Detecting Flying Objects using a Single Moving Camera

Artem Rozantsev, Vincent Lepetit, and Pascal Fua, *Fellow, IEEE,*

Abstract—We propose an approach for detecting flying objects such as Unmanned Aerial Vehicles (UAVs) and aircrafts when they occupy a small portion of the field of view, possibly moving against complex backgrounds, and are filmed by a camera that itself moves. We argue that solving such a difficult problem requires combining both appearance and motion cues. To this end we propose a regression-based approach for object-centric motion stabilization of image patches that allows us to achieve effective classification on spatio-temporal image cubes and outperform state-of-the-art techniques.
As this problem has not yet been extensively studied, no test datasets are publicly available. We therefore built our own, both for UAVs and aircrafts, and will make them publicly available so they can be used to benchmark future flying object detection and collision avoidance algorithms.

Index Terms—Motion compensation, object detection.

1 Introduction

We are headed for a world in which the skies are occupied not only by birds and planes but also by unmanned drones ranging from relatively large Unmanned Aerial Vehicles (UAVs) to much smaller consumer ones. Some of these will be instrumented and able to communicate with each other to avoid collisions but not all. Therefore, the ability to use inexpensive and light sensors such as cameras for collision-avoidance purposes will become increasingly important.

This problem has been tackled successfully in the automotive world, for example there are now commercial products [1], [2] designed to sense and avoid both pedestrians and other cars. In the world of flying machines much progress has been made towards accurate position estimation and navigation from single or multiple cameras [3], [4], [5], [6], [7], [8], [9], but less in the field of visual-guided collision avoidance [10]. In particular, it is not possible to simply extend the algorithms used for pedestrian and automobile detection to the world of aircrafts and drones, as flying object detection poses some unique challenges:

- The environment is fully three dimensional, which makes the motions more complex (e.g., objects may move in any direction in the 3D space and may appear in any part of the frame).
- Flying objects have very diverse shapes and can be seen against either the ground or the sky, which produces complex and changing backgrounds.
- Given the speeds involved, potentially dangerous objects must be detected when they are still far away, which means they may be very small in the images.

- *A. Rozantsev is with the Computer Vision Laboratory, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland. E-mail: artem.rozantsev@epfl.ch*
- *V. Lepetit is with the Institute for Computer Graphics and Vision, Graz University of Technology, Graz, Austria.*
- *P. Fua is with the Computer Vision Laboratory, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland.*

Manuscript received June 19, 2015; revised September 17, 2014.

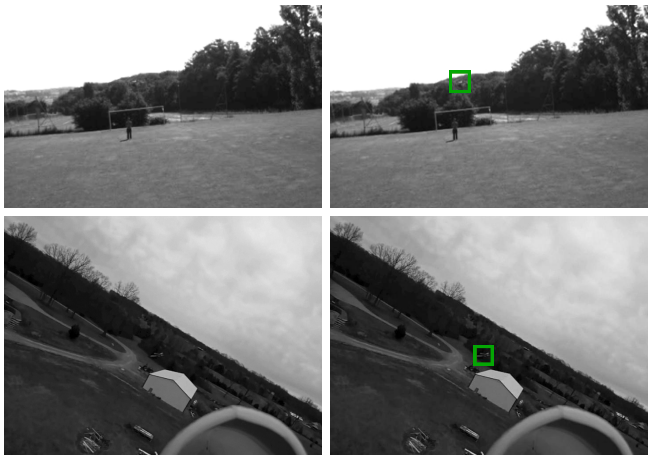


Figure 1: Detecting a small flying object against a complex moving background. (**Left**) It is almost invisible to the human eye and hard to detect from a single image. (**Right**) Yet, our algorithm can find it by using appearance and motion cues.

Fig. 1 illustrates some examples, where even for humans it is hard to find a flying object based just on a single image. By contrast, when looking at the sequence of frames, these objects suddenly pop up and are easily spotted, which suggests that motion cues are crucial for detection.

However, these motion cues are difficult to exploit when the images are acquired by a moving camera and feature backgrounds that are challenging to stabilize because they are non-planar and rapidly changing. Furthermore, since there may be other moving objects in the scene, such as a person in the top row of Fig. 1, motion by itself is not enough and appearance must also be taken into account.

In this paper, we detect whether an object of interest is present and constitutes danger by classifying 3D descriptors computed from spatio-temporal image cubes. We will refer to them as st-cubes. They are formed by stacking motion-stabilized image

