Machine Vision and Applications

Real-Time Tracking-with-Detection for Coping With Viewpoint Change

Shaul Oron · Aharon Bar-Hillel · Shai Avidan

Received: 11 May 2014 / Revised: 02 Nov 2014 / Accepted: 09 Mar 2015

Abstract We consider real-time visual tracking with targets undergoing view-point changes. The problem is evaluated on a new and extensive dataset of vehicles undergoing large view-point changes. We propose an evaluation method in which tracking accuracy is measured under real-time computational complexity constraints and find that state-of-the-art agnostic trackers, as well as class detectors, are still struggling with this task. We study tracking schemes fusing real-time agnostic trackers with a non-real-time class detector used for template update, with two dominating update strategies emerging. We rigorously analyze the template update latency and demonstrate such methods significantly outperform stand-alone trackers and class detectors. Results are demonstrated using two different trackers and a state-of-the-art classifier, and at several operating points of algorithm/hardware computational speed.

Keywords Tracking · Detection · Real time · Fusion

S.Oron
Tel Aviv University
Tel Aviv 69978, Israel
E-mail: shauloro@post.tau.ac.il

A.Bar-Hillel
Advanced Technical Labs Israel,
Microsoft research, Israel
E-mail: aharonb@microsoft.com

S.Avidan
Tel Aviv University
Tel Aviv 69978, Israel
E-mail: avidan@eng.tau.ac.il

1 Introduction

Visual tracking is a fundamental problem in computer vision and a key component in many higher level vision applications such as surveillance [40], robotic vision [34], active vehicle safety systems [3] and natural user interface [37] to name a few.
Performing reliable tracking is very challenging with many obstacles to overcome. One such challenge is handling large view-point changes or out-of-plane rotations. In such cases the tracking problem becomes extremely difficult as it gains an almost unsupervised nature: the target has to be detected after seeing one (or few) relevant examples, and its similarity even to these examples is limited. Coping with severe view-point changes requires updating the target template which is a difficult task with many pitfalls. In addition, in an application context, visual tracking is almost always required to operate in real-time, i.e. frames must be processed as fast as they are streaming in. This constraint is an essential aspect of the problem, as complying with it usually comes at the expense of accuracy.

We focus on real-time tracking of targets from a *known* class, in scenarios where these targets undergo severe view-point changes. Unlike agnostic trackers, that are not aware of the target class and train classifiers on-line, we assume the target class is known and train a class detector off-line.

We demonstrate that real time tracking under large view-point changes is difficult for state-of-the-art agnostic tracking methods and class detectors. To address this problem we propose several strategies for real-time tracking with detector assisted template update which we name tracking-with-detection. These methods are shown to outperform both trackers and class detectors when evaluated under run-time constraints.

In order to systematically span a space of targets undergoing severe view-point changes, we have created a challenging new dataset, which will be made publicly available. This dataset contains 167 annotated vehicle sequences captured from cameras mounted on a moving car. These sequences were chosen from a larger corpus mainly focusing on road scenes in which the target vehicle undergoes large view-point and scale changes (examples shown in figure 6).

The reason we propose a new dataset in this work is that we want to focus specifically on view-point changes in a setup where both camera and target are maneuvering. We are not aware of any other dataset with a similar emphasis. Moreover, as we show in the experimental section, performance of state-of-the-art tracking methods on this newly proposed dataset indicate that the tracking problem is still far from being solved, even in this simplified setup where the deformation space is limited almost exclusively to view point change and targets are rigid. This is true either with or without runtime constraints.

We propose a new evaluation methodology in which accuracy is measured under a computational complexity constraint. Correct evaluation methodology is important both for evaluating overall algorithm performance as well as for parameter tuning. Previous work on tracking evaluation focus only on algorithm accuracy. In this work we take the discussion on tracking evaluation one step further and argue that run-time is an integral part of tracking and as such tracking evaluation cannot be done without considering run-time constraints.

We attempt to systematically introduce real-time constraints into tracking performance evaluation and proposed a new evaluation protocol. The proposed method has the desired property of being generic with respect to the accuracy evaluation metric chosen, meaning any accuracy measure can be used.

In the proposed evaluation method computation time is measured on the fly and an algorithm is evaluated in a strictly causal manner. An algorithm operating slower than the frame rate would obtain a delayed state hypothesis, and missed frames will be evaluated using the last known state. We demonstrate how this evaluation methodology can be generalized, emulating more efficient algorithm implementations and faster / slower hardware configurations, allowing one to gain a fairly general understanding of the accuracy / processing-speed trade-off and providing a powerful design tool.

We establish a benchmark for real-time tracking under severe view point changes using our data. This is done by evaluating six publicly available, state-of-the-art, tracking methods. We find that all methods struggle when evaluated for real-time performance, and even when unconstrained accuracy is considered (i.e. algorithms are assumed to perform in real-time), the tested methods still leave room for improvement. These results demonstrate the difficulty of traditional tracking methods in coping with large target view-point changes, even without run-time constraints.

In light of these results we propose incorporating a detector into the tracking system, and demonstrate that without run-time constraint a detector carefully designed for the tracking task can deliver superior performance. We then address the question: how to compose a tracking system that incorporates a detector and runs in real-time ? Although tracking-by-detection or incorporating trackers and detectors is not a new concept, most such systems run detection exhaustively and are unable to deliver real-time performance. In general, some real-time detectors do exist, but still most state-of-the-art classifiers are computationally complex and not real-time. Therefore *tracking-with-detection* which enables non-real-time detectors to be incorporated into real-time tracking systems is an important contribution.

In this context we propose several strategies for performing tracking-with-detection, using the detector to validate the tracker response as well as for template update purposes. We demonstrate the performance improvement obtained both over individual system components as well as state-of-the-art methods. In addition, we analyze the template update latency associated with each such system and look into the relation between latency, tracking accuracy and template update accuracy.

The contributions of our work are two fold: i) Introducing and analyzing several tracking-with-detection methods and demonstrating their superior performance for tracking a known class under severe view point changes, and discussing latency accuracy relations. ii) Providing a new dataset and benchmark for tracking under severe view-point changes, and a new methodology for evaluating tracking algorithms under run-time constraints.

2 Related work

In this work we benchmark several trackers on our proposed dataset demonstrating the difficulty of tracking under view point changes and establishing that this problem is far from being solved. Readers interested in benchmark results for visual tracking methods are referred to the work of Wu et al. [42]. For a through survey of the vast literature on tracking techniques we refer the reader to Yilmaz et al. [43]. We focus our attention on real-time visual tracking research and work related to detector-tracker fusion and template update.

