Speeding Up Tracking by Ignoring Features

Lu Zhang Hamdi Dibeklioğlu Laurens van der Maaten
Pattern Recognition & Bioinformatics Group, Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands
{lu.zhang, h.dibeklioglu, l.j.p.vandermaaten}@tudelft.nl

Abstract

Most modern object trackers combine a motion prior with sliding-window detection, using binary classifiers that predict the presence of the target object based on histogram features. Although the accuracy of such trackers is generally very good, they are often impractical because of their high computational requirements. To resolve this problem, the paper presents a new approach that limits the computational costs of trackers by ignoring features in image regions that — after inspecting a few features — are unlikely to contain the target object. To this end, we derive an upper bound on the probability that a location is most likely to contain the target object, and we ignore (features in) locations for which this upper bound is small. We demonstrate the effectiveness of our new approach in experiments with model-free and model-based trackers that use linear models in combination with HOG features. The results of our experiments demonstrate that our approach allows us to reduce the average number of inspected features by up to 90% *without affecting the accuracy of the tracker.*

1. Introduction

The tracking of objects in an image sequence is a seminal problem in computer vision, on which substantial progress has been made in recent years. Trackers generally constitute a combination of a motion prior and an object appearance model. Many modern trackers use a tracking-by-detection framework [11], in which the object appearance model comprises a binary classifier that predicts target object presence or absence. The appearance model may be trained in an offline manner for model-based trackers or in an online manner for model-free trackers. The location of the target object is determined by evaluating the classifier on the contents of a sliding window over the image (at multiple scales) and finding the location where its response is highest. The main drawback of such a sliding-window approach is that it is computationally expensive, in particular, when the feature dimensionality is high or when the search space (*i.e.* the number of locations and scales) is large. To speed up sliding-window trackers, studies have proposed, *e.g.*, to use Haar features that can be computed efficiently via integral images [11], to use GPU implementations [7], and to exploit the fact that part of the computations may be shared between nearby locations [13].

This paper exploits another property of many modern trackers to speed up sliding-window search: *viz.* the fact that classifier scores often decompose into a sum over features, *e.g.*, in linear models. Specifically, we may already be able to infer that the target object is not present at a location after considering only a small subset of features. For instance, if the classifier subscore of a particular location is a large negative number after considering 10% of the features whilst other locations have a large positive subscore, the chance that the highest classifier response will be attained at that location after considering all features is very small (in particular, if we ordered the features according to their "importance"). In fact, we can derive exact upper and lower bounds on the final classifier score for that location if the feature values are bounded, as is the case for many popular features including Haar, HOG, and LBP features. Such bounds (or probabilistic versions thereof) allow us to discard image locations without considering all features at those locations. To the best of our knowledge, no prior work has exploited this to speed up sliding-window tracking.

The paper presents a new algorithm, called *feature ignoring tracking* (FIT), that speeds up sliding-window trackers using an upper bound on the probability that an image location —based on the features already processed— is the location that is most likely to contain the target object. We demonstrate the effectiveness of the new approach in experiments with model-free and model-based trackers, focusing on trackers of which the object appearance model is a linear model that is trained on HOG features. (It should be noted that the FIT algorithm is applicable to a much larger family of trackers.) The results of our experiments reveal that the FIT tracking algorithm may reduce the average number of inspected features by up to 90% without affecting the accuracy of the tracker.

2. Related Work

In tracking, there are two common approaches to search for the most likely location of the target object, *viz.* sliding-window search and particle filtering [10]. Particle filters use a set of particles to approximate the posterior over object locations, and are generally very fast because the number of particles can be set by the user based on the available computational budget. However, particle filters are also inaccurate, which may cause the tracker to drift, in particular, when the target object has been temporarily occluded. To circumvent this problem, most modern trackers are based on sliding-window search [3, 6, 17, 21]. The drawback of sliding-window search is that it may be slow, in particular, when the search space is large, *i.e.* when there are many locations and scales that need to be considered. Various strategies have been proposed to speed up sliding-window search, of which we review the most prominent ones below.

Specifically, [13] introduces a branch-and-bound search strategy that exploits the fact that computations can be shared between overlapping bounding boxes (*i.e.* target locations). The branch-and-bound approach can be extended to localize objects with non-rectangular shapes [16]. Whilst such branch-and-bound approaches are computationally very efficient, they rely on the assumption that the location of a feature within a bounding box is irrelevant for the classifier score. This assumption holds for the bag-of-visual-word features employed in [13], but it does not hold for features that are often used in tracking such as histograms of oriented gradients (HOG) [3], locally binary patterns (LBP) [14], and Haar features [17]. By contrast, our FIT algorithm does not assume that the effect of a feature on the classifier score is independent of its location in the bounding box under consideration. As a result, FIT is more generally applicable than branch-and-bound approaches.

Another approach to speeding up sliding-window search is to use a cascaded approach in which the evaluation of weak classifiers is terminated as soon as one of the weak classifiers outputs a negative decision [17]. Whilst such an approach is very effective, it can only be used with features that can be computed very efficiently, such as Haar features. By contrast, the FIT algorithm can be used with all types of features that can be bounded above and below.

