# On-line Hough Forests

Samuel Schulter[1]
schulter@icg.tugraz.at
Christian Leistner[2]
leistner@vision.ee.ethz.ch
Peter M. Roth[1]
pmroth@icg.tugraz.at
Luc Van Gool[2]
vangool@vision.ee.ethz.ch
Horst Bischof[1]
bischof@icg.tugraz.at

[1] Institute for Computer Graphics and Vision
Graz University of Technology
Austria

[2] Computer Vision Laboratory
ETH Zürich
Switzerland

**Abstract**

Hough forests have emerged as a powerful and versatile method, which achieves state-of-the-art results on various computer vision applications, ranging from object detection over pose estimation to action recognition. The original method operates in off-line mode, assuming to have access to the entire training set at once. This limits its applicability in domains where data arrives sequentially or when large amounts of data have to be exploited. In these cases, on-line approaches naturally would be beneficial. To this end, we propose an on-line extension of Hough forests, which is based on the principle of letting the trees evolve on-line while the data arrives sequentially, for both classification and regression. We further propose a modified version of off-line Hough forests, which only needs a small subset of the training data for optimization. In the experiments, we show that using these formulations, the classification results of classic Hough forests could be reached or even outperformed, while being orders of magnitudes faster. Furthermore, our method allows for tracking arbitrary objects without requiring any prior knowledge. We present state-of-the-art tracking results on publicly available data sets.

## 1 Introduction

Recently, several works have been proposed that show the applicability and usefulness of the generalized Hough transform [3] for visual object detection (*e.g.*, [14, 15, 18, 20]). Approaches based on the generalized Hough transform enjoy increased popularity due to their simple nature while being able to achieve state-of-the-art results in several vision problems.

Although previous methods provide highly accurate results for various detection tasks, they are limited by complicated parameter settings and by their slow computational speed in both learning and evaluation. Gall and Lempitsky [11] as well as Okoda [19] alleviated these practical disadvantages by incorporating the learning of the Hough transform as discriminative regression process into randomized decision trees. This leveraged the further usage of


BMVC 2011 http://dx.doi.org/10.5244/C.25.128

this method to a wide range of applications, such as, tracking [12], action recognition [26], pose estimation [10], and medical imaging [7].

Hough Forests (HF) operate in off-line mode, which means that they assume having access to the entire training set at once. However, this limits their application in situations where the data arrives sequentially, *e.g.*, in object tracking, in incremental or interactive learning, in dynamic environments or large-scale learning. Essentially, the latter one becomes increasingly important as the number of digital images and videos is exploding. For all of these applications, on-line methods inherently can perform better.

In this paper, we propose an on-line learning scheme for Hough forests, which allows to extend their usage to further applications, such as the tracking of arbitrary target instances or large-scale learning of visual classifiers. Hough forests are ensembles of randomized decision trees, consisting of both classification and regression nodes, which are trained recursively. Since this is a hard task to be done on-line, we follow a recent strand of research [8, 23] that circumvents the recursive on-line update of classification trees by following a tree-growing principle.

We let the trees grow on-line while the data arrives sequentially and we do this for both classification and regression nodes. This requires to find reasonable splitting functions with only a small subset of the data, which does not necessarily have to be a disadvantage when building random forests. In particular, the accuracy of a tree ensemble does not only depend on the predictive strength of the individual trees but also on the correlation between each tree. Thus, however, subsampling the data, which is a necessity of our on-line method, can lead to decreased correlation of the trees and increased accuracy, as we will see later. To this end, we further derive a new but simple splitting procedure for off-line Hough forests based on subsampling the input space on the node level. In summary, we propose two extensions of the Hough forest framework: first, an on-line version of the method, where both the codebook and the discriminative classifier can be trained on sequentially arriving data and, second, a simple yet effective subsampling scheme that also leads to improved results for the off-line approach.

We split our experimental evaluation into two parts. First, we demonstrate on three object detection data sets that both, our on-line formulation and subsample splitting scheme, can reach similar performance compared to the classical Hough forests and can even outperform them. Additionally, during training both proposed methods are magnitudes faster than the original approach. Second, we demonstrate the power of our method for visual object tracking. Especially, our focus lies on tracking objects of a priori unknown classes. We present results on seven tracking data sets and show that our on-line HFs can outperform state-of-the-art tracking-by-detection methods.

