keypoint at all were matched positively, we set p_t to the outcome of the tracker, which, depending on the tracking process may or may not be defined. In short, we always give preference to the detector, as long as it produces a single result. We argue that the risk of making a wrong decision when selecting a positively labeled keypoint is lower than when a sliding window is selected out of tens of thousands of candidate windows, as the number of keypoints typically is much lower.

F. Learning

The aim of a matching algorithm is to find as many correct matches as possible, while avoiding both wrong matches and missed matches. In order to achieve this aim, we update the classifier whenever we can identify an error. Due to its adaptive nature, the tracker will track keypoints successfully even if their appearance changes slightly. If the matching algorithm already identifies this new view as a correct match, we do not perform an update. If however, the new view of the keypoint is matched to the background class, we have successfully identified a missed match and use the corresponding descriptor as a positive training example, meaning that we increment p_{F_k} for all individual classifiers in Eq. 7. We identify wrong matches the following way. When p_t is defined by the tracking result and the matcher identified more than keypoint to be a match, we use the descriptors of these keypoints as negative training examples and increment the correspoding n_{F_k}. Intuitively, this means that the matcher has given a contradictory result, since we assume that the keypoint exists only once. A wrong match might lie close to p_t, which happens as nearby keypoints sometimes share similar descriptors. We therefore do not use wrong matches as training examples that are closer than 20 pixels to p_t.

An important aspect that has to be considered when employing an adaptive method such as Lucas-Kanade is the danger of drift. Even though we presented some constraints in Sec. III-A that allow us to detect inconsistencies in the frame-to-frame tracker, it still can happen that the tracker wanders off, for instance in the case of slow occlusions. A certain degree of adaptivity is desirable in order for the learning mechanism to have an effect, but the question is when the border is crossed from tracking a novel view of a keypoint to tracking a wrong keypoint. In the context of object tracking, both Kalal et al. [8] and Santner [20] argue that maintaining static examples eases this problem to some extent, as these examples are never altered during the update process and thus can be used to achieve more stable results.

We follow this argumentation and use the descriptor of the initial keypoint x_0 as a single static template. In every learning step we compute the distance of the current keypoint descriptor to the initial descriptor

$$\delta_t = d_{HAMM}(x_0, BRIEF(p_t)) \tag{10}$$

and perform an update of the classifier only if δ_t is smaller than a threshold

$$\delta_t < \theta_{static}. \tag{11}$$

The threshold θ_{static} affects the adaptivity of the system. When θ_{static} is low, then only a small amount of new views of the keypoint of interest will be discovered. When θ_{static} is maximal, then all the discovered views will be used for updating. Currently, we use $\theta_{static} = 30$.

IV. Experimental Results

A. Qualitative results

For evaluation purposes, we employed 5 sequences from the literature: car [25], david indoor [18], ball [9], dudek [6] and lemming [20]. Qualitative results for representative keypoints on these sequences are shown in Fig. 4. These results demonstrate the ability of our system to robustly perform keypoint assocation over the course of long sequences and various kinds of changes in appearance. The last column of each row confirms the necessity of incorporating false positives into the matching process. Even for humans, the last picture of the fourth row is highly ambiguous.

B. Keypoint Evolution and System Reaction

In this experiment, we provide evidence that the descriptor of a keypoint evolves over time and show how our system reacts to these changes. To this end, we initialised our system with the keypoint shown in the first image of the top row of Fig. 4, corresponding to a corner of a vehicle. It is obvious to a human beholder that the view on the keypoint changes over time (see the other images of the same row). We visualise this change by plotting the distance δ between the descriptor of the current location estimate and the initial descriptor (Eq. 10) in each frame of the sequence car in Fig. 5. Starting from the first frame, δ increases even if the keypoint is matched correctly. The amplitudes correspond to situtations when the tracker drifts away from the keypoint. A sudden drop in d_t typically corresponds to a re-initialisation by the matcher.