Next to work on speeding up sliding-window search, several recent studies consider object detection on a budget (although such approaches have not yet been used for tracking). Specifically, [12] learns a policy for multi-class detection tasks that decides which detector to deploy next. In [20], feature extraction costs during training are considered to learn detectors that jointly maximize accuracy and minimize CPU usage at runtime. Whilst it may be possible to adapt FIT to such settings, we do not aim to perform tracking under a strict computational budget in this paper; we leave such extensions of FIT to future work.

3. Ignoring Features in Tracking

Sliding-window trackers compute a score $s(\cdot)$ for a large number of bounding boxes $B \in \mathcal{B}$ that measures the likelihood of the target object being present at the corresponding location in the image, and return the bounding box with the highest score as the location of the target object. A wide range of popular trackers have two main properties that we can exploit to speed up sliding-window search: (1) the score they compute is defined as the sum of a bias b and the inner product between the object model $\mathbf{w}$ and the features $\mathbf{x}^{(B)} = \phi(\mathbf{I}, B)$ extracted from bounding box B: $s(B; \mathbf{I}, \mathbf{w}, b) = \mathbf{w}^\top \mathbf{x}^{(B)} + b$; and (2) the individual feature values can be upper and lower bounded, *i.e.* we can find values l and u such that $\forall d, B : l \leq x_d^{(B)} \leq u$. It is straightforward to find such lower and upper bounds for popular features including HOG features [3], LBPs [14], and Haar features [17]. Popular trackers that have both of the aforementioned properties include [1, 4, 8, 9, 11, 21].

Our *feature ignoring tracker* (FIT) exploits the two properties introduced above to find the highest scoring bounding box $B^* = \operatorname{argmax}_B s(B; \mathbf{I}, \mathbf{w}, b)$ without computing the full score for most of the bounding boxes. In particular, FIT discards bounding boxes that have a small chance of attaining the highest score based on a small part of the inner product $\mathbf{w}^\top \mathbf{x}$, *i.e.* after only a small subset of the features is considered. FIT does this by (1) upper bounding the probability that a bounding box can attain the highest score, considering the part of the inner product currently computed and (2) discarding bounding boxes for which this probability is below some threshold θ. As a result, the threshold θ allows one to trade-off speed for accuracy: if θ is set to zero, FIT discards a small number of bounding boxes but is guaranteed to find the highest-scoring location, whereas higher values of θ allow FIT to discard the majority of bounding boxes after considering only a few features at the cost of a small loss in tracking accuracy. Such a trade-off parameter is very useful, *e.g.*, when the same tracker is deployed on platforms with different computational budgets.

FIT sorts the elements of the object appearance model $\mathbf{w}$ in such a way that features $x_d^{(B)}$ with the highest absolute weight w_d are considered first[1] (the sorting has to be performed only once, so does not affect the computational complexity of the tracker). Next, FIT computes the subscore for all possible bounding boxes based on the first d features, *i.e.* it computes $s_d(B; \mathbf{I}, \mathbf{w}_{1:d}, b) = b + \mathbf{w}_{1:d}^\top \mathbf{x}_{1:d}^{(B)}$, where we use the notation $\mathbf{x}_{a:b}$ to represent elements a through b of vector $\mathbf{x}$. We select a first candidate location by picking the bounding box B_d^* with the highest subscore, and compute the full score for this bounding box. To determine whether or not we still need to consider more features for an

[1] For simplicity, we assume that all features have the same "scale". Features may need to be normalized first if the scale of the features varies.

arbitrary other bounding box B, we would like to compute the probability that bounding box B can still attain a higher score than the candidate bounding box B_d^*:

$$p(s(B;\mathbf{I},\mathbf{w},b) \geq s(B_d^*;\mathbf{I},\mathbf{w},b)). \quad (1)$$

Whilst this probability cannot be computed without knowledge of the feature distribution $p(\mathbf{x})$, it is possible to upper bound the probability that bounding box B will obtain a higher score than the first candidate B_d^* using an upper bound that resembles a Chernoff bound:

$$\begin{aligned} &p(s(B;\mathbf{I},\mathbf{w},b) \geq s(B_d^*;\mathbf{I},\mathbf{w},b)) \\ &\quad = p(\exp(s(B;\mathbf{I},\mathbf{w},b)) \geq \exp(s(B_d^*;\mathbf{I},\mathbf{w},b))) \\ &\quad \leq \frac{\mathbb{E}\left[\exp(s(B;\mathbf{I},\mathbf{w},b))\right]}{\exp(s(B_d^*;\mathbf{I},\mathbf{w},b))} \\ &\quad \leq \max_{\mathbf{w}_{d+1:D}^\top \mathbf{x}_{d+1:D}^{(B)}} \frac{\exp(s(B;\mathbf{I},\mathbf{w},b))}{\exp(s(B_d^*;\mathbf{I},\mathbf{w},b))}, \end{aligned} \quad (2)$$

where the expectation is over the part of the bounding box score that has not yet been computed, *i.e.* over $p\left(\mathbf{w}_{d+1:D}^\top \mathbf{x}_{d+1:D}^{(B)}\right)$. The first inequality follows directly from Markov's inequality because $\exp(s(B;\mathbf{I},\mathbf{w},b))$ is guaranteed to be non-negative. The second inequality follows from the fact that: $\max_x f(x) \geq \mathbb{E}[f(x)]_{p(x)}$.