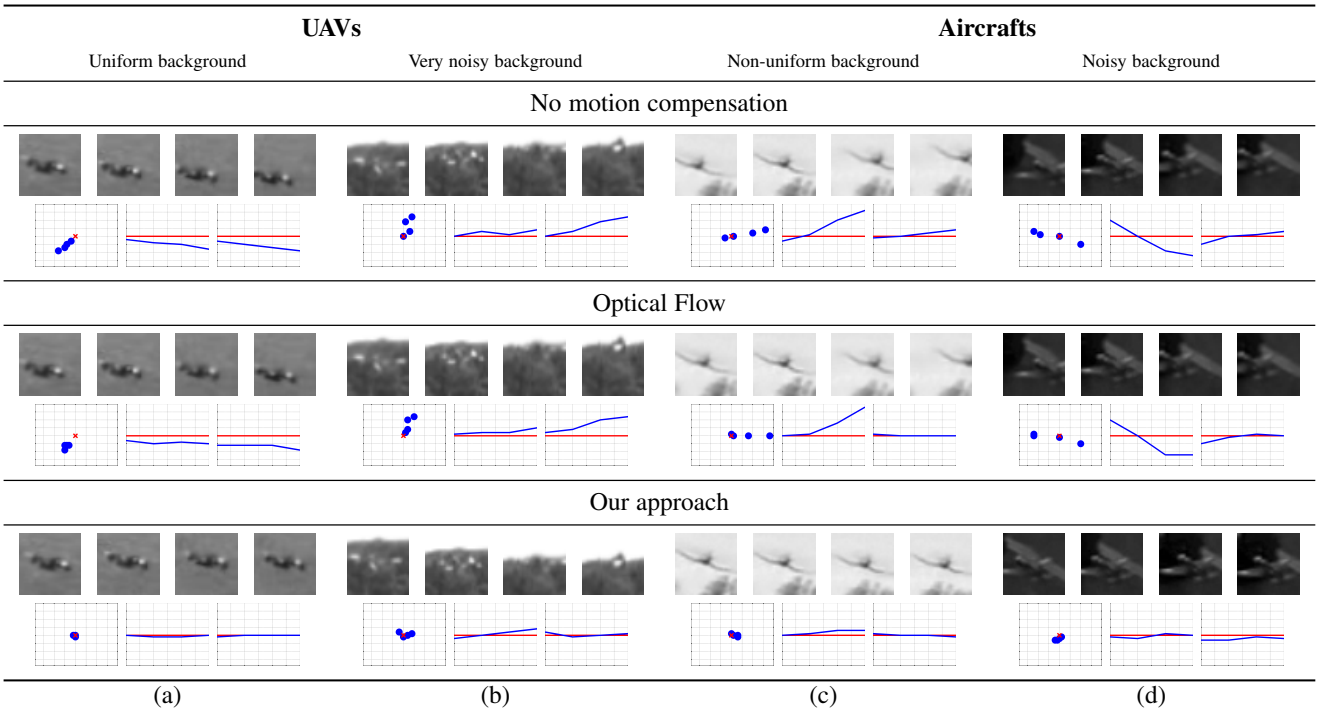


Figure 2: Motion compensation for four different st-cubes of flying objects seen against different backgrounds. (**Top**) For each one, we show four consecutive patches before motion stabilization. In the leftmost plot below the patches, the blue dots denote the location of the true center of the drone and the red cross is the patch center over time. The other two plots depict the x and y deviations of the drone center with respect to the patch center. (**Middle**) The same four st-cubes and corresponding graphs after motion compensation using an optical flow approach, as suggested by [11]. (**Bottom**) The same four st-cubes and corresponding graphs after motion compensation using our approach.

windows over several consecutive frames, which give more information than using a single image. What makes this approach both practical and effective is a regression-based motion-stabilization algorithm. Unlike those relying on optical flow, it remains effective even when the shape of the object to be detected is blurry or barely visible, as illustrated by Fig. 2. This arises from the fact that learning-based motion compensation focuses on the object and is more resistant to complicated backgrounds, compared to the optical flow method as shown in Fig. 2.

St-cubes have been routinely used for action recognition purposes [12], [13], [14] using a monocular camera. By contrast, most current detection algorithms work either on a single frame, or by estimating the optical flow from consecutive frames. Our approach can therefore be seen as a way to combine both the appearance and motion information to achieve effective detection in a very challenging context. In our experiments we show that this method allows to achieve higher accuracy, comparing to either appearance or motion-based methods individually.

We first proposed using st-cubes for flying objects detection in an earlier conference paper [15]. In this initial version of our processing pipeline, we performed motion compensation using boosted trees. In this paper we refine this idea by using deep learning techniques that yield better stabilization and, thus, better overall performance.

2 Related Work

Approaches for detecting moving objects can be classified into three main categories: those that rely on appearance in individual frames, those that rely primarily on motion information across frames, and those that combine the two. We briefly review all three types in this section. In the results section, we will demonstrate that we can outperform state-of-the-art representatives of each class.

Appearance-based methods rely on Machine Learning and have proved to be powerful even in the presence of complex lighting variations or cluttered background. They are typically based on Deformable Part Models (DPM) [16], Convolutional Neural Networks (CNN) [17], or Random Forests [18]. Among them the Aggregate Channel Features (ACF) [19] algorithm is considered as one of the best.

These approaches work best when the target objects are sufficiently large and clearly visible in individual images, which is often not the case in our applications. For example, in the images of Fig. 1, the object is small and it is almost impossible to make out from the background without motion cues.

Motion-based approaches can themselves be subdivided into two subclasses. The first comprises those that rely on background subtraction [20], [21], [22], [23] and determine objects as groups of pixels that are different from the background. The second includes those that depend on optical flow [24], [25], [26].

Background subtraction works best when the camera is static or its motion is small enough to be easily compensated for, which is not the case for the on-board camera of a fast moving aircraft.

Flow-based methods are more reliable in such situations but still critically dependent on the quality of the flow vectors, which tends to be low when the target objects are small and blurry. Some methods combine both optical flow and background subtraction algorithms [27], [28]. However, in our case there may be motion in different parts of the images, for example people or tree

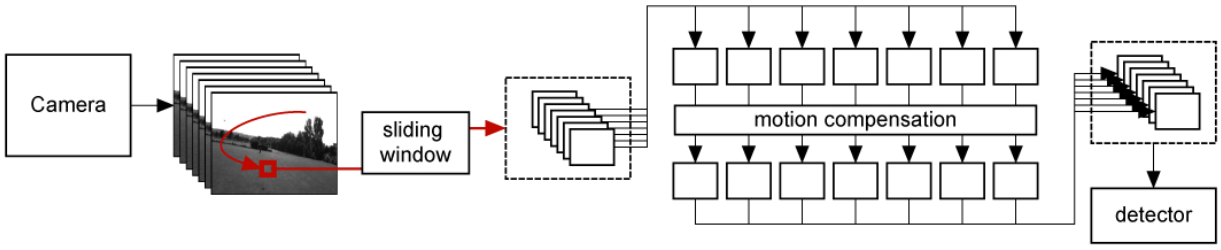


Figure 3: Object detection pipeline with st-cubes and motion compensation. Provided a set of video frames from the camera, we use a multi-scale sliding window approach to extract st-cubes. We than process every patch of the st-cube to compensate for the motion of the aircraft and then run the detector. (best seen in color)

tops. Thus motion information is not enough for reliable flying object detection. Other methods that combine optical flow and background subtraction, such as [29], [30], [31], [32] still critically depend on optical flow, which is often estimated with [26] and thus may suffer from the low quality of the flow vectors. In addition to optical flow dependence, [31] makes an assumption that camera motion is translational, which is violated in aerial videos.