Handling drift in visual tracking systems is a difficult task, often addressed by some form of target template update. Different strategies have been proposed to perform template update. The work of Matthews, Ishikawa and Baker [31] is an early attempt to address this problem directly. Their method, like many others that followed, propose a template update strategy regularized by the first template (i.e. at frame 1) and using some template appearance basis that can change over time [10,30]. This work inspired methods using multiple templates and sparse representations to represent target appearance [6,23,35,45]. Another widely used strategy for template update is using on-line learning based methods. This can be achieved either by pixel-wise classification [4,21], or exemplar based classification [5,44]. These methods learn a classifier on-line adjusting it according to target appearance changes. One prominent work in this category is that of Kalal et al. [24] learning an on-line target detector, updating it by mining positive and negative examples. However, this technique is agnostic to the target class and learns a detector on-line. It also keeps using the initial target appearance to avoid drift, unlike our proposed method. An additional strategy that can be viewed as a type of template update is updating the template not directly but rather through adjusting parameters controlling the space of possible matches [26,32].

Tracking-by-detection or incorporating trackers and detectors [2,12,27,38] is an appealing approach for limiting drift and producing reliable tracking. This idea can be used when one knows the class of objects being tracked. Unfortunately the above mentioned methods require running the detector exhaustively on each frame and thus are unable to run in real-time. Fan et al.[15] propose a unified approach of tracking and recognition incorporating trackers and detectors for multiple object classes, however their method is also unable to run in real-time and not evaluated under such constraints. Some template-based detectors for specific classes (pedestrians, faces) achieve real-time detection [8,11] and enable real-time tracking-by-detection. However, state-of-the-art general-purpose classifiers usually do not operate in real-time. Such classifiers often include multiple deformable parts [7,19], and/or multiple prototypes for handling different viewpoints or large class variability. More generally, classification is far from being a solved problem, and in many cases requires considerable computations. It is likely that while performance improves, state-of-the-art classifiers will remain slow in the near future (for example because of the need to run on weak hardware) and the best classifiers will usually not be real-time ones. On the other hand, many visual tracking algorithms are able to operate at close to real-time or faster [30,31,45]. Our work hence focuses on fusing non real-time detectors with real-time tracking algorithms.

Evaluation methodology for visual tracking was discussed in multiple papers and an elaborated review of performance measures and evaluation programs can be found in [9,25,28]. However we are not aware of any previous work attempting to systematically introduce run-time constraints into tracking evaluation.

3 Tracking-with-detection

We present several tracker-detector fusion schemes, evaluating them under real-time constraints. We consider trackers of various speeds integrated with a 'slow' detector for which typical detection takes a few frames to carry out. In these systems the tracker is responsible for continuously tracking the target while the detector provides updates to the appearance of the target being tracked. In this context we analyze the latency associated with updating the template, i.e. the time or number of frames that pass between the beginning of a detection process until the template is updated.

For the latency analysis the following notations are used. Let us denote the input frame rate f_{in} and the interval between frames $t_{in} = \frac{1}{f_{in}}$. We denote t_{det} the time required to obtain a detection and t_{trk} the tracker processing time for a single frame. Assuming this tracker operates in real-time i.e. $t_{trk} < t_{in}$, we denote by t_{idle} the time between consecutive frames not used for tracking computations:

$$t_{idle} = t_{in} - t_{trk} \tag{1}$$

This idle processing bandwidth can be used to perform additional tasks such as detection as will be discussed later. Intuitively, larger latency entails accuracy decrease since by the time a new template is available, it is already outdated. In section 4 we show empirically that this is indeed the case.

3.1 Tracking-by-detection

We begin with the known tracking-by-detection scheme (e.g. [2,12,27]). In this scheme a detector is used for tracking by performing detection for every frame streaming in. In this system the detector performs the tracking and there is no additional tracker. We can compute the latency l_{det0} associated with the detection process:

$$l_{det0} = \frac{t_{det}}{t_{in}} \tag{2}$$

This means that every time we perform detection it takes us l_{det0} frames until the results is obtained and updated. We note that, of course, if $t_{det} < t_{in}$ resulting in $l_{det0} < 1$ then we have a system working in real time without any latency.

3.2 Template update with near-real-time tracker

We consider scenarios where $t_{trk} \approx t_{in}$ i.e. the tracker is near-real-time, but not much faster (or slower). In this situation applying the tracker requires almost all the computational bandwidth i.e $t_{idle} \to 0$. In such a scenario, for any given frame, we can either run tracking or perform detection in order to update our template (but not both). If we perform detection then we suffer a latency of l_{det0} frames, as in the tracking-by-detection case, and this affects accuracy. On the other hand if we do not perform template update our tracker may drift, losing the target all together. This trade off leaves us with the difficult question of “When should we update the template?". A trivial solution would be to perform template update on a regular basis, using some fixed interval i.e. performing template update every K frames. However we would like to perform tracking most of the time updating the template only when necessary in order to avoid state extrapolation which damages accuracy. A key observation in this respect is that classification is much faster than detection, since the latter requires, at least implicitly, classification of many locations and scales. A single detector query (classification) can usually be done very fast even for a complex detector. In light of this observation we propose the validation scheme, resembling [41]. In this scheme the tracker result, i.e. its proposed state, is validated by classifying it using the detector. If the validation succeeds, meaning the proposed state is a valid detection, we continue tracking. Only when the validation fails we resort to template update by running a detection scheme in a window around the last known target state, while extrapolating until a detection is obtained. Once we obtain the detection results (although delayed) we update template appearance and continue from the last known target state. The described scheme answers the question of "when should we update the template?", and is expected to outperform the naive fixed interval template update.

3.3 Template update with a fast tracker

In this scenario $t_{trk} \ll t_{in}$, which means that $t_{idle} \gg 0$, i.e. we have substantial idle time, not used for tracking computations, that can be used for other purposes. We propose using this idle time to perform detection and update the template. The most simple way to do this would be to run the detector in the idle time and update the template once a detection is obtained. Using this update scheme would result in template update latency of l_{det1} frames.

$$l_{det1} = \frac{t_{det}}{t_{idle}} = \frac{t_{det}}{t_{in} - t_{trk}} \tag{3}$$

The interpretation of this scenario is that tracking is performed continuously with template updates occurring every l_{det1} frames. In this scenario the template is updated once obtained although it is relevant to l_{det1} frames ago. We note that equation (3) is only meaningful when $t_{in} > t_{trk}$, also naturally $l_{det0} < l_{det1}$ (for any feasible setup where $t_{trk} > 0$).