Importantly, the upper bound in Eqn. 2 can be computed very efficiently because the elements in $\mathbf{w}$ are sorted and we have an upper and lower bound on the features, u and l. Specifically, Eqn. 2 can be computed by noting that:

$$\begin{aligned} &\max_{\mathbf{w}_{d+1:D}^\top \mathbf{x}_{d+1:D}^{(B)}} \frac{\exp(s(B;\mathbf{I},\mathbf{w},b))}{\exp(s(B_d^*;\mathbf{I},\mathbf{w},b))} \\ &\qquad = \frac{\exp(u\sum_{i:\mathbf{w}_{d+1:D}>0} w_i + l\sum_{i:\mathbf{w}_{d+1:D}<0} w_i)}{\exp\left(s(B_d^*;\mathbf{I},\mathbf{w},b) - s_d(B;\mathbf{I},\mathbf{w}_{1:d},b)\right)}. \end{aligned}$$

Note that the sums over the positive and negative elements of $\mathbf{w}$ in the numerator can be computed offline and stored for every value of d, and that all the terms in the denominator were already pre-computed. Therefore, the upper bound for bounding box B in Eqn. 2 can be computed in $\mathcal{O}(1)$.

FIT proceeds by comparing the upper bound in Eqn. 2 with a user-specified threshold θ, and discarding all bounding boxes for which the upper bound is sufficiently small. Next, the subscores for the remaining bounding boxes are updated by including a new subset of features, the actual score is computed for the most promising location based on the updated subscores, and bounding boxes for which Eqn. 2 is smaller than threshold θ are removed from the remaining candidate boxes. This process is iterated until only one candidate bounding box remains or until all features have been included in the scores for (a small subset) of the bounding boxes. Pseudocode for our FIT algorithm is presented in Algorithm 1.

Algorithm 1 Feature-Ignoring Tracker

Require: image $\mathbf{I}$
Require: scoring function $s(B;\mathbf{I},\mathbf{w},b)$ with weights $\mathbf{w}$ sorted based on absolute value (in descending order)
Require: upper bound u and lower bound l on features
Require: speed-accuracy trade-off parameter θ
Require: set of candidate bounding boxes $\mathcal{B}$
Require: number of features δd to add at each iteration
Ensure: $B^* = \text{argmax}_{B\in\mathcal{B}}\, s(B;\cdot)$ with high probability

```
d ← 0
while |B| > 1 and d ≤ D do
  d ← d + δd
  for ∀B ∈ B do
    compute subscore s_d(B; I, w_{1:d}, b)
  end for
  find candidate B*_d = arg max_{B∈B} s_d(B; I, w_{1:d}, b)
  compute score s(B*_d; I, w, b) for candidate box B*_d
  for ∀B ∈ B do
    if max_{w^⊤_{d+1:D} x^{(B)}_{d+1:D}} exp(s(B;I,w,b)) / exp(s(B*_d;I,w,b)) < θ then
      remove B from B
    end if
  end for
end while
B* ← B*_d
```

4. Experiment 1: Model-free Tracking

Model-free tracking refers to the scenario in which the tracker receives a single bounding-box annotation of the target object, and in which an appearance model for the target object is learned in an online manner. We use an adapted version of the recently introduced SPOT tracker[2] [21] as the basis for our experiments. The SPOT tracker is essentially a Felzenszwalb detector [6] that is trained online; it uses HOG features to represent image patches and a linear SVM to predict target object presence. We adapted the SPOT tracker to (1) exclude part descriptors, *i.e.* we did not use pictorial structures; and (2) extract HOG features at two different scales simultaneously to increase the quality of the features. (Specifically, we used HOG cells of size 8×8 and 4×4 pixels.) We refer to the resulting tracker, which serves as the baseline for our experiments, as the SPOT* tracker. It measures the score of a bounding box B as:

$$s(B;\mathbf{I},\mathbf{w},b) = \mathbf{w}^\top \mathbf{x}^{(B)} - \frac{1}{2}\lambda \left\|\mathbf{z}^{(B)} - \mathbf{z}\right\|^2, \quad (3)$$

where the bias term $b = -\frac{1}{2}\lambda \left\|\mathbf{z}^{(B)} - \tilde{\mathbf{z}}\right\|^2$ corresponds to a simple Gaussian motion prior of the tracker: $\mathbf{z}^{(B)}$ denotes the location (x, y) of the center of bounding box B, whereas

[2]The source code of the SPOT tracker can be obtained from `http://visionlab.tudelft.nl/spot`.

$\mathbf{z}$ represents the location of the center of the tracked bounding box in the previous frame. The parameter λ is a trade-off parameter between the object appearance score and the motion prior score. Depending on the number of HOG cells that the target object comprises, the dimensionality of the features in the SPOT* tracker may be up to $D = 11,800$.

To update the object appearance model, SPOT* uses the tracked object in the previous frame as a positive example, and image regions with high appearance score that were not selected by the tracker as negative examples. The SPOT* tracker minimizes the hinge loss on these positive and negative examples by performing online updates using the passive-aggressive algorithm [2]. For more details on the online-learning algorithm in SPOT*, we refer to [21].