Hybrid approaches combine information about object appearance and motion patterns and are therefore the closest in spirit to what we propose. For example, in [33], histograms of flow vectors are used as features in conjunction with more standard appearance features and are fed to a statistical learning method. This approach was refined in [11] by first aligning the patches to compensate for motion and then using the differences of the frames, which may or may not be consecutive, as additional features. The alignment relies on the Lucas-Kanade optical flow algorithm [25]. The resulting algorithm works very well for pedestrian detection and outperforms most of the single-frame methods. However, when the target objects become smaller and harder to see, the flow estimates become unreliable and this approach, like the purely flow-based ones, becomes less effective.

3 Detection Framework

Our detection pipeline is illustrated by Fig. 3 and comprises the following steps:

- Divide the video sequence into N-frame overlapping temporal slices. The larger the overlap is, the higher the precision but only up to a point. Our experiments show that making the overlap more than 50% increases computation time without improving performance. Thus, 50% is what we used.
- Build st-cubes from each slice using a sliding window approach, independently at each scale.
- Apply our motion compensation algorithm to the patches of each of the st-cubes to create stabilized st-cubes.
- Classify each st-cube as containing an object of interest or not.
- Since each scale has been processed independently, we perform non-maximum suppression in scale space. If there are several detections for the same spatial location at different scales, we only retain the highest-scoring one. As an alternative to this simple scheme, we have developed a more sophisticated learning-based one, which we discuss in more details in Section 6.4.

In this section, we introduce two separate approaches—one based on boosted trees, the other one on Convolutional Neural Networks—to deciding whether or not an st-cube contains a target object and will compare their respective performance in Section 5. We will discuss motion compensation in Section 4.

More specifically, we want to train a classifier that takes as input st-cubes such as those depicted by Fig. 4 and returns 1 or -1, depending on the presence or absence of a flying object. Let (s_x, s_y, s_t) be the size of our st-cubes. For training purposes, we use a dataset of pairs $(b_i, y_i), i \in [1, N]$, where $b_i \in \mathbb{R}^{s_x \times s_y \times s_t}$ is an st-cube, in other words s_t image patches of resolution $s_x \times s_y$ pixels. Label $y_i \in \{-1, 1\}$ indicates whether or not a target object is present.

3.1 3D HoG with Gradient Boost

The first approach we tested relies on boosted trees [34] to learn a classifier $\psi(\cdot)$ of the the form $\psi(b) = \sum_{j=1}^{H} \alpha_j h_j(b)$, where $\alpha_{j=1..H}$ are real valued weights, $b \in \mathbb{R}^{s_x \times s_y \times s_t}$ is the input st-cube, $h_j : \mathbb{R}^{s_x \times s_y \times s_t} \rightarrow \mathbb{R}$ are weak learners, and H is the number of selected weak learners, which controls the complexity of the classifier. The α's and h's are learned in a greedy manner, using the Gradient Boost algorithm [34], which can be seen as an extension of the classic AdaBoost to real-valued weak learners and more general loss functions.

In standard Gradient Boost fashion, we take our weak learners to be regression trees $h_j(b) = T(\theta_j, \text{HoG3D}(b))$, where θ_j denotes the tree parameters and HoG3D(b), the 3-dimensional Histograms of Gradients (HoG3D) computed for b. HoG3D was introduced in [14], and can be seen as an extension of the standard HoG [35] with an additional temporal dimension. It is fast to compute and proved to be robust to illumination changes in many applications, and allows us to combine appearance and motion efficiently.

At each iteration j, the weak learner $h_j(\cdot)$ with the corresponding weight α_j is taken as the one that minimizes the exponential loss function:

$$(h_j(.), \alpha_j) = \underset{h(.),\alpha}{\text{argmin}} \sum_{i=1}^{N} e^{-y(\psi_{j-1}(b_i) + \alpha h(b_i))} . \qquad (1)$$

The tests in the nodes of the trees compare one coordinate of the HoG3D vector with a threshold, both selected during the optimization.

3.2 Convolutional Neural Networks

Since Convolutional Neural Networks (CNN) [36] have proved very successful in many detection problems, we have tested it as an alternative classification method. We use the architecture depicted by Fig. 5, which alternates convolutional layers and pooling layers. Convolutional layers use 3D linear filters while

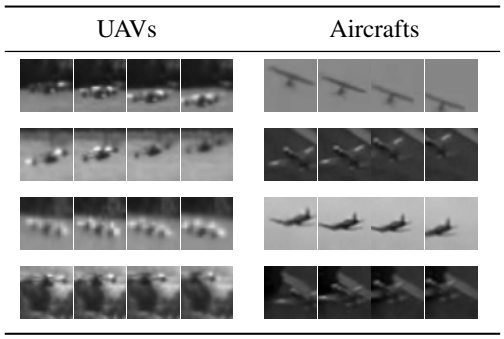


Figure 4: Sample patches of the UAVs and aircrafts. Each row corresponds to a single st-cube and illustrates different possible motions that an aircraft could have.

pooling layers apply max-pooling in 2D spatial regions only. The last layer is fully connected and outputs the probability that the input st-cube contains an object of interest. We use the hyperbolic tangent function as the non-linear operator [37].

We take the input of our CNN is a normalized st-cube

$$\eta = \frac{b - \mu(b)}{\sigma(b)}\,, \tag{2}$$

where $\mu(b)$ and $\sigma(b)$ are the mean and standard deviation of the pixel intensities in b, respectively. Normalization is an important step because network parameters optimization fails to converge when using raw image intensities.