We consider a second option. Since any detection obtained is delayed, i.e. relevant to some past frame, we propose tracking it to the current frame rather than instantaneously updating the template. This means we take the new detection, 'catch-up' to the input stream, and only then perform the template update. This would of course result in longer latency but might provide more accurate appearance update. We denote the latency associated with this strategy by l_{det2}. This latency can be broken down into two parts:

$$l_{det2} = l_{det1} + l_{trk} \tag{4}$$

where l_{det1} is the detection latency calculated earlier and l_{trk} the tracking latency induced by tracking the detection to the current frame (the 'catch-up'). We can deduce l_{det2} from the following equation:

$$l_{trk} \cdot t_{idle} = (l_{det1} + l_{trk}) \cdot t_{trk} \tag{5}$$

To understand this equation note that the number of frames between the time a detection is obtained and until we catch-up is l_{trk}. This means the catch-up tracker has a time interval of $l_{trk} \cdot t_{idle}$ (left-hand side) in which it has to track through $l_{det1} + l_{trk}$ frames (right hand side). Isolating l_{trk} we have:

$$l_{trk} = \frac{l_{det1} \cdot t_{trk}}{t_{idle} - t_{trk}} = \frac{l_{det1} \cdot t_{trk}}{t_{in} - 2t_{trk}} \tag{6}$$

Substituting l_{trk} and l_{det1} in equation (4) gives:

$$l_{det2} = l_{det1}(1 + \frac{t_{trk}}{t_{in} - 2t_{trk}}) = \frac{t_{det}}{t_{in} - 2t_{trk}} \tag{7}$$

We observe that equation (7) is only meaningful when $t_{in} > 2t_{trk}$, since this method can only work if two tracker applications can be done faster than the inter-frame interval (one for the main tracker, and one for the 'catch-up' tracker).

From a system design perspective, in order to avoid very long latencies, limitations on t_{trk} emerge. In the scenario of instantaneous update, with latency l_{det1}, we should have $t_{trk} \ll t_{in}$. If we perform 'catch-up', with latency l_{det2}, then we require $t_{trk} \ll \frac{1}{2}t_{in}$.

Comparing the performance of these two strategies provides interesting insights into the trade off between tracking accuracy, template update accuracy and template update latency, and will be further explored in the experimental section.

4 Experiments

We evaluate tracking-with-detection using the vehicle object class. We present a new dataset, which will be made publicly available[1]. The data contains 167, fully annotated sequences, ranging from 100 to 1200 frames. These sequences were captured from 3 cameras (two grayscale, one color), recording $640 \times 480p$ images at $f_{in} = 15fps$. The cameras were mounted on a moving car, facing backwards. These sequences were mostly chosen to exhibit road scenes in which cars undergo large view-point and scale changes. The data contains 77 overtake sequences, 22 traffic circles, 42 turn and 26 tailing sequences with over 35,000 frames overall. Typical examples from this data are presented in figure 6. For our experiments the data was divided into a training set

[1] http://www.eng.tau.ac.il/~oron/TWD/TWD.html

comprised of 117 sequences and a test set of 50 sequences. The sequences were divided between the train and test sets such that both sets have examples of overtakes, traffic circles, turns and tailing.

The rest of this section is organized as follows. We begin by introducing a new real-time evaluation methodology for real-time and non-real-time algorithms under run-time constrains. We then provide implementation details related to our experiments. Next, we present benchmark results of several state-of-the-art tracking methods on our dataset, followed by tests demonstrating the performance of our tracking-with-detection schemes. We validate empirically our latency analysis equations. Finally we demonstrate how our real-time evaluation methodology can be used to emulate more efficient algorithm implementations or different hardware operating points providing a powerful system design tool.

Algorithm 1: Evaluation protocol considering run-time constraints

```
   Input: $\{f_i\}_{i=1}^N$ Sequence of video frames
   Output: $\{X_i\}_{i=1}^N$ State hypothesis for each input frame
1  Set $t \leftarrow 0$ (initialize time line)
2  Set $f \leftarrow f_1$
3  while $t < N/f_{in}$ do
4      Set $f \leftarrow \lfloor t \cdot f_{in} \rfloor$
5      if frame $f$ has already been processed then
6          Set $t \leftarrow \lceil t \cdot f_{in} \rceil / f_{in}$
7      else
8          Send frame $f$ to $Alg$ for processing
9          Upon receiving algorithm response $X$ and $t_{proc}$
10         Set $t \leftarrow t + t_{proc}$
11         Record state $X$ for frame $\lfloor t \cdot f_{in} \rfloor$ (but not for earlier
           frames)
12     end
13 end
14 for each frame $f$ from $0$ to $N-1$ do
15     if frame $f$ was not processed i.e. has no recorded response
       then
16         Use the last recorded response (prior to $f$) as the state
           at frame $f$
17     end
18 end
```

4.1 Evaluation methodology

Most applications requiring tracking also require it to operate in real-time. Therefore evaluation of tracking algorithms should also be carried out considering run-time constraints. We suggest a methodology allowing any algorithm to be evaluated under real-time constraints, inherently demonstrating the speed-accuracy tradeoff. The proposed protocol is complementary to previous works on tracking evaluation in the sense that it is accuracy measure independent meaning any accuracy measure can be used with our protocol. We also discuss and demonstrate how the proposed methodology can be used to simulate acceleration of different system components as well as simulate running on platforms with different computational power. In doing so one can gain an understanding of the speed-accuracy tradeoff in a given tracking system.

Simulating real-time is done by measuring actual processing times, on the fly (not including I/O), of the algorithm used for tracking, denoted Alg. We note that Alg is a tracking algorithm in the broad sense of the word and can be a tracker, a detector (used for tracking), or a fusion of the two. Denote the input frame rate f_{in} and the number of frames in the video by N, indexing them $0, .., N-1$. The algorithm Alg receives a certain frame f and returns a state X and its processing time t_{proc}. Algorithm Alg is evaluated based on a sequence of N responses, with response i reflecting its knowledge of state X at time $(i+1)/f_{in}$, i.e. when frame i is replaced by its successor. The algorithm response sequence is determined using the protocol given in algorithm 1.

The scheme described in algorithm 1 can be trivially extended to handle tracking with multiple components (e.g. a tracker and a classifier). Note that for 'skipped' frames (frames with no recorded response) we perform zero-order-hold (ZOH) extrapolation i.e. holding the last known target state. Our main justifications for using ZOH over more complex extrapolation techniques is the desire to make a clean comparison, that does not depend on the extrapolation method and its fit to various tracking schemes.

As presented above, the evaluation protocol is limited to measuring real-time tracking performance of specific algorithm implementations and specific hardware. However, the protocol can easily simulate faster or slower computing speed of all, or part, of the algorithms, simply by dividing the measured computation time t_{proc} by an 'acceleration factor' α. For example, dividing t_{proc} by $\alpha = 2$ predicts the tracking performance using twice-as-powerful hardware. Moreover, this evaluation scheme may guide more subtle engineering decisions. For example, given some fused detector-tracker system one might be interested in knowing: What would happen if we implement one of the components more efficiently? Where should we invest our optimization efforts? To answer such questions one can assign two different virtual acceleration coefficients for both tracker α_{trk} and detector α_{det}.