Speeding up SPOT* with FIT. Most computation time in the SPOT* tracker is spent on (1) computing the high-dimensional HOG features at every image location and (2) computing the inner product of the object appearance model with those HOG features at every image location. Because HOG features are normalized histograms, it is not possible to compute a subset of the features; this would not save much computation anyway, because the binning operation is relatively cheap once the gradient magnitudes and orientations are computed. Therefore, we focus on speeding up the computation of the HOG features with the object appearance model $\mathbf{w}$ at every image location, which is relatively expensive because multiplications are expensive operations. To use FIT in the SPOT* tracker, we need to define the upper and lower bound of the HOG features $\mathbf{x}$. These bounds can be derived based on how HOG features are computed.: (1) an unnormalized gradient histogram is computed, (2) the histogram is normalized, (3) all values are clipped at 0.2, and (4) the histograms are then averaged over the four neighboring cells and multiplied by 2. As a result, the lower bound of HOG features is 0 and the upper bound is 0.4.

4.1. Experimental Setup

We performed experiments on a publicly available collection of nine videos [1]. The videos contain a wide range of objects that are subject to sudden movements and (out-of-plane) rotations, and have cluttered, dynamic backgrounds. The videos have an average length of 556 frames. Each video contains a single object to be tracked, which is indicated by a bounding box in the first frame of the video. (First-frame annotations for all movies are shown in [1].)

We evaluate the performance of the trackers by measuring (1) *average location error* (ALE): the average distance of the center of the identified bounding box to the center of the ground-truth bounding box; and (2) *correct detection rate* (CDR): the percentage of frames for which the overlap between the identified bounding box and the ground truth bounding box is at least 50 percent. We compare the performance of our FIT-SPOT* tracker with that of the SPOT* tracker, which performs a full sliding-window search. As additional baselines, we also compare the performance FIT-SPOT* with those of the TLD [11] and Struck [9] trackers. In all experiments, we set the trade-off parameter λ in Eqn. 3 to $\frac{1}{2}\sigma^2$ with $\sigma^2 = 40$ pixels, and we set the number of features we add at each iteration of FIT, δd, to 10% of the total number of features. Code to reproduce the results of our experiments is available on `http://bit.ly/1hjOWHM`.

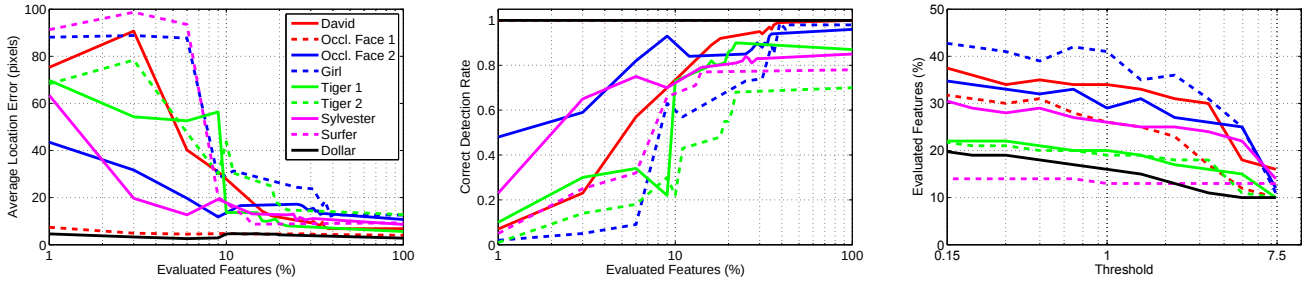
4.2. Results

In Figure 1, we present the results of our experiments with FIT-SPOT* on all nine videos. The different feature percentages were obtained by varying the threshold θ. The situation in which 100% of the features are used corresponds to the baseline SPOT* tracker. The figure presents the average location error of FIT-SPOT* as a function of the percentage of features considered by the tracker for each of the nine videos (1a); the correct detection rate as a function of the percentage of features evaluated by the tracker (1b); and the percentage of features evaluated as a function of θ (1c). To obtain results for feature percentages lower than 10%, we lowered the value of δd accordingly.

The results presented in Figure 1 show that on most videos, good results may be obtained even when only 10% of the features is used, whilst considering 20% of the features suffices to obtain a performance that is on par with that of a tracker that considers all features. In other words, FIT speeds up the most computationally costly part of a tracker such as SPOT*, *viz.* the computation of the responses of the object appearance model, by at least five times without a loss in performance. Figure 1(c) highlights the effect of varying the trade-off parameter θ: the percentage of features evaluated is decreasing as θ increases, but tends to be stable when θ is smaller than 1. We also measured the runtime of FIT-SPOT* in a Matlab/C++ implementation. Figure 2 presents the results of these experiments: it shows the tracking performance of of FIT-SPOT* as a function of its relative speed (compared to SPOT*). The results show that FIT-SPOT* is two to three times faster than SPOT* whilst achieving the same accuracy.

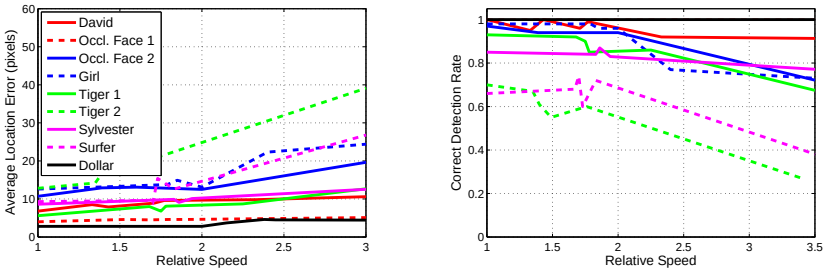
In Table 1, we compare the performance of the FIT-SPOT* tracker (using $\theta = 1$) with that of SPOT*, TLD [11], and Struck [9]. The results in the table show that —even though it ignores most of the features— the FIT-SPOT* tracker performs on par with state-of-the-art trackers such as Struck and TLD. Since FIT-SPOT* may ignore some confusing features, it occasionally outperforms SPOT*.