During training, we write the probability that an st-cube η contains an object of interest ($y = 1$) or is a part of the background ($y = 0$) as

$$P(Y = y \mid \eta) = \frac{e^{\text{CNN}(\eta)[y]}}{e^{\text{CNN}(\eta)[0]} + e^{\text{CNN}(\eta)[1]}}, y = \{0, 1\}\,, \tag{3}$$

where $\text{CNN}(\eta)[y]$ denotes the classification score that the network predicts for η as being part of class y and $e^{(\cdot)}$ denotes the exponential function. We then minimize the negative log-likelihood

$$\mathcal{L}(W, bias) = -\sum_{k=1}^{N} \log P(Y = y_k \mid \eta_k) \tag{4}$$

with respect to the CNN parameters. Here (η_k, y_k) are pairs of normalized st-cubes and their corresponding labels from the training dataset, as defined in Section 3. To this end, we use the algorithm of [38] combined with Dropout [39] to improve generalization.

We tried many different network configurations, in terms of the number of filters per layer and the size of the filters. However, they all yield similar performance, which suggests that only minor improvements could be obtained by further tweaking the network. We also tried varying the dimensions of the st-cube. These variations have a more significant influence on performance, which will be evaluated in Section 5.

4 Motion Compensation

Neither of the two approaches to classifying st-cubes introduced in the previous section accounts for the fact that both the gradient orientations used to build the 3D HoG and the filter responses in the CNN case are biased by the global object motion. This makes the learning task much more difficult and we propose

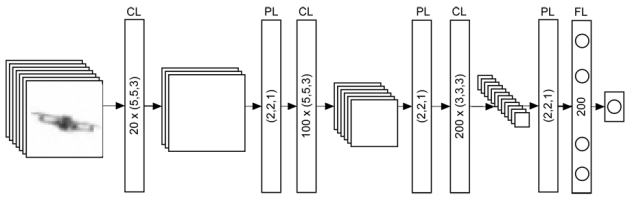


Figure 5: The structure of the Convolutional Neural Network, which we used for flying object detection. **CL**, **PL** and **FL** correspond to Convolution, Pooling, and Fully-connected layers respectively.

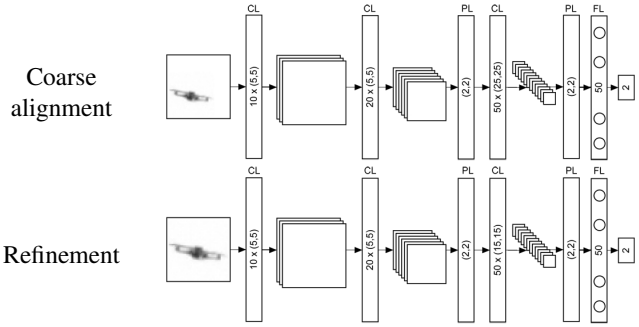


Figure 6: Structure of the CNNs used for motion compensation. (**Top**) The first network uses extended patches to correct for the large displacements of the aircraft. (**Bottom**) The second network is applied after rectification by the motion predicted by the first network, and is designed to correct for the small motions.

to use motion compensation to eliminate this problem. Motion compensation will allow us to accumulate visual evidence from multiple frames, without adding variation due to the object motion. We therefore aim at centering the target object, so that when present in an st-cube, it remains at the center of all its image patches.

More specifically, let I_t denote the t-th frame of the video sequence and (i, j) some pixel position in it. The st-cube $b_{i,j,t}$ is the 3D array of pixel intensities from images I_z with $z \in [t - s_t + 1, t]$ at image locations (k, l) with $k \in [i - s_x + 1, i]$ and $l \in [j - s_y + 1, j]$, as depicted by Fig. 4. Correcting for motion can be formulated as allowing patches $m_{i,j,z}, z \in [t - s_t + 1, t]$ of the st-cube to shift horizontally and vertically in individual images.

In [11], these shifts are computed using optical flow information, which has been shown to be effective for pedestrians occupying a large fraction of the patch and moving relatively slowly from one frame to the next. However, as can be seen in Fig. 4, these assumptions do not hold in our case and we will show in Section 6 that this negatively impacts performance. To overcome this difficulty, we introduce instead a learning-based approach to compensate for motion and keep the object in the center of the $m_{i,j,z}$ patches of the st-cube even when the target object's appearance changes drastically.

More specifically, we treat motion compensation problem as a regression task: given a single image patch, we want to predict the 2D translation that best centers the target object. By rectifying all the image patches in an st-cube with their predicted translation, we can then align the images of the object of interest together.

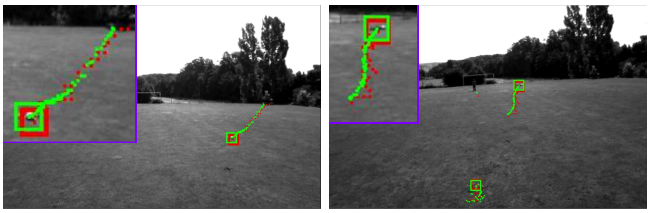


Figure 7: Combining multiple detections in several images of a video sequence. The red square and dots depict the positions of the original detection across the 50 frames preceding two different images. The green square and dots illustrate the position of the same detections after refinement. They are superposed and form much smoother trajectories. (best seen in color)

(a) UAV dataset (b) Aircraft dataset

Figure 8: Sample image patches containing aircrafts or UAVs from our datasets.

4.1 Boosted tree-based regressors

One way to predict the translation for an input patch m, is to train two different boosted trees regressors [40] $\phi_x(m)$ and $\phi_y(m)$, one for each 2D direction (horizontal and vertical).

As for detection, we use regression trees $h_j(m) = T(\theta_j, \text{HoG}(m))$ as weak learners, where $\text{HoG}(m)$ denotes the Histograms of Oriented Gradients for patch m. The difference is that we minimize here a quadratic loss function instead of an exponential one

$$L(r, \phi_*(m)) = (r - \phi_*(m))^2, \tag{5}$$

where m is the input patch, r the corresponding expected 2D vector, and $\phi_*(m) = [\phi_x(m), \phi_y(m)]^\top$ the 2D vector predicted by the 2 regression trees.

We then apply these regressors in an iterative way: we obtain a first estimate of the shift of the target object—if present—from the center of the patch. We translate it according to this estimate, and we re-apply the regressors. We iterate until both shift estimates drop to 0 or the algorithm reaches a preset number of iterations. In practice, 4 to 5 iterations are enough to achieve good accuracy.