For measuring tracking accuracy we adopt the method presented in a recently published tracking benchmark [42]. For each frame we measure the bounding box overlap given by $overlap = \frac{area(B_d \cap B_{gt})}{area(B_d \cup B_{gt})}$ where B_d and B_{gt} denotes the detected and ground truth bounding boxes respectively. We build a curve showing success rate, i.e. the fraction of frames with $overlap \geq threshold$ for $threshold \in [0, 1]$ for each sequence and average over all sequences in order to obtain a final success curve. We then measure the area under the curve (AUC). This measure quantifies both centering accuracy as well as scale accuracy giving a broad performance overview, not restricted to a single threshold value (which might be biased).

4.2 Implementation detail

We have implemented the different tracking with detector assisted template update systems presented in section 3. The detector used in our implementations is a DPM [17]-based detector, which was adapted for the tracking task. The classifier is based on the cascade implementation [17] available with the on-line code package [16]. We trained a view-point invariant vehicle classifier with resemblance to the vehicle classifier of Leibe et al.[27]. Four models were trained, each capturing the vehicle in a different-view point ($0°$, $\pm 30°$, $\pm 60°$, $\pm 90°$). The classifiers were trained on examples extracted from our training set of 117 sequences.

The basic tracker used is our own implementation of the standard optical flow LK tracker [30], this tracker searches for a rigid 2D warp, with 3 degrees of freedom (x,y,scale), between the target template and the new frame starting from the last known target location and scale.

Turning the DPM classifier into a tracking-by-detection algorithm that runs at a reasonable rate was done in the following manner. First, spatial, scale and view-point consistencies were considered, i.e. detection is performed in a small region-of-interest (ROI), around the last known target state, and only in the last known and adjacent view-points (e.g. if the last view-point was $30°$ we search at $0°, 30°, 60°$). Multiple detections, in the ROI, are ranked based on classification score as well as spatial and scale consistencies relative to the last known state (modeled using normal distributions around the last known location and scale). Secondly, standard DPM implementation performs a fine grain scale space search using 6 octaves each of which is divided into many sub-scales resulting in ~ 90 scales. This is done since the detector has no prior knowledge regarding target scale. In tracking, on the other hand, one has a good estimation of target scale using the current state. This allows using fewer scales and in fact we limit DPM scale search to 3 octaves $(0.25 - 2)$ containing only 25 fine grain scales. The scale 0.75 is located in the middle of this scale range (i.e. at scale index 13) and we resize our image aiming that the target is detected at this scale. This is done in order to ensure the target is detected in the limited scale space searched and also in order to help maintain constant detection time. Performing this is easily done as we know for each DPM detection at which scale it occurred. In order to limit degradation in resolution we also limit image resizing factor to the range $(0.25 - 2)$.

All our experiments were conducted using a machine equipped with an Intel® CORE™ i7-2620M 2.7GHz processor and 8GB of RAM. All our code is Matlab® based with some mex implementations. The input frame rate was $f_{in} = 15fps$, typical runtime performance for our tracker and detector for this setup are as follows: The DPM detector runs at $\sim 2.5fps$ $(= f_{in}/6)$ for a search window of 150×150 pixels. Our LK tracker runs at $\sim 60fps$ $(= 4 \times f_{in})$, and when also performing validation we run at $\sim 30fps$ $(= 2 \times f_{in})$, both for a 100×50 pixel target.

Table 1 summarizes all the tracking methods evaluated in this work. We note that any parameter used by any algorithm was kept fixed in all experiments.

Table 1 Summary of tracking methods evaluated.

Method	Description
ASLA	Tracking via Adaptive Structural Local Sparse Appearance Model [23]
CT	Compressive Tracking [44]
DPM	Tracking based on DPM only
L1	Real-time Robust L1 Tracker [6]
L1-xα_{trk}-DPM-xα_{det}	L1-DPM fusion with $\alpha_{trk}, \alpha_{det}$ acceleration factors for tracking and detection
LK	Basic LK tracker [30]
LK-DPM-FIX	LK with DPM based template update at fixed intervals
LK-DPM-WO-CUP	LK at each frame with DPM using LK idle time
LK-DPM-CUP	As LK-DPM-WO-CUP only LK also used for “catch-up”
LK-DPM-VLD	LK with validation and DPM based template update
LOT	Locally orderless tracking [32]
SCM	Tracking via Sparsity-based Collaborative Model [45]
TLD	Tracking-Detection-Learning [24]

4.3 Experiment 1 - Tracking under severe view point changes is difficult

We benchmark several tracking methods on our new dataset. Among these methods are several state-of-the-art algorithms producing top tier results in a recently published tracking benchmark [42]. The algorithms are: Tracking-Detection-Learning (**TLD**) [24], Locally Orderless Tracking (**LOT**) [32], Real-time Robust L1 Tracker (**L1**) [6], Tracking via Sparsity-based Collaborative Model (**SCM**) [45], Compressive Tracking (**CT**) [44] and Tracking via Adaptive Structural Local Sparse Appearance Model (**ASLA**) [23].

The tracking performance of the benchmarked methods are presented in figure 1. Figure 1-(a) shows real-time tracking results, measured using the methodology outlined in section 4 (TLD results are missing from this graph as it is only available in an executable version and could not be adjusted for running with our evaluation methodology). It is clear that all methods perform rather poorly in these conditions, with all methods having AUC< 0.5. Also if we look at $overlap = 0.65$, which is a reasonable operating point for tracking, we observe that all methods produces a success rate < 0.4.

Since most of the provided implementations are academic codes, one may claim that they are not optimized and can be better implemented to meet the real-time constraint. To estimate the potential for improvement, we evaluated these algorithms *without* run-time constraints, i.e. assuming they run at real-time. These results are shown in figure 1-(b). Removing the real-time constraint significantly improves the tracking performance, yet still at an overlap of 0.65, the best methods provide a success rate of ~ 0.65 which is still not suited for any practical application.

4.4 Experiment 2 - Tracking-with-detection

We evaluate the performance of the proposed tracking-with detection systems. To do so we implement these systems using our DPM detector and LK-tracker. Results are presented in figure 2.

Tracking-by-detection discussed in section 3 is implemented using the DPM detector. We first evaluate tracking-by-detection *without* run-time constraints, i.e. performing detection at every frame, denoting it **DPM-NOT-RT**. We observe that, as expected, this specially designed view-point invariant detector is able to produce good results with AUC of 0.766 and a success rate > 0.9 for overlap threshold of 0.65. These results indicate that using a detector can indeed solve the accuracy problem. However once tracking-by-detection is evaluated under real-time constraints (denoted **DPM**), we see that accuracy decreases significantly. These results, along with the benchmark results, motivate us to perform tracking-with-detection incorporating a detector in the tracking system in an attempt to achieve more reliable tracking performance under real-time constraints.