In Figure 4, we explore the average percentage of features evaluated in each frame for four different videos: *Occl. Face 1*, *Girl*, *Tiger 1*, and *David*. The results were obtained using a threshold value of $\theta = 1$. The results



(a) (b) (c)

Figure 1. Performance of the FIT-SPOT* model-free tracker on all nine videos. (a) The average distance error by FIT-SPOT* as a function of the percentage of features evaluated (lower is better). (b) Correct detection rate as a function of the percentage of features evaluated by FIT-SPOT* (higher is better). (c) The percentage of features evaluated by FIT-SPOT* as a function of threshold θ. Note that the evaluation of all features (100%) corresponds to the baseline SPOT* tracker, and that the x-axis of the plots uses a logarithmic scale.



(a) (b)

Figure 2. Relative speed of FIT-SPOT* compared to SPOT* (the baseline SPOT* tracker runs at speed 1). The results are obtained by varying the trade-off parameter θ. (a) The average distance error of FIT-SPOT* as a function of relative speed. (b) Correct detection rate of FIT-SPOT* as a function of relative speed.

presented in Figure 4 provide insight into how FIT works: specifically, FIT ignores more features when the target object is subject to small appearance changes, whilst it uses more features when there are large changes in object appearance due to rotations or occlusions of the target object.

Figure 3 illustrates the convergence of FIT-SPOT* on a single frame of the *David* sequence: it shows the subscores obtained after one through four iterations of FIT. The subscores show that after considering 30% of the features, there is hardly any uncertainty about the location of the target object. In practice, we may use even fewer features: after considering 10% of the features, there is clear maximum in the subscores near the location of the target object.

5. Experiment 2: Model-based Tracking

Model-based tracking uses detectors that are trained offline to recognize particular poses/views of a single object class. To evaluate the efficiency of FIT in model-based tracking, we implemented a standard baseline tracker. Our baseline model-based tracker (MBT) combines an isotropic Gaussian motion prior with the root detectors of the models presented in [6]. These root detectors are linear SVMs that operate on extended 32-dimensional HOG features, and that are trained on annotated subsets of the Pascal VOC data [6].

Table 1. Performance of four model-free trackers on nine videos measured in terms of: (1) average location error (lower is better) and (2) correct detection rate (higher is better). To measure the correct detection rate, a detection is counted as correct if the overlap between the identified bounding box and the ground truth bounding box is at least 50%. The best performance on each video is boldfaced.

	Struck [9]		**TLD** [11]		**SPOT*** [21]		**FIT-SPOT***	
	ALE	*CDR*	*ALE*	*CDR*	*ALE*	*CDR*	*ALE*	*CDR*
Sylvester	9.3	0.86	20.0	**0.91**	**8.6**	0.85	8.7	0.83
David	8.2	0.96	**4.5**	**1.00**	6.8	**1.00**	4.9	**1.00**
Occl. Face 1	7.5	**1.00**	16.8	0.99	**4.0**	**1.00**	4.4	**1.00**
Occl. Face 2	**6.3**	**0.98**	22.1	0.77	10.7	0.97	10.2	0.96
Surfer	**6.9**	0.83	7.9	**0.84**	9.3	0.66	9.2	0.78
Tiger 1	7.2	0.82	28.7	0.13	**5.6**	**0.93**	6.9	0.87
Tiger 2	**11.0**	0.57	37.5	0.27	12.8	0.70	**9.2**	**0.75**
Dollar	14.4	**1.00**	14.8	0.95	2.8	**1.00**	**2.6**	**1.00**
Girl	**10.8**	**1.00**	24.7	0.78	12.6	0.98	12.0	0.97

To track an object of the object class under consideration, a detection score $s(B; \mathbf{I}, \mathbf{w}, b)$ for each bounding box $B \in \mathcal{B}$ in the image $\mathbf{I}$ is computed using Eqn. 3 for all mixture components in the model [6]. The bounding box with the highest score is selected as the detection of the target object. In order to deal with scale changes, the MBT tracker explores two adjacent scales (a coarser and a finer

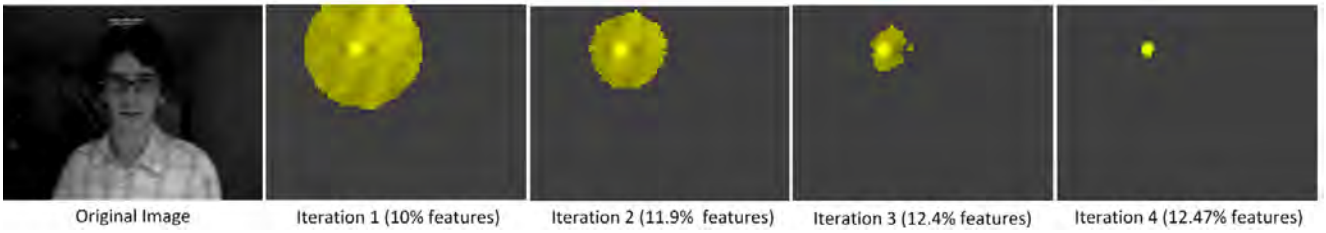


Figure 3. Search space left after one through four iterations of the FIT algorithm. Black regions indicate locations that were removed by FIT, whereas the intensity of non-black regions indicates the subscore of evaluated features at these locations (left-upper corner of each bounding box). The average evaluated features in total are given below each image.