The remainder of the paper is organized as follows: In Section 2, we review the Hough forest framework in more detail. Section 3 introduces on-line Hough forests. Additionally, we present our subsampling scheme as a modification to the off-line HFs. Experimental results are presented in Section 4, and Section 5 concludes the paper with a summary and an outlook.

## 2 Hough Forests

The Generalized Hough transformation was originally used to detect general parametric shapes, such as lines or circles [3]. Nowadays, this principle is used with great success to learn the mapping from localized features into a Hough image using a codebook. Typ-

ical approaches aggregate votes of local patches or "voting elements" in the Hough space, where the voting elements may be image regions [18], interest points [15] or densely sampled patches [11, 19]. Usually, the voting elements are clustered into a codebook, which is further used for learning a classifier on labeled data. The actual detection is then performed by searching for peaks in the Hough image.

For instance, the implicit shape model (ISM) proposed by Leibe *et al.* [15] first clusters local image descriptors based on their relative location with respect to marked object centers and then trains a discriminative classifier. During testing, one can use the classifier in accordance with the codebook entries to get probabilistic estimates for object center locations. The method can further be improved by re-weighting the votes within a max-margin framework [18] or by smarter post-processing of the vote estimates [4, 20].

In contrast to previous methods, Hough forests [11, 19] simultaneously cluster image features based on their spatial distribution and train a discriminative classifier using randomized trees [6]. Random forests were introduced by Breiman [6] and are ensembles of decision trees $\mathcal{F} = \{\mathcal{T}_1, \ldots, \mathcal{T}_T\}$, where $T$ is the number of trees. Each decision tree is a function of the form $f(\mathbf{x};\Theta) : \mathcal{X} \rightarrow \mathcal{Y}$, where $\Theta$ defines the parameters for all splitting nodes in the tree; $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^d$ is a feature vector. During training, each tree is provided with a subset of the data, consisting of labeled samples $\{\mathbf{x}_i, y_i\} \in \{\mathcal{X}, \mathcal{Y}\}$, where $y_i \in \{1, \ldots, K\}$ and $K$ is the number of classes. RFs are constructed in a recursive manner, where each node tries to find a splitting function $\xi(\cdot)$ by optimizing the information gain

$$\Delta H = -\frac{|I_l|}{|I_l| + |I_r|} H(I_l) - \frac{|I_r|}{|I_l| + |I_r|} H(I_r) \quad , \tag{1}$$

where $I_l$ and $I_r$ are the subsets of samples falling in the left and right child nodes, respectively. $H(\cdot)$ is chosen to either be the entropy $-\sum_{k=1}^{K} p_k \cdot log(p_k)$ or the Gini index $-\sum_{k=1}^{K} p_k \cdot (1 - p_k)$; $p_k$ is the probability of the current node to belong to class $k$. The trees stop growing if the maximum depth is reached or if a node is pure, *i.e.*, it contains only samples from one class. Then, each leaf node collects the statistics of the samples falling in this node. In the evaluation phase, the probability of a test sample $\mathbf{x}$ belonging to class $k$ is given by

$$p(k|\mathbf{x}) = \frac{1}{T} \sum_{t=1}^{T} p_t(k|\mathbf{x}) \quad , \tag{2}$$

where $p_t(\cdot)$ is the estimate of class $k$ in the leaf node of the $t^{th}$ tree for the test sample. As stated in [6], one can define a classification margin $m_l(\mathbf{x}, y) = p(y|\mathbf{x}) - max_{k \in \mathcal{Y}, k \neq y} p(k|\mathbf{x})$. It follows that for correct predictions, $m_l(\mathbf{x}, y) > 0$ has to hold. Based on the margin $m_l$, Breiman [6] defines the generalization error as $GE = E_{(\mathcal{X},\mathcal{Y})}(m_l(\mathbf{x}, y) < 0)$ and showed that $GE$ has the upper bound

$$GE \leq \bar{\rho} \frac{1 - s^2}{s^2} \quad , \tag{3}$$

where $\bar{\rho}$ defines the mean correlation between two pairs of trees ($\bar{\rho}$ is measured in terms of similarity between the predictions of two trees) and $s$ is the strength of the trees (*i.e.*, the expectation of $m_l(\mathbf{x}, y)$ over the data distribution). That is, a low generalization error requires strong, but also decorrelated trees. When training random forests, the trees are decorrelated by sub-sampling the samples for each tree individually with replacement (*a.k.a.* bagging) and by randomly selecting the split tests.