Both template update with near-real-time tracker methods, presented in section 3, are implemented. The first, includes template update at fixed intervals, denoted **LK-DPM-FIX**. The second, where the template is update using the validation scheme is denoted **LK-DPM-VLD**. The template update latency of both methods is l_{det0} and the main difference between the two is the update interval. In order to make a fare comparison we measure the performance of **LK-DPM-FIX** with all update interval values between 2 frames and 10 frames ($K = \{2, 3, .., 10\}$) and present the best results (obtained for $K = 5$).

The two strategies of section 3 which use a fast tracker ($t_{trk} < \frac{1}{2}t_{in}$) are implemented next. These algorithms perform tracking at each frame then use t_{idle} for template update purposes. We denote by **LK-DPM-WO-CUP**, the strategy where the template is updated as soon as a detection is obtained. This strategy has an update latency of l_{det1}. The second option is not to update the template once it is obtained but rather use t_{idle} to perform 'catch-up'. This strategy is denoted **LK-DPM-CUP** and has an update latency of l_{det2}.

In addition to the above methods we also present the performance of the LK tracker on its own as well as the "best-tracker" curve showing the best performance achieved by any of the benchmark trackers tested with real-time constraint at each operating point independently (i.e. for each threshold we choose the tracker from figure 1-(a) that performed best).

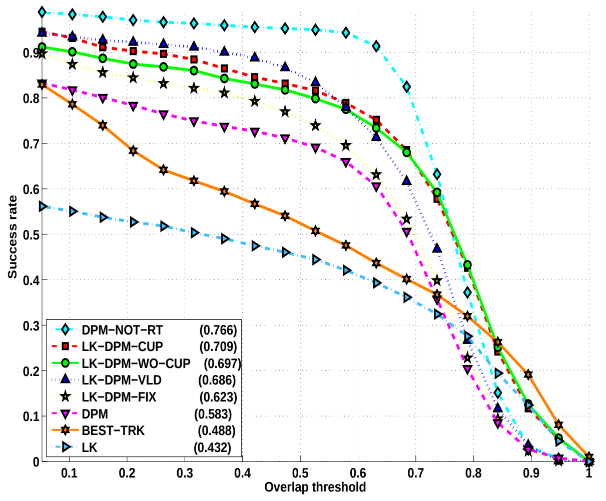
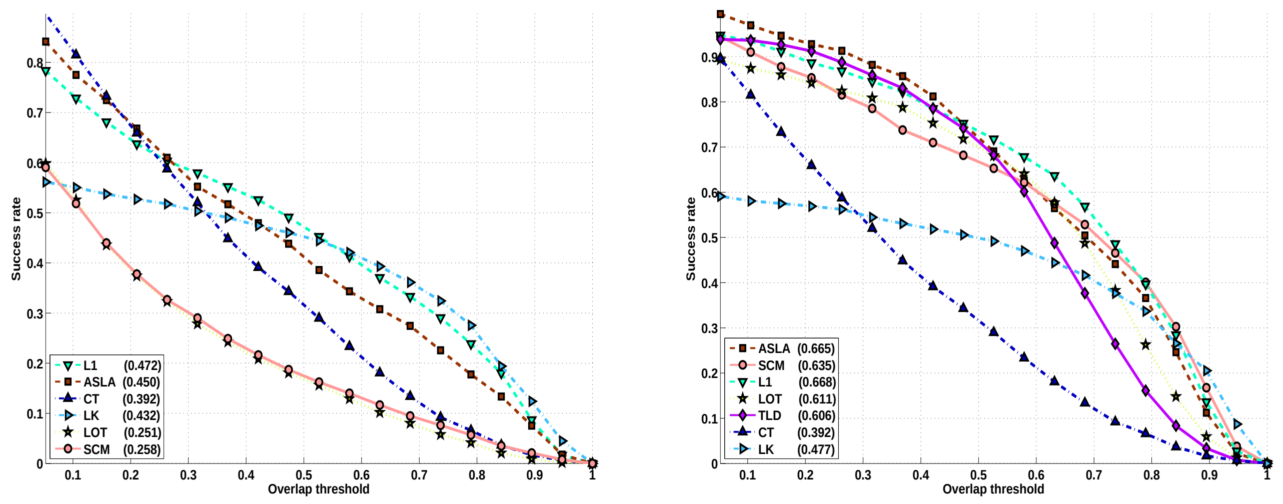


Fig. 2 Experiment 2 - Tracking-with-detection: LK-DPM methods dominate over LK and DPM alone which have low performance, the former due to lack of information, the latter due to lack of processing speed. Cyan curve shows performance of DPM tracker without any computational constraints, demonstrating potential performance. See text for more detail (best viewed in color).

Results presented in figure 2 show that the different template update strategies provide significant improvement of $\sim 40\%$ (multiplicative) in AUC over the best tracking method, and a $17\% - 21\%$ improvements over the DPM tracker. The results show the significance of template update and the synergy obtained by tracker-detector fusion, producing a system superior to its components. Moreover, the LK-DPM methods, evaluated under real-time constraint, have a $3\% - 6\%$ increase in AUC over the best tracking methods evaluated *without* real-time constraint. For $overlap = 0.65$ perfor-



(a) With real-time constraint

(b) Without real-time constraint

Fig. 1 Experiment 1 - Tracking is difficult: Benchmark of several state-of-the-art tracking methods on the proposed dataset both with (a) and without (b) real-time constraint. Graphs show the tracking success rate for overlap threshold $\in [0, 1]$. Numbers in the legend indicate the AUC for each method. The task is difficult due to frequent view-point changes, and this difficulty is stressed when methods are evaluated under a computational complexity constraint (best viewed in color).

mance jumps from 0.6 (**L1** *not* real-time) to ~ 0.75 for the LK-DPM methods. This performance gain can be attributed to the additional class prior information available for the LK-DPM methods.

We compared **LK-DPM-VLD** which uses the validation scheme to decide when to perform template update and **LK-DPM-FIX** which updates the template at fixed intervals. In the comparison, we notice that using the validation scheme outperforms fixed interval updates. This indicates that the template update rate is not uniformly distributed, which of course makes sense, as we expect the template to be updated more frequently when appearance changes occur, e.g. turning, relative to a simple tailing scenario.