C. Matching Comparison

In this experiment, we perform a comparison of the matching performance between our proposed method and a standard method of matching BRIEF descriptors. For the standard method, we compute BRIEF descriptors on all FAST keypoints in the current image. We then perform matching by finding the keypoint that exhibits minimal distance to the initial keypoint.

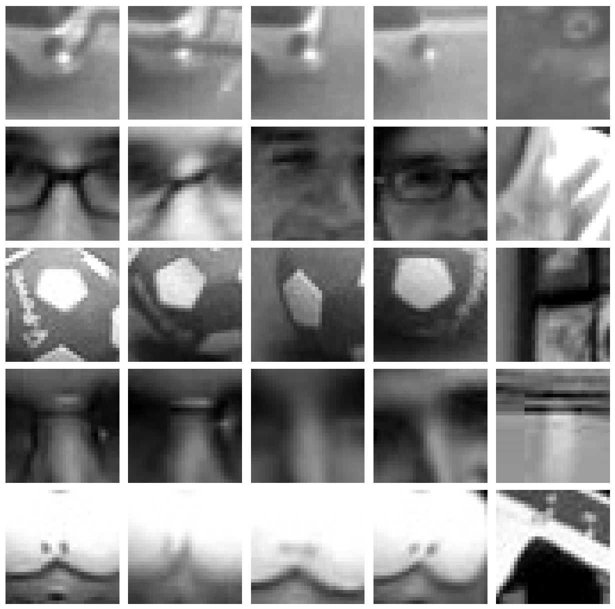


Fig. 4: Qualitative results on sequences from the literature. The initial selection of the keypoint of interest is shown in the first image of each row. The middle images in each row depict correct matching results, while the last column shows a false positive that was discovered. From top to bottom: car, david indoor, ball, dudek, lemming.

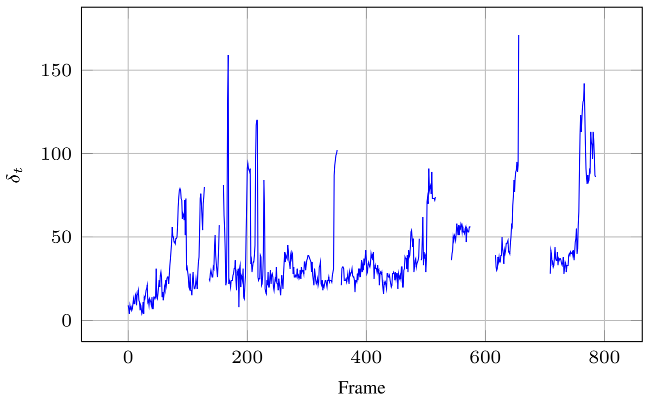


Fig. 5: Hamming distance δ_t between the descriptor at position p_t and the initial descriptor on the car sequence. It can be seen that the distance to the initial descriptor becomes considerably larger in later frames. Amplitudes typically correspond to errors of the frame-to-frame tracker.

For computing the matching performance we follow the work of Mikolajczyk and Schmid [15] and measure

$$Recall = \frac{\#correct}{\#keypoint} \tag{12}$$

and

$$Precision = \frac{\#correct}{\#correct + \#incorrect} \tag{13}$$

The term $\#visible$ refers to the number of frames where the keypoint is visible. In precision-recall curves matching performance is plotted as the threshold of the matcher is varied

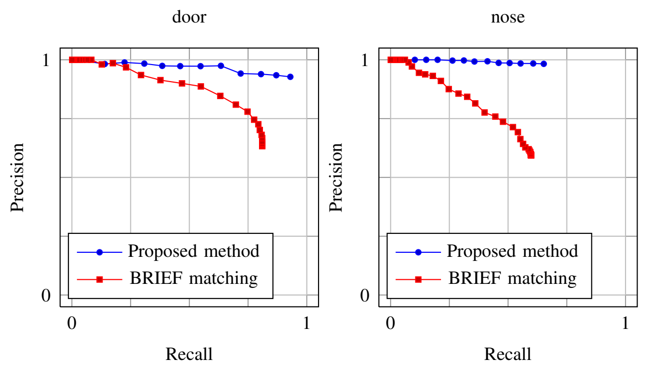
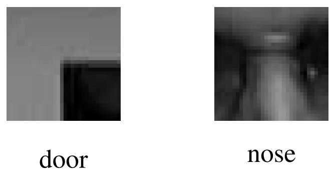