Hough forests [11] combine classification trees and regression trees within a single framework. They use the labeled training images to extract patches in the form $\{P_i =$

$(A_i, y_i, \mathbf{d}_i)\}$, where $A_i$ is the image descriptor and consists of 32 channels, each a $16 \times 16$ patch, which describes different properties of the image. The object label $y_i$ and the offset vector $\mathbf{d}_i$, which points to the object centroid and is undefined for negative patches, complete a patch $P_i$. All patches are used to build the trees, where each node tries to find the best split according to some optimality criterion. Hough forests define two different optimization methods, one based on classification the other one on regression. Each node selects randomly which of these criteria is used. The classification criterion is equal to that of the standard random forests, *i.e.*, the information gain (see Equation 1). The regression criterion tries to minimize the variance of the offset vectors $\mathbf{d}_i$ for each of its child nodes as follows:

$$min \sum_{I_l} (\mathbf{d}_i^l - \bar{\mathbf{d}}^l) + \sum_{I_r} (\mathbf{d}_i^r - \bar{\mathbf{d}}^r) \quad , \tag{4}$$

where $\bar{\mathbf{d}}^l$ and $\bar{\mathbf{d}}^r$ are the means of all offset vectors $\mathbf{d}_i$ falling in the left and right child nodes, respectively. Each leaf node estimates a probability about fore- and background equal to Equation 2 and collects all offset vectors $\mathbf{d}_i$, which fall in this leaf node.

For detection, a sliding window approach (also at different scales) is applied in combination with the generalized Hough transformation [3]. Patches $P_i$ are extracted at each location $\mathbf{l}$ in the test image and propagated through the Hough forest according to the appearance $A_i$ of the patch. All patches end up in leaf nodes of the forest, which have a set of offset vectors $D_L$ and a probability of foreground $C_L$ stored. Each patch $P_i$ then "votes" for an object center in a 2D Hough image $V$ by adding the value $\frac{C_L}{\|D_L\|}$ to the positions $\{\mathbf{l} - \mathbf{d}_i | \mathbf{d}_i \in D_L\}$. That is, the offset vectors from the corresponding leaf nodes vote for an object center in the test image with a weight corresponding to the probability of foreground.

Besides object detection, Hough forests have successfully been applied to action recognition [26], pose estimation [10], medical imaging [7] and robotics [25]. They have also been applied to class-specific tracking [12], where an accurate model is learned off-line using hand-annotated data of a predefined object class. The learned codebook is adapted on-line for tracking a specific instance of that object class. That is, the object classes have to be known beforehand, whereas our method can track arbitrary objects, which are a priori unknown. Recently, Godec *et al.* [13] proposed a tracking method for non-rigid objects with an on-line formulation of HFs including back-projection for a rough segmentation of the tracked object. This segmentation improves the sequential updates of the algorithm, but they build their trees completely random to their full size and just update the leaf node statistics during tracking. In contrast, our approach builds the trees in an on-line fashion and also updates the leaf node statistics.

## 3 On-line Hough Forests

Hough forests, as reviewed above, are off-line learners. That is, they assume having access to all labeled training samples $\{\mathbf{x}_i, y_i\}_{i=1}^N$ at once. This eases the optimization but limits their application in scenarios where the data arrives sequentially, such as object tracking or robotics. In the following, we will show how to extend Hough forests to on-line learning, where we assume that the individual samples $\{\mathbf{x}_t, y_t\}$ are sampled *i.i.d* and sequentially from an infinite pool of data. Each sample $\mathbf{x}_t$ can only be observed once at time $t$ by the learner and is discarded afterwards. In other words, the learner never has access to all samples concurrently.