4.5 Experiment 3 - Latency and accuracy

We simulate different detection and tracking operating points for two tracking-with-detection methods, **LK-DPM-WO-CUP** with latency l_{det1} and **LK-DPM-CUP** with latency l_{det2}. This is done by accelerating or decelerating our system assigning different values of 'acceleration factor' α as explained in section 4. Introducing α into latency equations (3) and (7) gives:

$$l_{det1} = \frac{\frac{t_{det}}{\alpha}}{t_{in} - \frac{t_{trk}}{\alpha}} \tag{8}$$

$$l_{det2} = \frac{\frac{t_{det}}{\alpha}}{t_{in} - 2\frac{t_{trk}}{\alpha}} \tag{9}$$

We measure the empirical average tracking and detection processing intervals t_{trk} and t_{det} when the algorithms run without acceleration (i.e. $\alpha = 1$). We plug in these values along with t_{in} into equations (8) and (9) with different values of α, ranging from 0.5 to 10, and obtain a prediction for the latency of each method at several operating points. We then run the tracking methods with these α values using our evaluation methodology and compare the predicted latency values with the latency values measured empirically.

Results are presented in figure 3 showing the predicted latency for different values of α as obtained from equations (8) and (9) along with the latency measured experimentally. The empirical latencies are highly correlated with the predicted ones, with $\rho = 0.996$ for **LK-DPM-CUP** and $\rho = 0.997$ for **LK-DPM-WO-CUP**, validating our latency analysis.

After seeing our latency predictions are correct we turn our attention to the relation between tracking accuracy and latency, plotting this data for the different α values, as presented in figure 4. We first note the trends showing an increase in accuracy for decreasing latency, for each method on its own. This trend is broken for **LK-DPM-CUP** at $\alpha > 2$ where the performance reaches that of the **DPM** detector. We observe the performance margin between the methods, meaning that for a given latency **LK-DPM-CUP** outperforms **LK-DPM-WO-CUP** (again this is true for $\alpha < 3$) . These results indicate that more accurate template update, achieved by performing 'catch-up', leads to better system accuracy compared to instantaneous appearance update. More generally we note that producing more accurate template update should be considered even at the expense of in-

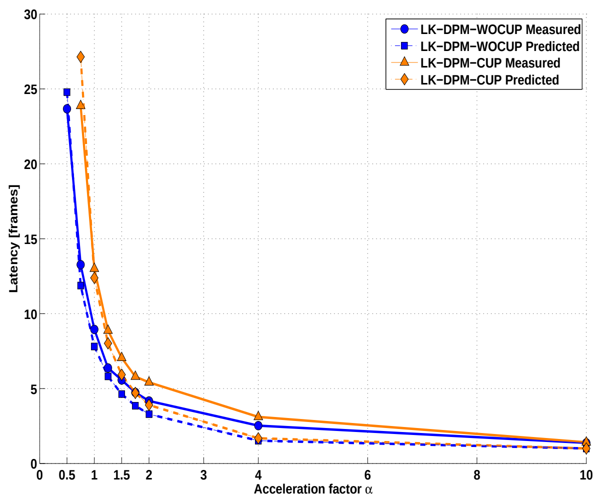


Fig. 3 Experiment 3 - Latency prediction: Measured and predicted latency value for two tracking-with-detection methods. Prediction and measurements are highly correlated ($\rho \leq 0.996$ for both methods), validating our latency analysis (best viewed in color).

creased latency, as this might improve overall performance.

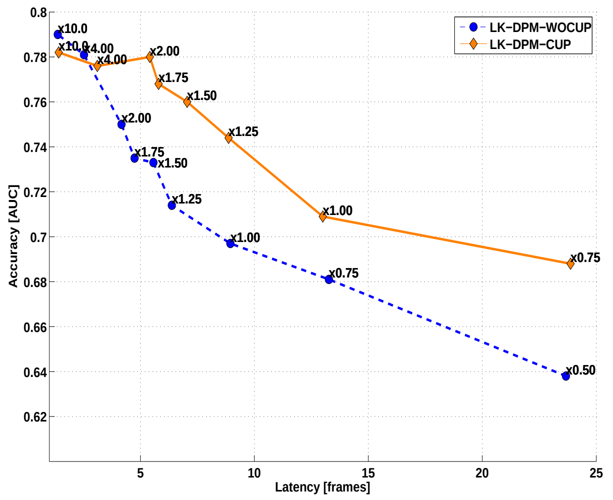


Fig. 4 Experiment 3 - Accuracy and latency: For each of the methods lower latency leads to higher accuracy. The performance margin between methods indicates that more accurate template update can result in better overall system performance even at the expense of increased latency. See text for more details (best viewed in color).

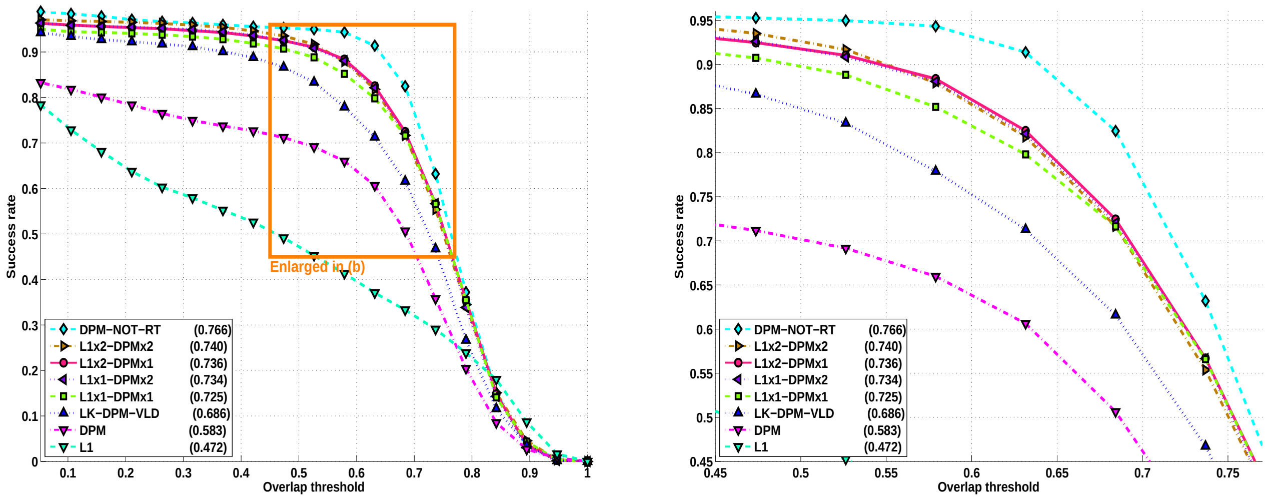
4.6 Experiment 4 - Real-time evaluation for better design

In the following experiment we take the **L1** tracker, which is the leading tracking algorithm under real time constraints, and combine it with our DPM detector using the validation scheme presented in section 3, we denote this system **L1-DPM**. We assign individual acceleration factors, $\alpha_{trk}, \alpha_{det}$, for tracking and detection respectively. Using these factors we emulate the following scenarios i) Accelerating only the tracker by $\times 2$ ($\alpha_{trk} = 2, \alpha_{det} = 1$) ii) Accelerating only the detector by $\times 2$ ($\alpha_{trk} = 1, \alpha_{det} = 2$) iii) Running the system on twice as powerful hardware ($\alpha_{trk} = 2, \alpha_{det} = 2$). We denote the different accelerated systems by **L1x**α_{trk}**-DPMx**α_{det} e.g. **L1x1-DPMx1** denotes the original system without any accelerations.