4.2 CNN-based regressors

Another possible approach is to use a Convolutional Neural Network (CNN) to solve the regression task. CNNs are more flexible, as features are learned directly from the training data, in contrast to the hand-designed HoG features we need to use with our boosted tree-based regressors.

We trained two separate CNNs whose structure is depicted by Fig. 6. Note that there is no pooling layer after the first convolutional one. This is because pooling layers are typically used not only to reduce computational complexity but also to achieve invariance to small motions. In our case, such invariance would be counter-productive because these motions are precisely what we are trying to estimate. Furthermore, the computational complexity remains manageable even without the first pooling layer. We trained the first CNN using examples involving large 2D translations (coarse-CNN) and the second smaller ones (fine-CNN). In practice we use the latter to refine the predictions of the former. As when using boosted-trees, we use CNN-regressors iteratively until convergence, as described at the end of Section 4.1. We first correct for large displacements by applying several times coarse-CNN and we then apply fine-CNN, which is trained to compensate for small shifts of the object, for a couple more iterations.

In fact, we also tried training two different boosted-tree regressors such as those discussed in Section 4.1. Unlike in the case of the CNN regressors, it produced no significant improvement. This likely happens because our boosted trees motion compensation algorithm is based on HoG, where histograms are computed over the bins of fixed size. This, in fact, introduces invariance to small deviations of objects, which makes it hard to achieve high localization precision.

4.3 Motion Compensated st-cubes

Once the regressors have been trained, we use them to compensate for motion and build the st-cubes that we will use as input for classification, as depicted by Fig. 3. Fig. 2 illustrates several st-cubes of a drone from the testing dataset and after motion compensation, using either optical flow from [11] or our approach. Note that the latter tends to keep the target object much closer to the center, especially when the background is non-uniform and noisy or under lighting changes.

Part of the difficulty in detecting fast moving flying objects is that they can appear anywhere in the 3D environment and that their apparent size can vary enormously. This makes it necessary to scan the whole image at different scales using a sliding window to avoid missing anything, which is computationally expensive.

Fortunately, our motion compensation scheme frees us from the need to evaluate every image position. When there is a target object, our algorithm automatically shifts the patch so it is in the center. As a result, instead of having to test windows centered at every pixel location, we only have to check non-overlapping ones because the algorithm will automatically shift their location to center the target object when one is present. This also makes its unnecessary to use heuristics such as non-maximum suppression, as all the detections that arise from a single object will be shifted to the same position. The duplicates can therefore easily be removed, leaving us with just a single detection per object, as illustrated by Fig. 7.

As discussed in Section 3, we process each scale independently. We then perform non-maximum suppression in scale-space as a final step.

5 Designing the Optimal Approach

The two key components of our pipeline are motion compensation and classification of the st-cubes, both of which can be implemented using either CNNs or hand-designed features. In this section, we test the various possible combinations and justify the parameter choices we made for the final evaluation of our whole approach against several baselines, as described in Section 6.

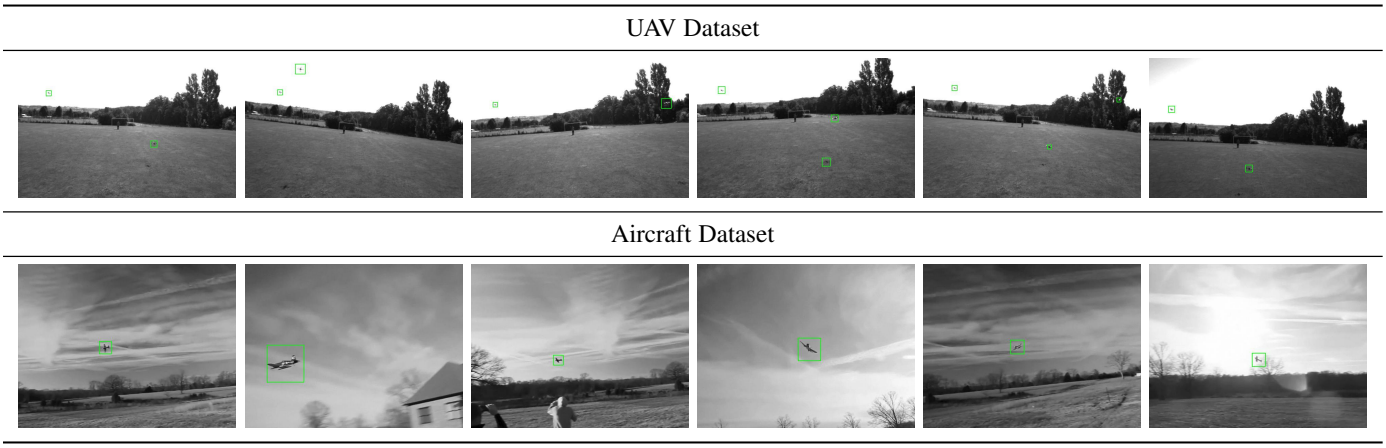


Figure 9: An object's apparent size can change enormously depending on its pose and distance to the camera. We therefore use a sliding window approach at different resolutions. The green boxes denote detections by our algorithm, which successfully handles background, lighting, scale, and pose changes.

Since the problem of detecting small flying objects has not yet received extensive attention from our community, there is not yet any standard dataset that can be used for testing purposes. We therefore built our own, one for UAVs and one for planes. We first describe them and then describe our testing protocol and the metrics we used for evaluation purposes. Finally, we perform the above-mentioned comparisons and demonstrate that the best results are obtained by using the CNN approach of Section 4.2 for motion compensation and the HoG3D descriptors of Section 3.1 for actual detection.

5.1 Datasets

To evaluate the performance of our approach, we built two separate datasets. They feature many real-world challenges including fast illumination changes and complex backgrounds, such as those created by moving treetops seen against a changing sky. They are as follows.