In order to let HFs grow in an on-line fashion, two essential ingredients of random forests have to be done in an on-line manner. First, RFs use bagging to train the trees, which can, however, also be done on-line [21] by updating all trees with the input sample $k$ times, where $k \sim Poisson(\lambda)$. The second issue which has to be tackled, is the on-line growing of the individual trees, which is less trivial because each tree is trained recursively using hard splitting rules, making it difficult to correct errors on-line. We follow the principles of [8, 22, 23], where the basic idea is to start with a tree consisting of only one node, which is the root node and the only leaf at that time. The tree then starts collecting data and propagating the samples $\{\mathbf{x}_t, y_t\}$ to the leaf nodes. Each leaf node decides on its own, based on a certain criterion (see below), whether to change itself to a splitting node and create two new leaf nodes, or to keep being a leaf node and update its statistics. An overview of this procedure is given in Figure 1.

It is easy to see that one of the main issues in this growing approach is to find a reasonable splitting function $\xi(\cdot)$ with a limited set of data samples. Domingos *et al.* [8] tackle this problem by relying on the so-called Hoeffding bound, which states that, with probability $1-\delta$, the true mean of a random variable $r$ is at least $\bar{r}-\varepsilon$, where $\varepsilon = \sqrt{\frac{R^2 \cdot ln(1/\delta)}{2n}}$; $n$ is the number of samples available and $R$ is the range of the random variable $r$. In contrast, on-line random forests [23] continuously calculate the potential information gain $\Delta H = H(I_p) - \frac{|I_l|}{|I_p|}H(I_l) - \frac{|I_r|}{|I_p|}H(I_r)$ of randomly created node splitting functions. $I_p$, $I_l$, and $I_r$ are the sets of samples of the parent, left child, and right child nodes, respectively. The node gets split if $\Delta H$ exceeds a predefined threshold $\beta$ and a minimum number $\gamma$ of samples has been processed. Although these methods have strong theoretical support, we will show in the experiments that it even suffices to only count the number $n_i$ of samples that a node has seen so far and split when $n_i > \gamma$. That is, each leaf node in the trees collects the first $\gamma$ data samples arriving and, after that, generates several random split functions $\xi_i(\cdot)$. The split function $\xi^*(\cdot)$ that optimizes the impurity criterion of the Hough forests, which is either the information gain for classification nodes (see Equation 1) or the variance of the offset vectors $\mathbf{d}_i$ for regression nodes (see Equation 4), is chosen. Thus, when growing HFs in an on-line fashion, we consider both settings for optimization, classification and regression nodes.

To sum up, each tree in the forest starts with only one leaf node $N_L^0$, which is the root node at that time (0 denotes the current depth of the tree and $L$ stands for "leaf"). This leaf node starts collecting samples $\{\mathbf{x}, y\}$ falling in it and builds its statistics about fore- and background $\frac{\|\{\mathbf{x}|y=1\}\|}{\|\mathbf{x}\|}$. After seeing $\gamma$ samples, the leaf node $N_L^0$ is transformed into a splitting node $N_S^0$ ($S$ now stands for "split") by finding an optimal splitting function $\xi(\cdot)$ (classification or regression) on the available data samples $\{\mathbf{x}^t, y^t\}$, where $t = 1 \dots \gamma$. Two child leaf nodes $N_L^1$ are created and the samples, which were already collected in the parent node $N_S^0$ are propagated to the child nodes according to the new split function $\xi(\cdot)$. These samples can then already be used for building the statistics in the newly created child nodes. This process continues until a maximum depth of the tree is reached or no more training samples are available. As in classic RFs [6] and its on-line version [23], no pruning is applied.

**Modified Off-line Splitting** In the end, this on-line growing scheme simply relies on a subset of the training data $\{\hat{\mathcal{X}}, \hat{\mathcal{Y}}\} \subset \{\mathcal{X}, \mathcal{Y}\}$, which arrives sequentially in the on-line case. Thus, we can also formulate a similar principle in the off-line case, where each splitting node is optimized on such a subset of the data. The random nature of classic random forests, *i.e.*, the random splitting functions and the implemented bagging, has the primary goal of