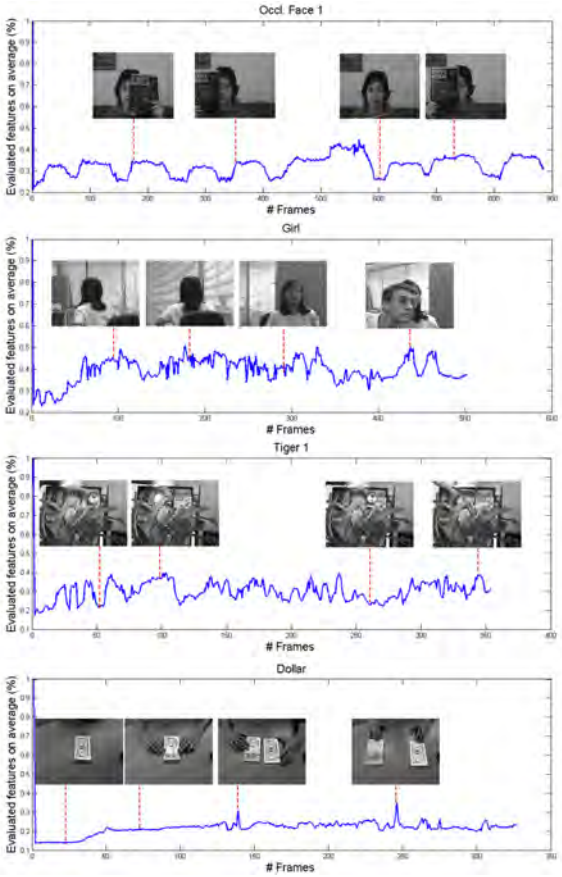


Figure 4. Average percentage of the evaluated features in each frame for videos *Occl. Face 1*, *Girl*, *Tiger 1*, and *David*.

one) in addition to the scale of the target object in the previous frame. Two mixture components (which correspond to different viewpoints) are used to track pedestrians, whereas six components are used for car tracking. Since we have lower and upper bounds on the HOG features, FIT can be used in MBT as before.

5.1. Experimental Setup

We gathered a set of pedestrian and car videos for our experiments. To evaluate the efficiency of FIT in model-based tracking, we manually selected segments from each video in which the baseline MBT tracker properly tracks the target object. In total, 11 segments from 10 pedestrian videos[3] and 7 segments from 7 car videos[4] are used in the experiments. The selected segments of the pedestrian and car videos have an average length of 121 and 293 frames, respectively. Each test segment contains a single object to be tracked. First-frame annotations for all movie segments are shown in Figure 5.1. We measure the performance of MBT and FIT-MBT in our experiments using ALE and CDR. We set λ to $\frac{1}{2}\sigma^2$ with $\sigma^2 = 5$ pixels.

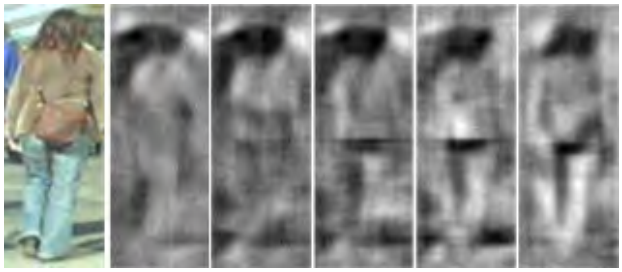


Figure 6. HOGgles visualizations of feature subsets. The first image on the left shows the target object in *Sunny day*. The rest of the images from left to right visualize 10%, 20%, 30%, 50%, and 100% feature usage, respectively.

5.2. Results

Figure 7 presents the results obtained using our the model-based pedestrian and car trackers (MBT) with and without FIT. The first and the second row of the Figure 7 show the results for pedestrian and car tracking, respectively. The left column of the figure presents the ALE as a function of the number of features evaluated, the middle column displays the CDR as a function of the number of features evaluated, and the right column shows the change in the percentage of the evaluated features as a function of

[3]Pedestrian videos were obtained from `http://cvlab.hanyang.ac.kr/tracker_benchmark_v10.html` [19] and `http://www.vision.ee.ethz.ch/~aess/dataset` [5].

[4]Car videos were obtained from `http://www.alov300.org` [15] and `http://cvlab.hanyang.ac.kr/tracker_benchmark_v10.html` [19].