Fig. 6: Precision-recall curves on two keypoints. Our proposed approach clearly outperforms the standard method regardless of which thresholds are used.

from its minimum value to its maximum value. When the threshold is at its minimum value, all matching results are kept, resulting in maximum recall. When the threshold is increased, both the number of correct and incorrect matches decrease, which leads to a decrease of recall and to an increase in precision. For our method, we threshold the confidence values computed in Eq. 8. For the standard BRIEF matching, we employ the threshold on the distance between the descriptor of the matched keypoint and the descriptor of the initial keypoint.

In order to compute precision and recall, we manually annotated two keypoints in two different sequences, a stable one corresponding to the upper left corner of a door and a dynamic one, corresponding to a nose.



The door keypoint does not change its appearance, but it gets occluded multiple times. The nose keypoint undergoes a variety of changes, as it can be seen in the fourth row of Fig. 4. In order to compute $\#correct$ and $\#incorrect$, we measure the Euclidean distance between the matched keypoint and the annotated keypoint. If the distance does not exceed a certain threshold, we increment $\#correct$. If the distance exceeds the threshold, or if the matching algorithm gives a result, even though the keypoint is not visible in the image, we increment $\#incorrect$. We currently use 10 pixels as the distance threshold. In Fig. 6 the precision-recall curves are shown. For both keypoints, our method clearly is superior to the standard BRIEF matching algorithm.

D. Execution Time

In order to assess the computational demands of our proposed method, we measured wall clock time for each of the operations described in Sec. III on a smart camera equipped with an Intel Atom N270 CPU. In Tab. I the timings are shown for the two most expensive stages (keypoint detection and

Sequence	Resolution	Detection	Description	Total
Ball	320×240	0.015	0.029	0.062
Car	320×240	0.011	0.014	0.057
David	320×240	0.013	0.050	0.100
Dudek	720×480	0.042	0.196	0.392
Lemming	640×480	0.051	0.386	0.599

TABLE I: Timings in seconds.

description) as well as for the total execution time. While the time for detecting keypoints depends mainly on the resolution of the input images, the description stage depends on the number of keypoints found in an image. This explains why processing time for the Lemming sequence is higher than that of the Dudek sequence, even though the resolution of the Lemming sequence is lower. The Lemming sequence consists of many different objects, thus yielding a high number of keypoints, while the background in the Dudek sequence is homogenous. This means that it is reasonable to adjust the sensitivity of the keypoint detector in order to reduce computational demands. For scenes with little texture, such as the Ball and Car sequences, real-time processing is reachable.

V. Conclusions

Our experimental results have shown that the TLD principle is well suited for the robust matching of single keypoints and is able to cope with a variety of appearance changes. We also have shown how to avoid the expensive sliding-window approach of TLD by employing already existing methods for keypoint detection and matching. The most compelling open task is to extend this method to the simultaneous association of multiple keypoints, which we will address in future work. It also might be interesting to incorporate the keypoint detector into the learning process, as the sensitivity of the feature detector obviously plays an important role in avoiding wrong decisions. At present, an unsolved problem is how to perform an extensive performance evaluation, as to the best of our knowledge no ground truth for long-term keypoint matching in videos exist. To overcome this problem, the creation of a data set of annotated keypoints in various sequences is desirable.

Acknowledgements

The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement no 257906.

References

[1] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua, "BRIEF: Computing a local binary descriptor very fast," *Pattern Analysis and Machine Intelligence*, vol. 34, no. 7, pp. 1281–1298, Jul. 2012.

[2] O. Chapelle, B. Schölkopf, and A. Zien, Eds., *Semi-Supervised Learning*. The MIT Press, Sep. 2006.

[3] H. Grabner and H. Bischof, "On-line boosting and vision," in *Computer Vision and Pattern Recognition*, vol. 1, Jun. 2006.