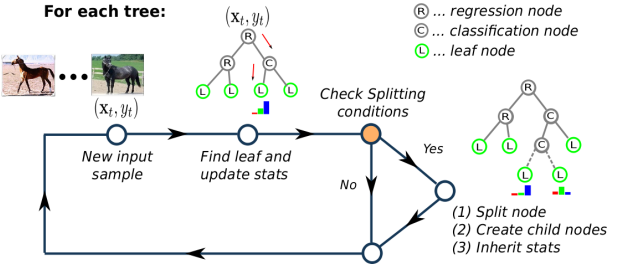



Figure 1: Flowchart of the proposed method: While labeled samples arrive on-line, each tree is updated independently. Each sample is traversed down the tree and the leaf node where it ends up is updated. Then the sample is discarded. If the splitting condition is met, the updated node becomes a split node and creates two child nodes to which it passes its collected statistics.

increasing the decorrelation $\bar{\rho}$ between the trees in the ensemble, see Equation 3. By considering this fact, optimizing each node of the Hough forest on such a subset $\{\hat{\mathcal{X}}, \hat{\mathcal{Y}}\} \subset \{\mathcal{X}, \mathcal{Y}\}$ even makes sense to further increase $\bar{\rho}$. If the subset of the data still gives a reasonable representation of the data set's underlying structure, the strength $s$ of the trees does not suffer too much. Furthermore, in case of computer vision applications, the data representation is often ambiguous. Thus, optimization on a subset of the data in the node level, can work against overfitting on the training set and improve the generalization performance.

For that reasons, we propose a further modification of the learning procedure in the off-line HFs by sub-sampling the available training set in each splitting node. The subset $\{\hat{\mathcal{X}}, \hat{\mathcal{Y}}\}$ is drawn uniformly from all training data available. Then, we optimize either the classification problem or the regression, see Equations 1 and 4, respectively. After finding an optimal splitting function $\xi(\cdot)$ on the sub-sampled data set, all available samples are propagated to the newly created left and right child nodes accordingly. The only parameter, which has to be defined, is the size $\gamma$ of the subset $\{\hat{\mathcal{X}}, \hat{\mathcal{Y}}\}$.

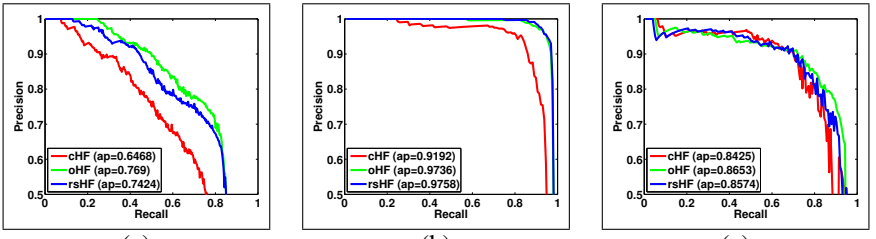
# 4 Experiments

To show the benefits of the proposed approaches we demonstrate them for object detection as well as for tracking. First, we compare the classic Hough forests with our proposed sub-sampling and on-line methods for object detection. Then, we present tracking results, where we demonstrate our on-line formulation of the Hough forests for several standard tracking data sets[1].

## 4.1 Object detection

In this section, we compare our two proposed methods with the original Hough forest implementation for several object detection tasks. To this end, we chose three different data sets, namely TUD-pedestrian [1], Weizmann-horses [5], and ETHZ-cars [16]. In the following, we denote our sub-sampling Hough forest with *rsHF* and the on-line extension with

[1] To allow for fair comparison, we used the original C++ code provided by [11], where we added our extensions. The code can be downloaded from http://lrs.icg.tugraz.at/download.php



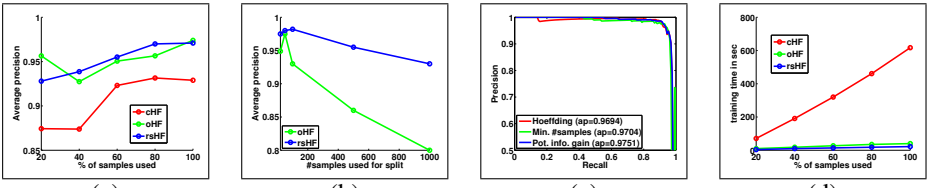

(a) (b) (c)

Figure 2: Detection results for the two proposed methods *oHF* and *rsHF*, compared to the classic Hough forest implementation. Precision-recall curves are shown for three different datasets: TUD-pedestrian [1] (a), Weizmann-horses [5] (b), and ETHZ-cars [16] (c).