- **UAV dataset**. It comprises 20 video sequences of 4000 752×480 frames each on average. They were acquired by a camera mounted on a drone filming similar ones while flying indoors and outdoors. The outdoor sequences present a broad variety of lighting and weather conditions. All these videos contain up to two objects of the same category per frame. However, the shape of the drones is rarely perfectly visible and thus their appearance is extremely variable due to changing altitudes, lighting conditions, and even aliasing and color saturation due to their small apparent sizes. Fig. 8(a) illustrates some examples of the variety of appearance of a drone present in this dataset.
- **Aircraft dataset**. It consists of 20 publicly available videos of radio-controlled planes. Some videos were acquired by a camera on the ground and the rest was filmed by a camera on board of an aircraft. These videos vary in length from hundreds to thousands of frames and in resolution from 640×480 to 1280×720. Fig. 8(b) depicts the variety of plane types. The aircrafts may also appear under different angles, which makes the problem more complex. Fig. 9 shows some examples of the pose variation that a plane could have throughout the video sequence.

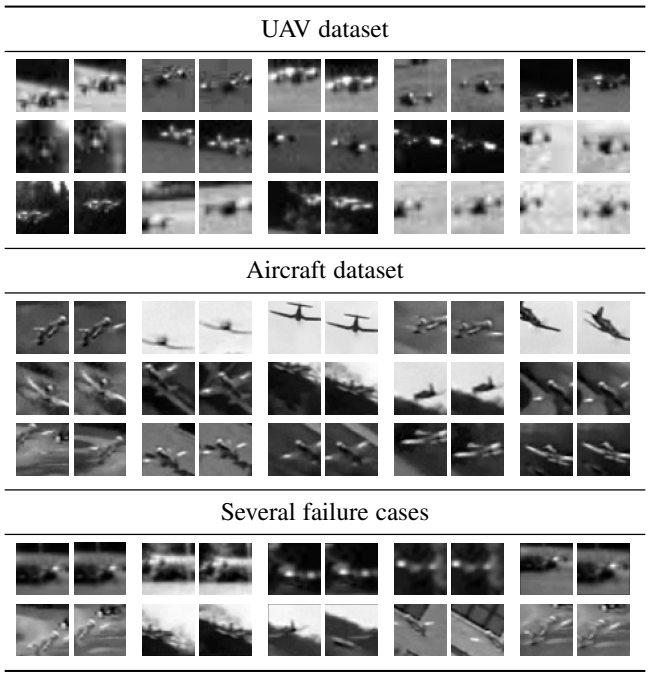
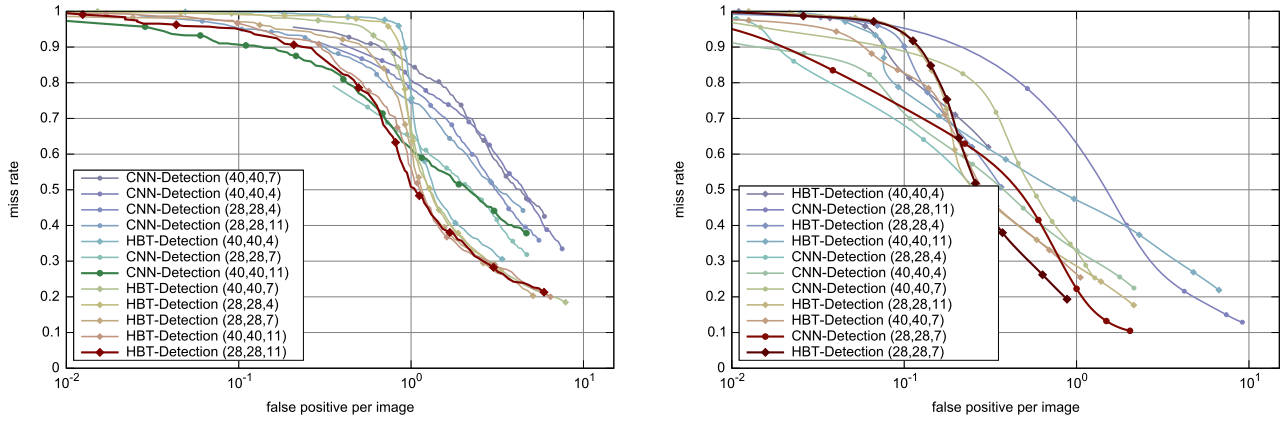


Figure 10: Examples of motion compensation. The first image in each pair shows the middle patch of the original st-cube, coming from the sliding window. The second image corresponds to the same patch after applying our motion compensation algorithm. Failure cases are often due to motion estimation failures, which happen when the appearance of the object is heavily corrupted by noise.

5.2 Training and Testing

In all cases, we used half of the data to train regressors and detectors. We manually supplied 8000 bounding boxes centered on a UAV and 4000 on a plane.

We used the Boosted trees implementation of [41] for both regression and detection. To compute the HoG3D and HoG descriptors, we used the publicly available implementations [14] and [42], respectively. We used Theano [17] to build the CNN models for both regression and detection tasks. In both of these cases we used the method described in [38] for optimization. The



UAV dataset Aircraft dataset

Figure 11: Influence of the st-cubes sizes on the performance of Boosted trees (HBT-Detection) and CNN (CNN-Detection) detectors with CNN-based motion compensation method, as described in Section 5.2.3. The plots are colored according $MR|_{FPPI=1}$ criterion (introduced in Section 5.2.2). Here blue corresponds to the higher $MR|_{FPPI=1}$, while red to the lower one. The darker lines on both plots correspond to the best performing examples of two different types of machine learning algorithms, according to the same criterion. The evaluation was performed on the validation subsets of the UAV and Aircraft datasets. (best seen in color)

structures of the CNNs for detection and motion compensation are depicted by Figs. 5 and 6 respectively. Here the parameters of each layer—the numbers of filters per layer and their dimensions—are given in the figures in the format $N \times (k_x, k_y, k_t)$, where N and (k_x, k_y, k_t) are the number of filters and their sizes respectively.