Results are presented in figure 5, on the left side, (a), is the full scale results and on the right, (b), we zoom in to the region $threshold \in [0.5, 0.8]$ which is the most interesting region for practical applications. We first observe that **L1-x1-DPM-x1** significantly outperforms both of its components (**L1** and **DPM**). In addition we observe that this system also outperforms **LK-DPM-VLD**, which can be attributed to the performance margin between **L1** and **LK** trackers. The results for accelerating the detector and tracker provide interesting design insights. We observe that the performance gained by accelerating the detector by a factor of $\times 2$ is equivalent to the expected performance gain obtained by accelerating the tracker by a factor of $\times 2$. Such a result is of high practical engineering interest as one can now decide where to direct research resources. Typically, accelerating a tracker is easier than accelerating a detector and, according to these results, is expected to produce the same performance gain. Moreover we can see that increasing the computational power by a factor of $\times 2$ is only marginally better than accelerating only the tracker, with improvement noticed mostly at low $threshold$ values, this again may suggest that for this system accelerating the tracker is the most cost-effective coarse of action for increasing performance.

5 Conclusions

Tracking, even rigid, targets undergoing severe view-point changes is a challenging task, made even more difficult under runtime constraints. Using our newly proposed dataset we have demonstrated that state-of-the-art tracking methods still struggle with tracking rigid targets undergoing large view point changes both with and without runtime constraints. When the target class is known, using a class detector is key for making a successful tracking system. Using a carefully trained classifier can boost performance significantly, transferring a system from 'almost always fail' to 'usually succeed'.



(a) Full scale (b) Zoom in

Fig. 5 Experiment 4 - Real-time evaluation as system design tool: We examine **L1-DPM** when accelerating only the tracker, only the detector or both. Results indicate that the performance gained by tracker acceleration is similar to detector or full system acceleration providing invaluable insight into where optimization efforts should be focused (best viewed in color).

When working under runtime constraints wise fusion of a detector into a tracking system is advised and is expected to deliver superior performance compared to plain fusion techniques. Advanced fusion schemes, such as the 'validation' or 'catch-up' schemes, proposed in this work, were demonstrated to out perform simpler fusion techniques such as simply running the detector on every possible frame ('DPM') of updating the template at fixed intervals using the detector ('LK-DPM-FIX').

Finally, when practical real-time tracking systems are designed one cannot evaluate tracking accuracy alone but rather consider accuracy subject to runtime constraints. In this context our newly proposed evaluation methodology will give a better estimate of actual system performance, enable better understanding of its latency and allow an insight into the sensitivity of SW/HW acceleration.

References

1. Thirteenth ieee international workshop on performance evaluation of tracking and surveillance (pets) (2010)
2. Andriluka, M., Roth, S., Schiele, B.: People-tracking-by-detection and people-detection-by-tracking. CVPR (2008)
3. Avidan, S.: Support vector tracking. PAMI (2004)
4. Avidan, S.: Ensemble tracking. CVPR (2005)
5. Babenko, B., Yang, M., Belongie, S.: Visual tracking with online multiple instance learning. CVPR (2009)
6. Bao, C., Wu, Y., Ling, H., Ji, H.: Real time robust l1 tracker using accelerated proximal gradient approach. CVPR (2012)
7. Bar-Hillel, A., Levi, D., Krupka, E., Goldberg, C.: Part-based feature synthesis for human detection. ECCV (2010)
8. Benenson, R., Mathias, M., Timofte, R., Van Gool, L.: Pedestrian detection at 100 frames per second. CVPR (2012)
9. Bernardin, K., Stiefelhagen, R.: Evaluating multiple object tracking performance: the clear mot metrics. EURASIP Journal on Image and Video Processing (2008)
10. Cootes, T., Edwards, G., Taylor, C.: Active appearance models. TPAMI (2001)
11. Dollr, P., Belongie, S., Perona, P.: The fastest pedestrian detector in the west. BMVC (2010)
12. Ess, A., Leibe, B., Schindler, K., Van-Gool, L.: Robust multi-person tracking from a mobile platform. TPAMI (2009)
13. Everingham, M., Van Gool, L., Williams, C., Winn, J., Zisserman, A.: The PASCAL Visual Object Classes Challenge 2011 (VOC2011) Results. http://www.pascal-network.org/challenges/VOC/voc2011/workshop/index.html
14. Everingham, M., Van-Gool, L., Williams, C., Winn, J., Zisserman, A.: The pascal visual object classes (voc) challenge. IJCV (2010)
15. Fan, J., Shen, X., Wu, Y.: What are we tracking: a unified approach of tracking and recognition. IEEE Transactions on Image Processing **22**(2), 549–560 (2013)
16. Felzenszwalb, P., Girshick, R., McAllester, D.: Discriminatively trained deformable part models, release 4. http://people.cs.uchicago.edu/ pff/latent-release4/
17. Felzenszwalb, P., Girshick, R., McAllester, D.: Cascade object detection with deformable part models. CVPR (2010)
18. Felzenszwalb, P., Girshick, R., McAllester, D., Ramanan, D.: Object detection with discriminatively trained part based models. TPAMI (2010)
19. Felzenszwalb, P., Huttenlocher, D.: Pictorial structures for object recognition. IJCV **61**(1) (2005)
20. Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? the kitti vision benchmark suite. In: CVPR (2012)
21. Grabner, H., Grabner, M., Bischof, H.: Real-time tracking via on-line boosting. BMVC (2006)
22. Hager, G., Belhumeur, P.: Efficient region tracking with parametric models of geometry and illumination. TPAMI (1998)
23. Jia, X., Lu, H., Yang, M.: Visual tracking via adaptive structural local sparse appearance model. CVPR (2012)
24. Kalal, Z., Mikolajczyk, K., Matas, J.: Tracking-learning-detection. TPAMI (2010)
25. Kasturi, R., Goldgof, D., Manohar, S., Garofolo, J., Bowers, R., Boonstra, M., Korzhova, V., Zhang, J.: Framework for perfor-