*oHF*. Our evaluation is based on precision-recall curves following the PASCAL overlap criterion [9]. We used 15 trees with a maximum depth of 15 for all methods in each of the experiments. Also the number of randomly created splitting functions was equal for all methods and experiments and set to 2000.

**Overall performance:** In this experiment, our main intention is to compare the generalization performance of the methods on all three data sets. The parameter $\gamma$ was set to 20 for both *rsHF* and *oHF*. The resulting precision-recall curves are depicted in Figure 2. As can easily be seen from the curves, both proposed methods, *i.e.*, *rsHF* and *oHF*, can significantly outperform the classic HF on the three datasets. While the improvement is actually very high for the TUD and Weizmann data sets, one can only obtain a slight increase in performance for the ETHZ data set. However, the results indicate that the random sub-sampling and on-line growing of the trees increase the decorrelation $\bar{\rho}$ of the trees, while still keeping its strength $s$.

**Experimental analysis:** In addition, we investigate the overfitting effect of the classic HF in this experiment in more detail and present results about the computational costs of our proposed methods for the Weizmann-horses dataset. Furthermore, we compare the three different splitting methods when a tree is grown on-line, *i.e.*, the Hoeffding bound, the potential information gain, and the minimum number of seen samples. In a first experiment, we varied the number of samples used for training and the parameter $\gamma$. The resulting curves are presented in Figure 3 (a) & (b). As can easily be seen from Figure 3(a), rsHF and oHF can outperform the classic HF on this dataset even for smaller amounts of available training data. The second plot shows the influence of the parameter $\gamma$. As one can see, too low or too high values decrease the predictive performance of *rsHF* and *oHF*. Please note that setting $\gamma$ to the maximum value, *i.e.*, the amount of data available in each node, yields the classic formulation of the Hough forests. For the on-line version (*oHF*), the strong decrease of performance for higher values of $\gamma$ can be explained by the fact that *oHF* stops growing if the number of samples in a leaf node cannot exceed $\gamma$ anymore. In contrast, *rsHF* just cannot perform the sub-sampling anymore, but the tree can still grow. Figure 3(c) shows the influence of the choice of the split function, as described above. Here, we cannot see significant differences in the three methods except the effort of finding suitable parameters, which is easier with our simple method (only the parameter $\gamma$ has to be defined). In a last experiment, we investigated the computational cost for *rsHF*, *oHF*, and the classic Hough forests. The




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

Figure 3: Figures (a) and (b) show average precision values when the number of samples or the parameter $\gamma$ are varied, respectively. The effect of different splitting methods is shown in (c) and the training time for *oHF*, *rsHF*, and *cHF* is shown in (d).

results, shown in Figure 3(d), show that both proposed methods significantly outperform the original HF in the necessary time for building the forest.

## 4.2 Object tracking

Next, we evaluate our on-line formulation of Hough forests (oHF) on several standard tracking sequences. We compare with on-line random forests (ORF) [23], which builds the base of our approach, and two other state-of-the-art trackers, namely Multiple Instance Boosting (MILB) [2] and PROST [24]. We do not compare to [12] because this approach requires class-specific knowledge of the tracking object in terms of labeled data, whereas we do not assume any prior knowledge about the target object. For evaluation, we chose the percentage of correctly predicted frames, where we calculate the PASCAL overlap and define a correctly predicted frame for overlaps bigger than 0.5. We resized the tracking sequences such that the bounding box of the target object in the first frame has around 100px in height or width. In this way, we can ensure that the extracted patches contain reasonable information.