5.2.1 Training the Motion Regressors

To provide labeled examples where the aircraft or UAV is not in the center of the patch but still at least partially within it, we randomly shifted the ground truth bounding boxes by a translation of magnitude up to half of their sizes. This step was repeated for all the frames of the training database to cover the variety of shapes and backgrounds in front of which the aircraft might appear.

Applying large translations to the training data allows us to run the detection to only non-overlapping patches without missing the target, as explained at the end of Section 4.3. This procedure allows us to generate as much training data as needed for both Boosted trees (HBT-Regression) and CNN regressors (CNN-Regression), which is important for performance especially as the latter is known to require large amounts of training data.

The apparent size of the objects in the UAV and Aircraft datasets varies from 10 to 100 pixels. To train the regressor, we used 40×40 patches containing the UAV or aircraft shifted from the center.

The CNN-based regressor relies on convolutions of the original patch with filters from different network layers, which may produce artifacts close to the patch borders and degrade performance when the object is only partially visible. To reduce the influence of such artifacts, we extend the input patch by 25% in both the horizontal and vertical directions. This needs to be done only for the coarse alignment CNN, as depicted by the top row of Fig. 6. It is not required for the refinement CNN that only estimates small motions.

Fig. 10 depicts some examples of motion compensation. Note that even though both aircrafts and drones appear in front of changing backgrounds, the motion compensation algorithm correctly estimates the object location within the patch. Fig. 10 also illustrates some cases when the motion compensation system is unable to correctly predict the location of the object in the patch. This typically occurs when the patches are very noisy and the object is almost not visible.

To handle the wide range of flying objects apparent sizes, we use a multi-scale sliding window detector. Fig. 9 shows the same UAV and plane appearing at various distances from the camera throughout the video sequence.

5.2.2 Evaluation Metrics

In our experiments we consider an object to be correctly detected if there is 50% overlap between the detected bounding box and the ground-truth bounding box.

We report precision-recall curves. Precision is computed as the number of true positives detected by the algorithm divided by the total number of detections. Recall is the number of true positives divided by the number of the positive test examples. Additionally we use the *Average Precision* measure, which we take to be the integral $\int_0^1 p(r)dr$, where p is the precision, and r the recall.

We also report the log-average miss-rate (MR) with respect to the average number of false positive per image (FPPI). The miss-rate is computed as the number of true positives missed by the detector, divided by the total number of true positives; FPPI is computed as the total number of false positives, divided by the total number of images in the testing dataset:

$$\begin{aligned} MR &= 1 - \frac{N_d}{N_{tp}}, \\ FPPI &= \frac{N_{fd}}{N_f}, \end{aligned} \tag{6}$$

where N_d, N_{fd}, N_{tp}, N_f are the number of true and false detections, the number of positively labeled examples and the number of frames in the test set, respectively.

5.2.3 Motion Compensation Performance Analysis

Prior to evaluating the detection accuracy of the methods we need to apply motion compensation to the st-cubes. Thus we need to evaluate, which motion compensation method performs best. To this end, we created a validation dataset by selecting one video from each dataset. These videos are then used to generate data, using the method introduced in Section 5.2.1. We

use the validation set to tune the parameters and then perform the comparison against competing approaches on the test set.

We compare HBT-Regression and CNN-Regression in terms of Root Mean Square Error (RMSE). More formally, we are given a validation set of pairs $(X_i, S_i^a), i \in 1..N$, where X_i is a patch and $S_i^a \in \mathbb{R}^2$ corresponds to the true shift of the object from the center of the patch. Let also $S_i^p \in \mathbb{R}^2 : S_i^p = \phi(X_i)$ be the prediction of the shift of the object, obtained by the motion compensation system. Then the RMSE is computed using the following equation:

$$\mathtt{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(S_i^p - S_i^a)^2}\,. \tag{7}$$

Note that S_i^p and S_i^a do not depend on the size of the patch.

Table 1 depicts the results of this comparison. CNN-Regression outperforms HBT-Regression on both datasets. For reference we also provide $\mathtt{RMSE}_0$, which is computed as:

$$\mathtt{RMSE}_0 = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(S_i^a)^2}. \tag{8}$$

$\mathtt{RMSE}_0$ reflects the case when no motion compensation is applied.

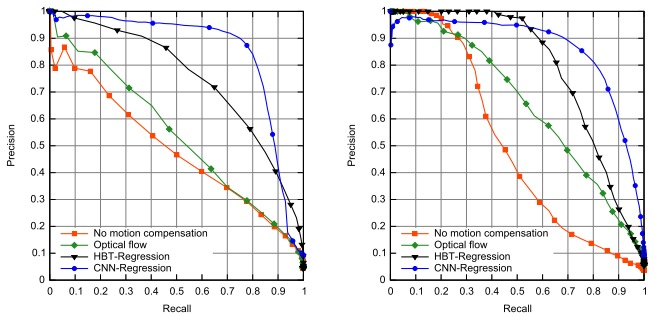
	RMSE	
method	UAV dataset	Aircraft dataset
No motion compensation ($\mathtt{RMSE}_0$)	0.1474	0.1451
HBT-Regression	0.0939	0.0805
CNN-Regression	**0.0669**	**0.0749**

Table 1: Performance of motion compensation methods. The valuation was performed on the validation subsets of the UAV and Aircraft datasets.

We therefore used the CNN-Regression algorithm to produce a number of aligned st-cubes of sizes ranging from $(s_x, s_y, s_t) = (28, 28, 4)$ to $(s_x, s_y, s_t) = (40, 40, 11)$, some of which we used for training and others for testing. For patches smaller than 40×40, we simply upscale them to 40×40 before applying the motion compensation regressors. The choice of s_t controls the trade-off between detecting far away objects using large values and closer ones using smaller ones. This is because, when the object is very close, the apparent motion may become too large for our motion compensation scheme. We found that increasing s_t beyond 11 did not bring any improvement in performance, while decreasing it below 4 left us with too little motion information.