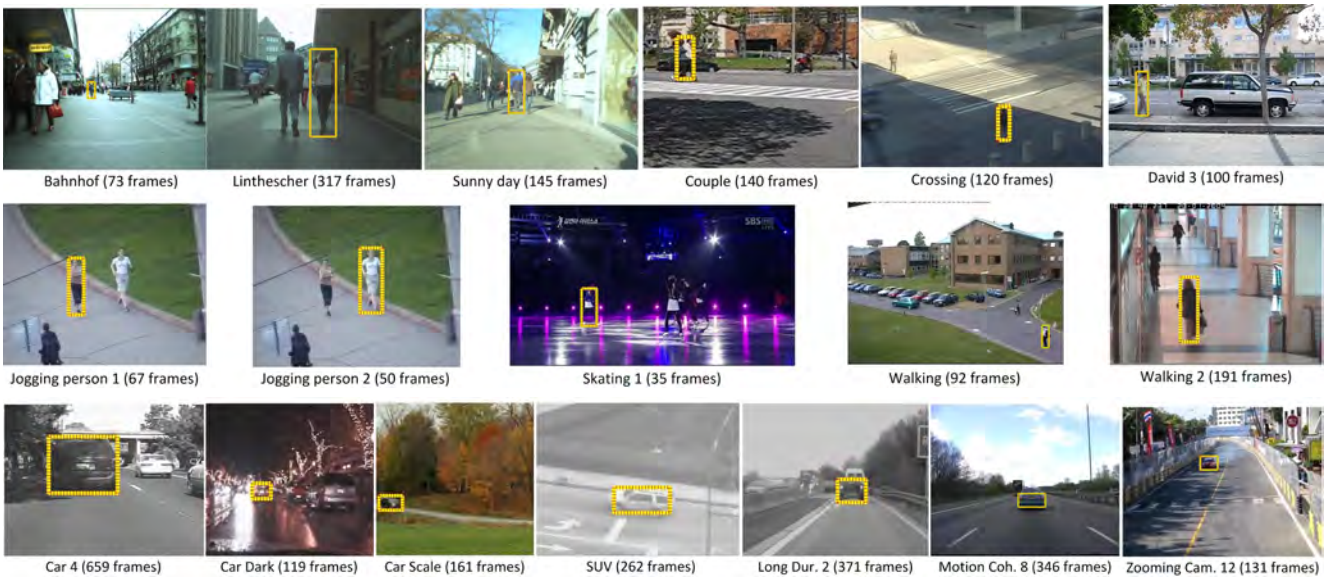


Figure 5. First frame and the initial bounding box for each pedestrian and car movie segment.

threshold θ. Note that the evaluation of all features (100%) corresponds to the baseline MBT tracker.

The results presented in the figure show that evaluating 10%–20% of the features suffices to obtain the same accuracy and precision as the baseline MBT tracker. On average, the CDR increases when more features are used and the ALE decreases. The results also show that using less than 10% of the features can cause the tracker to perform unpredictably. For instance, on the *Skating* video, using 1% of the features performs 17 pixels better (in terms of ALE) than using 2% of the features. This can be explained by the low dimensionality of the evaluated features: adding 1% of features to 1% has a much larger effect than adding 1% to 10% of the features. Another interesting observation from our results is that on the *Crossing* video, evaluating 2% of the features leads to a higher CDR (95.83%) than evaluating all features (93.33%). Indeed, for the most of pedestrian videos, using a very small amount of the features appears to perform on par with using all features, both in terms of ALE as in terms of CDR. This suggests that the pedestrian detector may be overfitted to the Pascal VOC data.

In Figure 6, we visualize the subsets of features (30%, 20%, and 10%) that FIT-MBT uses in *Sunny day* using HOGgles [18]. The visualizations illustrate it is possible to represent a target object well with only a small amount of features, which explains the strong performance of FIT-MBT even when most features are ignored.