[4] H. Grabner, C. Leistner, and H. Bischof, "Semi-supervised On-Line boosting for robust tracking," in *European Conference on Computer Vision*, 2008.

[5] M. Grabner, H. Grabner, and H. Bischof, "Learning features for tracking," in *Computer Vision and Pattern Recognition*. IEEE, Jun. 2007, pp. 1–8.

[6] A. D. Jepson, D. J. Fleet, and T. F. El-Maraghi, "Robust online appearance models for visual tracking," *Pattern Analysis and Machine Intelligence*, vol. 25, no. 10, pp. 1296–1311, Oct. 2003.

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

[8] ——, "Tracking-Learning-detection," *Pattern Analysis and Machine Intelligence*, vol. 34, no. 7, pp. 1409–1422, Jul. 2012.

[9] D. A. Klein, D. Schulz, S. Frintrop, and A. B. Cremers, "Adaptive real-time video-tracking for arbitrary objects," in *International Conference on Intelligent Robots and Systems*. IEEE, Oct. 2010, pp. 772–777.

[10] V. Lepetit, P. Lagger, and P. Fua, "Randomized trees for Real-Time keypoint recognition," in *Computer Vision and Pattern Recognition*, vol. 2. Los Alamitos, CA, USA: IEEE, 2005, pp. 775–781.

[11] D. G. Lowe, "Distinctive image features from Scale-Invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[12] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 1981, pp. 674–679.

[13] E. Maggio and A. Cavallaro, "Hybrid particle filter and mean shift tracker with adaptive transition model," in *International Conference on Acoustics, Speech, and Signal Processing*, 2005, pp. 221–224.

[14] J. Meltzer, M.-H. Yang, R. Gupta, and S. Soatto, "Multiple view feature descriptors from image sequences via kernel principal component analysis," in *European Conference on Computer Vision*, 2004, pp. 215–227.

[15] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *Pattern Analysis and Machine Intelligence*, vol. 27, no. 10, pp. 1615–1630, 2005.

[16] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua, "Fast keypoint recognition using random ferns," *Pattern Analysis and Machine Intelligence*, vol. 32, no. 3, pp. 448–461, 2010.

[17] M. Özuysal, V. Lepetit, F. Fleuret, and P. Fua, "Feature harvesting for Tracking-by-Detection," in *European Conference on Computer Vision*. Springer, 2006, pp. 592–605.

[18] D. Ross, J. Lim, R.-S. Lin, and M.-H. Yang, "Incremental learning for robust visual tracking," *International Journal of Computer Vision*, vol. 77, no. 1, pp. 125–141, May 2008.

[19] E. Rosten, R. Porter, and T. Drummond, "Faster and better: A machine learning approach to corner detection," *Pattern Analysis and Machine Intelligence*, vol. 32, no. 1, pp. 105–119, Jan. 2010.

[20] J. Santner, C. Leistner, A. Saffari, T. Pock, and H. Bischof, "PROST: Parallel robust online simple tracking," in *Computer Vision and Pattern Recognition*, 2010.

[21] P. Y. Simard, L. Bottou, P. Haffner, and Y. LeCun, "Boxlets: a fast convolution algorithm for signal processing and neural networks," in *Conference on Advances in Neural Information Processing systems*. Cambridge, MA, USA: MIT Press, 1999, pp. 571–577.

[22] S. Stalder, H. Grabner, and L. van Gool, "Beyond semi-supervised tracking: Tracking should be as simple as detection, but not simpler than recognition," in *International Conference on Computer Vision Workshops*. IEEE, 2009, pp. 1409–1416.

[23] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer, 2010.

[24] T. Tuytelaars and K. Mikolajczyk, "Local invariant feature detectors: a survey," *Foundations and Trends in Computer Graphics and Vision*, vol. 3, pp. 177–280, Jul. 2008.

[25] Q. Yu, T. B. Dinh, and G. Medioni, "Online tracking and reacquisition using co-trained generative and discriminative trackers," in *European Conference on Computer Vision*, 2008.