We follow the tracking procedure from [17] and perform "one-shot" learning of the on-line HF with the first frame of a sequence by virtually warping the image five times. We take 100 positive and 500 negative patches from these images and update the model. Then, we define a search area in the subsequent frame, which has twice the size of the labeled bounding box from the first frame. Within this area, the Hough image is calculated and a maximum search finds the location $\mathbf{l}$ with the highest confidence $c$ for the target object. This search in the 2D Hough image can also be performed in a 3D Hough space, allowing the algorithm to handle scale changes. We simply calculate additional Hough images in the scales $s_t \pm 0.05$, where $s_t$ denotes the current scale at time $t$ relative to the scale of the first frame in the sequence. According to the value of $c$, we differentiate between three different cases: (*i*) if $c > \Theta_1$, we update the center of the target bounding box to the position $\mathbf{l}$ and, as appropriate, the current scale $s_t$ and the size of the bounding box. Additionally, we extract 10 positive and 10 negative patches from the current frame and update the on-line Hough forest. (*ii*) if $\Theta_1 > c > \Theta_2$, we also update the current position and scale of the model, but do not update the model itself with newly extracted patches. (*iii*) if $\Theta_2 > c$, we neither update the model nor its position and scale [17]. The thresholds $\Theta_1$ and $\Theta_2$, where $\Theta_1 > \Theta_2$, are set to multiples of the mean values of the confidence $c$ for the first few frames of a sequence. For all experiments we use 8 trees with a maximum depth of 8 and set the parameter $\gamma$ to 20.

As can be seen from the tracking results of *oHF* in Table 1, we can compete with all other proposed tracking algorithms and can outperform them in most of the cases. The table also shows that the Hough forest framework brings an enormous boost in tracking

| Sequence | oHF | ORF [23] | MILB [2] | PROST [24] |
|---|---|---|---|---|
| David | **93.5** | 72.0 | 60.0 | 80.0 |
| Facceocc2 | 90.6 | 65.0 | **96.0** | 82.0 |
| Girl | **99.0** | 96.0 | 57.0 | 89.0 |
| board | **95.3** | 10.0 | 36.6 | 75.0 |
| box | **93.7** | 28.3 | 7.6 | 91.4 |
| liquor | **94.0** | 54.0 | 21.0 | 85.4 |
| lemming | 82.3 | 17.0 | **83.0** | 70.5 |
| Average | **92.6** | 48.9 | 51.5 | 81.9 |

Table 1: Tracking results for several sequences of the proposed methods. Here, we show the percentage of correctly predicted frames. (Best performing methods are marked boldface and second best methods are underlined)

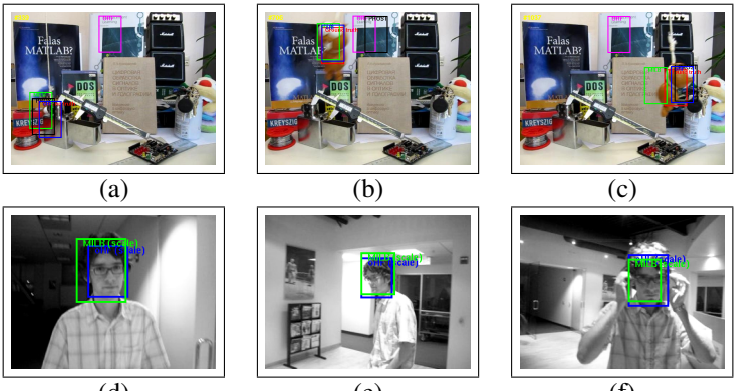



Figure 4: Results from all proposed methods in difficult tracking situations (a-c). Results of *oHF* and MILB [2], both considering scaling, are depicted in (d-f).

performance compared to on-line random forests [23]. We can even outperform PROST, which includes optical flow in the tracking procedure. Some illustrative results showing rather difficult tracking situations (occlusions, motion blur) are depicted in Figure 4(a-c).

Finally, we want to show the treatment of different scales of our proposed method. As we could not find many comparable works, which include scaling or present results about scaling, we only compare with the MILB framework from Babenko *et al*. [2] on the *David* sequence. We present some qualitative results in Figure 4(d-f), where we depict some representative frames comparing both methods and also quantitative results by measuring the mean offset center location. We observed an error of 6.9px for our method, whereas MILB has an error of 10.1px on this sequence.