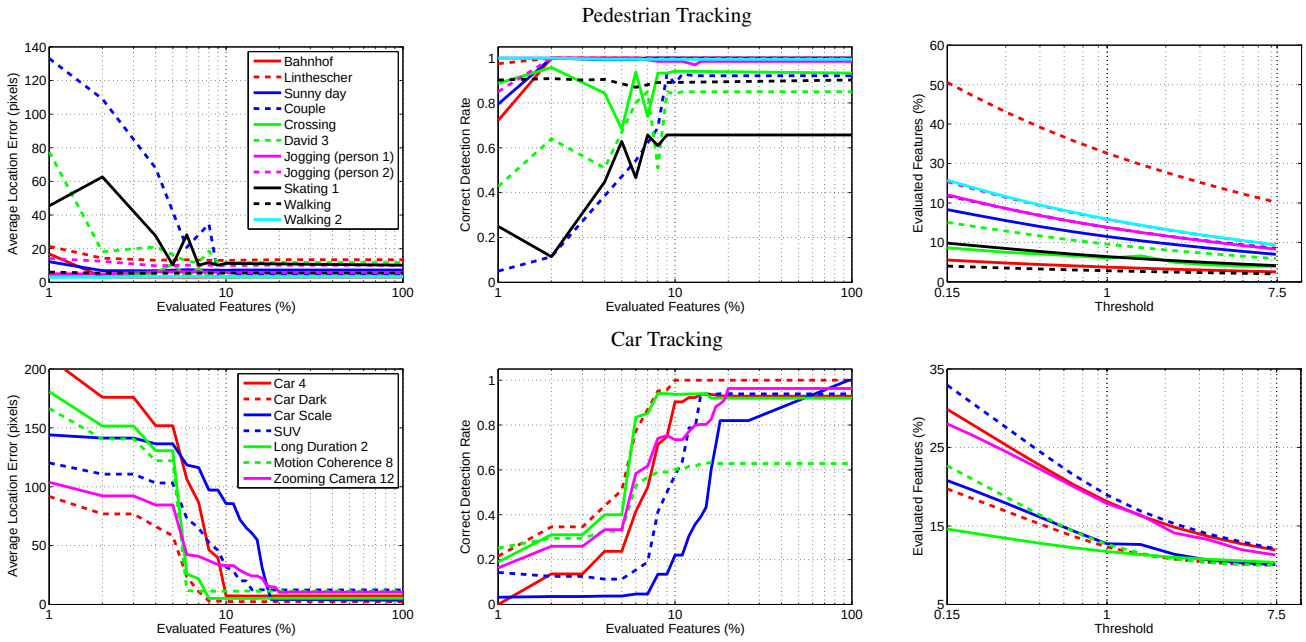
6. Conclusion

We presented *feature ignoring tracking* (FIT): a novel algorithm that reduces the computational costs of a popular family of sliding-window trackers based on tracking-by-detection. Our experiments with state-of-the-art model-free and model-based trackers based on HOG features and linear models show that FIT may reduce the average number of inspected features by up to 90% without affecting the accuracy of the trackers. It should be noted, however, that the FIT algorithm is much more widely applicable: the only two assumptions it makes is that (1) the classifier response decomposes across features or feature subsets and (2) feature values can be bounded above and below. The former assumption holds not only for linear models, but also for mixture models and for kernel-based models such as those used in Struck [9]. The latter assumption holds for many popular features, including HOG, SIFT, Haar, and LBP features, and bags of visual words.

One of the key advantages of FIT is that it has a parameter, θ, that permits trading off speed for accuracy. The availability of such a trade-off parameter is essential when there is a computational budget during runtime, *e.g.*, when the same tracker is deployed on different platforms. We did not consider tracking-on-a-budget in this paper, but we do aim to explore runtime budgets in future work. In future work, we also plan to study extensions of FIT to trackers based on pictorial-structures models as well as to trackers that employ mixture models [6, 21, 22].

Acknowledgements

This work was supported by EU-FP7 Social Signal Processing (SSPNet), EU-FP7 INSIDDE, AAL SALIG++, and by the China Scholarship Council.



(a) (b) (c)

Figure 7. Performance of FIT-MBT in pedestrian and car tracking: (a) The average distance error by FIT-MBT as a function of the percentage of features evaluated. (b) Correct detection rate as a function of the percentage of features evaluated by FIT-MBT. (c) The percentage of features evaluated by FIT-MBT as a function of threshold θ. Note that the evaluation of all features (100%) corresponds to the baseline, and that the x-axis of the plots uses a logarithmic scale.

References

[1] B. Babenko, M.-H. Yang, and S. Belongie. Robust object tracking with online multiple instance learning. *IEEE TPAMI*, 33(8):1619–1632, 2011.

[2] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7(Mar):551–585, 2006.

[3] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, pages 886–893, 2005.

[4] T. B. Dinh, N. Vo, and G. Medioni. Context tracker: Exploring supporters and distracters in unconstrained environments. In *CVPR*, pages 1177–1184, 2011.

[5] A. Ess, B. Leibe, K. Schindler, and L. van Gool. A mobile vision system for robust multi-person tracking. In *CVPR*, 2008.

[6] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *TPAMI*, 32(9):1627–1645, 2010.

[7] S. Gorbunov, D. Rohr, K. Aamodt, T. Alt, H. Appelshauser, A. Arend, M. Bach, B. Becker, S. Bottger, T. Breitner, et al. Alice hlt high speed tracking on gpu. *IEEE Trans. on Nuclear Science*, 58(4):1845–1851, 2011.

[8] H. Grabner, M. Grabner, and H. Bischof. Real-time tracking via on-line boosting. In *BMVC*, pages 47–56, 2006.

[9] S. Hare, A. Saffari, and P. H. Torr. Struck: Structured output tracking with kernels. In *ICCV*, pages 263–270, 2011.

[10] M. Isard and A. Blake. Condensation - conditional density propagation for visual tracking. *IJCV*, 29(1):5–28, 1998.

[11] Z. Kalal, J. Matas, and K. Mikolajczyk. P-n learning: Bootstrapping binary classifiers by structural constraints. In *CVPR*, pages 49–56, 2010.

[12] S. Karayev, T. Baumgartner, M. Fritz, and T. Darrell. Timely object recognition. In *NIPS*, 2012.

[13] C. H. Lampert, M. B. Blaschko, and T. Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *CVPR*, pages 1–8, 2008.

[14] T. Ojala, M. Pietikainen, and T. Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *TPAMI*, 24(7):971–987, 2002.

[15] A. W. M. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah. Visual tracking: An experimental survey. *TPAMI*, (PrePrints).

[16] S. Vijayanarasimhan and K. Grauman. Efficient region search for object detection. In *CVPR*, 2011.

[17] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, pages 511–518, 2001.

[18] C. Vondrick, A. Khosla, T. Malisiewicz, and A. Torralba. Hoggles: Visualizing object detection features. In *ICCV*, 2013.

[19] Y. Wu, J. Lim, and M.-H. Yang. Online object tracking: A benchmark. In *CVPR*, 2013.

[20] Z. Xu, K. Weinberger, and O. Chapelle. The greedy miser: Learning under test-time budgets. In *ICML*, 2012.

[21] L. Zhang and L. van der Maaten. Structure preserving object tracking. In *CVPR*, 2013.

[22] X. Zhu and D. Ramanan. Face detection, pose estimation, and landmark localization in the wild. In *CVPR*, 2012.