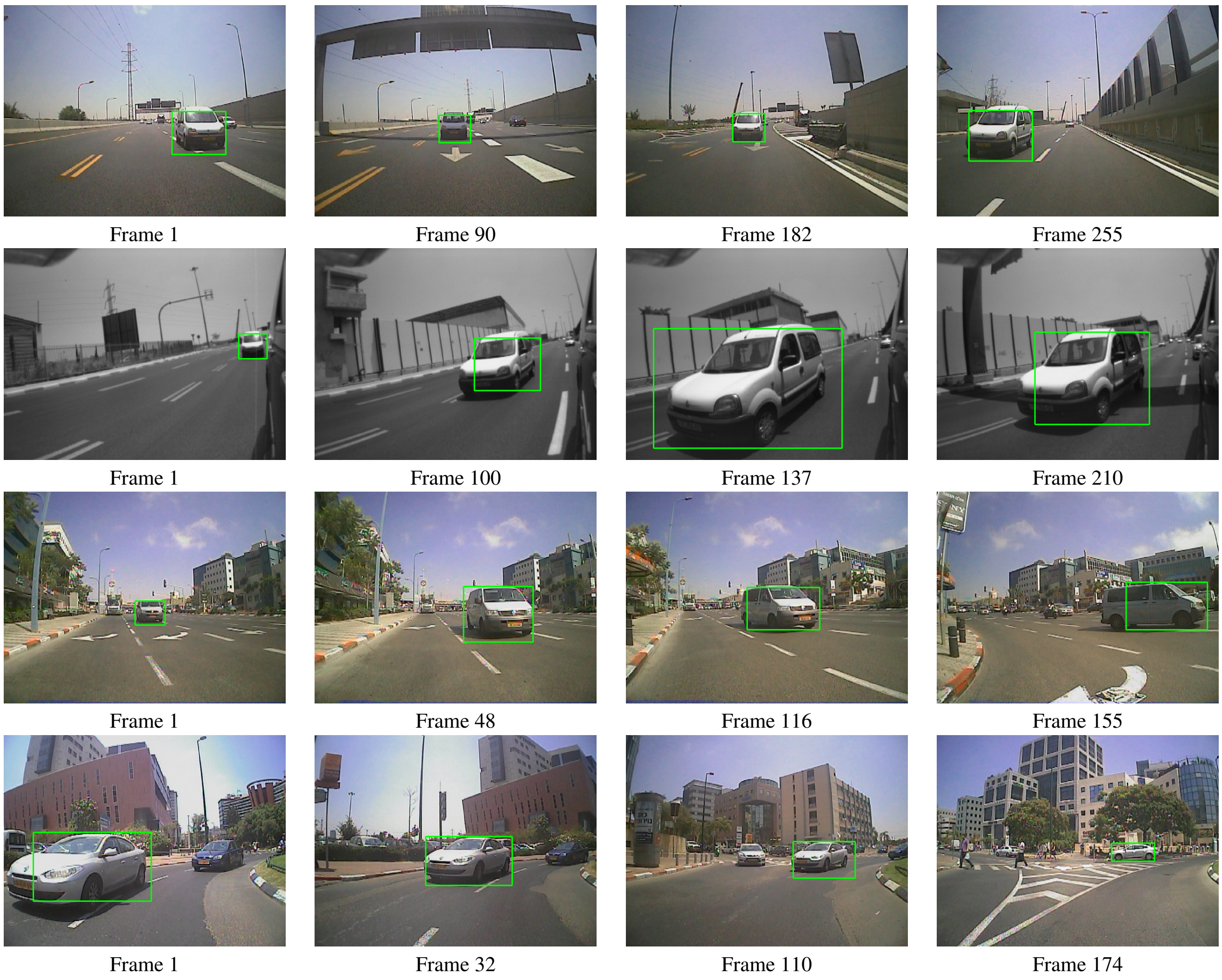


Fig. 6 Sample frames for several sequences. Tracking results for **LK-DPM-CUP** marked in Green. (best viewed in color)

mance evaluation of face, text, and vehicle detection and tracking in video: data, metrics, and protocol. PAMI (2009)
26. Kwon, J., Lee, K.: Visual tracking decomposition. CVPR (2010)
27. Leibe, B., Schindler, K., Cornelis, N., Van-Gool, L.: Coupled object detection and tracking from static cameras and moving vehicles. TPAMI (2008)
28. Leichter, I., Krupka, E.: Monotonicity and error type differentiability in performance measures for target detection and tracking in video. In: CVPR (2012)
29. Lucas, B., Kanade, T.: An iterative image registration technique with an application to stereo vision. In: Proccedings of Imageing Understanding Workshop (1981)
30. Matthews, I., Baker, S.: Lucas-kanade 20 years on: A unifying framework. IJCV (2004)
31. Matthews, I., Ishikawa, T., Baker, S.: The template update problem. TPAMI (2004)
32. Oron, S., Bar-Hillel, A., Levi, D., Avidan, S.: Locally orderless tracking. CVPR (2012)
33. Panin, G., Klose, S., Knoll, A.: Real-time articulated hand detection and pose estimation. Advances in Visual Computing (2009)
34. Papanikolopoulos, N., Khosla, P., Kanade, T.: Visual tracking of a moving target by a camera mounted on a robot: a combination of control and vision. IEEE transactions on robotics and automation (1993)
35. Ross, D., Lim, J., Lin, R., Yang, M.: Incremental learning for robust visual tracking. IJCV (2007)
36. Santner, J., Leistner, C., Saffari, A., Pock, T., Bischof, H.: Prost:parallel robust online simple tracking. CVPR (2010)
37. Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., Blake, A.: Real-time human pose recognition in parts from a single depth image. CVPR (2011)
38. Siebel, N., Maybank, S.: Fusion of multiple tracking algorithms for robust people tracking. ECCV (2002)
39. Stalder, S., Grabner, H., van Gool, L.: Beyond semi-supervised tracking: Tracking should be as simple as detection, but not simpler than recognition. ICCV workshops (2009)
40. Stauffer, C., Grimson, E.: Learning patterns of activity using real-time tracking. PAMI (2000)
41. Williams, O., Blake, A., Cipolla, R.: Sparse bayesian learning for efficient visual tracking. TPAMI (2005)
42. Wu, Y., Lim, J., Yang, M.: Online object tracking: A benchmark. In: CVPR (2005)
43. Yilmaz, A., Javed, O., Shah, M.: Object tracking: A survey. ACM. Comp. Survey **38(4)** (2006)
44. Zhang, K., Zhang, L., Yang, M.: Real-time compressive tracking. ECCV (2012)
45. Zhong, W., Lu, H., Yang, M.: Robust object tracking via sparsity-based collaborative model. CVPR (2012)

46. Zhu, X., Ramanan, D.: Face detection, pose estimation, and landmark localization in the wild. CVPR (2012)