## 5 Conclusion

This paper introduced an on-line extension of Hough forests, which we successfully applied to object detection and tracking. We integrated ideas from evolving decision trees in order to grow both classification and regression functions on-line, resulting in a powerful and com-

putationally efficient learning framework. We further proposed a novel growing procedure of the classic Hough forests, where, similar to the on-line version, each node is optimized on a small subset of the available data. This enforces the decorrelation between the trees and reduces overfitting in the structure of the trees. Thus, for several detection tasks an improved classification performance could finally be obtained. The experiments on three different data sets show that both the on-line and subsample formulations can outperform the classic Hough forests, while both are being magnitudes faster. Furthermore, we applied our method for standard tracking benchmark data sets and were able to show more than competitive results compared to state-of-the-art tracking-by-detection approaches. In future work, we plan to apply our algorithm to further vision tasks and to analyse the overfitting issue in more detail as well as work on improved splitting criteria for Hough forests.

## 6 Acknowledgement

This work has been supported by the Austrian FFG project Outlier (820923) under the FIT-IT program and the Austrian Science Foundation FWF under the project I535-N23.

## References

[1] M. Andriluka, S. Roth, and B. Schiele. People-tracking-by-detection and people-detection-by-tracking. In *CVPR*, 2008.

[2] B. Babenko, M.-H. Yang, and S. Belongie. Visual tracking with online multiple instance learning. *PAMI*, 2010.

[3] D. H. Ballard. Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122, 1981.

[4] O. Barinova, V. Lempitsky, and P. Kohli. On detection of multiple object instances using hough transforms. In *CVPR*, 2010.

[5] E. Borenstein and S. Ullman. Class-specific, top-down segmentation. In *ECCV*, 2002.

[6] L. Breiman. Random forests. In *Machine Learning*, pages 5–32, 2001.

[7] A. Criminisi, J. Shotton, D. Robertson, and E. Konukoglu. Regression forests for anatomy detection and localization in CT studies. In *MICCAI*, 2010.

[8] P. Domingos and G. Hulten. Mining high-speed data streams. In *ACM SIGKDD international conference on Knowledge discovery and data mining*, 2000.

[9] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88 (2):303–338, 2010.

[10] G. Fanelli, J. Gall, and L. Van Gool. Regression forests for anatomy detection and localization in ct studies. In *CVPR*, 2011.

[11] J. Gall and V. Lempitsky. Class-specific hough forests for object detection. In *CVPR*, 2009.

[12] J. Gall, N. Razavi, and L. Van Gool. On-line adaption of class-specific codebooks for instance tracking. In *BMVC*, 2010.

[13] M. Godec, P. M. Roth, and H. Bischof. Hough-based tracking of non-rigid objects. In *ICCV*, 2011. to appear.

[14] C. Gu, J. J. Lim, P. Arbelaez, and J. Malik. Recognition using regions. In *CVPR*, 2009.

[15] B. Leibe and B. Schiele. Robust object detection with interleaved categorization and segmentation. *IJCV*, 2008.

[16] B. Leibe, N. Cornelis, K. Cornelis, and L. Van Gool. Dynamic 3D scene analysis from a moving vehicle. In *CVPR*, 2007.

[17] C. Leistner, M. Godec, S. Schulter, M. Werlberger, A. Saffari, and H. Bischof. Improving classifiers with unlabeled weakly-related video. In *CVPR*, 2011.

[18] S. Maji and J. Malik. Object detection using a max-margin hough transform. In *ICCV*, 2009.

[19] R. Okoda. Discriminative generalized hough transform for object detection. In *ICCV*, 2009.

[20] B. Ommer and J. Malik. Multi-scale object detection by clustering lines. In *ICCV*, 2009.

[21] N. C. Oza and S. Russell. Online bagging and boosting. In *Artificial Intelligence and Statistics*, pages 105–112, 2001.

[22] J. Pakkanen, J. Ivarinen, and E. Oja. The evolving tree - a novel self-organizing network for data analysis. *Neural Processing Letters*, 2004.

[23] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof. On-line random forests. In *OLCV*, 2009.

[24] J. Santner, C. Leistner, A. Saffari, T. Pock, and H. Bischof. PROST Parallel Robust Online Simple Tracking. In *CVPR*, 2010.

[25] M. Sun, G. Bradsky, B. Xu, and S. Savarese. Depth-encoded hough voting for joint object detection and shape recovery. In *ECCV*, 2010.

[26] A. Yao, J. Gall, and L. Van Gool. A hough transform-based voting framework for action recognition. In *CVPR*, 2010.