As described above we have used the same video sequences to select the most appropriate size s_t for the st-cube. Fig. 11 summarizes our experiments, in terms of Average miss-rate curves. The legend of the plot describes the set-up used during the experiments. The number in brackets correspond to the (s_x, s_y, s_t) dimensions of the st-cube. The order of the curves in the legend is designed in the way that the highest curve is highest in terms of $MR|_{FPPI=1}$ measure. The lowest curve corresponds to the best performing set-up. For the different detection algorithms we show the best performing results by making the curves darker.

The classifiers of Section 3.1 rely on boosted trees operating on HoG3D descriptors [14]. We computed them using the default parameters, that is, 24 orientations per bin of size $4 \times 4 \times 2$ pixels. The Boosted trees detector uses 1500 trees of depth 2. We will further refer to this method as HBT-Detection.



(a) UAV dataset (b) Aircraft dataset

	Average Precision	
HBT-Detection detection algorithm, together with different motion compensation methods	UAV dataset	Aircraft dataset
No motion compensation	0.485	0.497
Optical flow	0.540	0.652
HBT-Regression	0.751	0.789
CNN-Regression	**0.849**	**0.864**

Figure 12: Comparison of motion compensation methods on the test subsets of our datasets. For all the motion compensation algorithms we have used the same HBT-Detection approach, as it proved to be more accurate, comparing to CNN-Detection. Unlike the optical flow-based algorithm, our regression-based ones properly identify the shift in object position and correct for it, even when the background is complex and the object outlines are barely visible. This yields a better precision/recall. Table in the bottom of the figure depicts the Average Precision score for the methods presented above.

For the CNNs of Section 3.2, we tried different network configurations, with variations of the number and size of filters in the convolutional layers and varying numbers of fully connected layers. In the end, they all ended up yielding very similar results. The final configuration that we used is illustrated by Fig. 5. We will refer to this method as CNN-Detection.

As depicted by Fig. 11, HBT and CNN detectors perform similarly on the plane dataset but the former clearly outperforms the latter on the UAV dataset when we allow a single false positive per frame on average. This may seem surprising but similar behaviors have been reported by [43] where the top four methods rely on decision forests while the Deep learning approach ranks only sixth. In our case, this may be attributable to the size of the training database not being large enough to take full advantage of the power of CNNs. Furthermore, for tasks that require as few false positives as possible, the CNNs win.

In any event, these experiments suggest that the optimal dimension of the st-cube depends on the task at hand. The apparent size of the UAVs is small, which favors large temporal dimension. As can be seen in Fig. 11(a), the best results are obtained for $s_t = 11$. By contrast, the Aircraft dataset comprises examples of planes flying at many different distances from the camera. In this case, $s_t = 7$ is optimal for both HoG3D descriptors and CNNs.

5.2.4 *Detection-Based Evaluation*

Another way to evaluate our motion compensation algorithm is to compare the detectors, trained on the data, processed with either HBT-Regression or CNN-Regression methods. This measures the

influence our motion compensation algorithm has on the accuracy of the detector, which is what we are interested in. We have chosen HBT-Detection method for detection task, as it is faster to train and it showed better accuracy on validation set, based on experiments, depicted by Fig. 11. We compared our two methods described in Section 4 with an optical flow based method [11], which is probably the best available.

Fig. 12 illustrates the results of this comparison. We also provide the performance of the same detector, trained and tested on the data without motion stabilization for reference.

Our methods are able to correctly compensate for the UAV motion even in the cases where the background is complex and the drone might not be visible due to image saturation and noise. Fig. 2(b,d) illustrates this hard situation with an example. On the contrary, the optical flow method is more focused on the background, which decreases its performance. Fig. 2(c) shows an example of a relatively easy situation, where the aircraft is clearly visible, but the optical flow algorithm fails to correctly compensate for its movement, while our regression-based approach succeeds.

Fig. 2(a) illustrates another situation, where the object is not in the center of the patch for the middle image of the st-cube. Optical flow methods will align other patches of the st-cube with respect to the middle one, which will result in object being shifted from the center in all the st-cube patches. By contrast, our motion compensation algorithm does not require any reference frame, leading to higher accuracy.

Using motion compensation for alignment of the st-cubes results into a higher performance of the detectors, as in-class variation of the data is decreased. Fig. 12 shows that we can achieve at least 15% improvement in average precision on both datasets using our motion compensation algorithm.

Our CNN-based motion compensation algorithm performs best. It yields about a 10% increase in accuracy, compared to the boosted trees method. Such difference in performance most likely lies in the nature of the features used by these machine learning techniques. The boosted trees regressor is using HoG features, which might not be perfectly suited for the problem, while the filters in the CNN are learned directly from the data. As the CNN obtains better accuracy, for our further experiments we will use the CNN-based motion compensation.

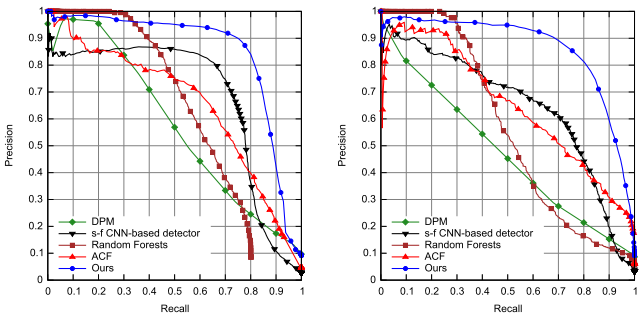
6 Comparing against Competing Methods

In this section, we compare the performance of the pipeline of Section 3, optimized as described in Section 5, against several state-of-the-art algorithms on the two challenging datasets introduced in Section 5.1. For these experiments, we therefore use st-cubes whose sizes are $(28, 28, 11)$ for UAVs and $(28, 28, 7)$ for planes, which are those we determined to yield the lowest miss-rates when we use HoG3D descriptors for detection and CNNs for motion compensation.

We first list the algorithms we use as baselines. and show that ours outperforms them consistently both for plane and UAV detection. We then demonstrate that motion compensation does not significantly degrade performance in cases when it is not strictly needed, such as when two aircrafts are on a collision course.

6.1 Baselines

To demonstrate the effectiveness of our approach, we compare it against state-of-the-art algorithms. We chose them to be representative of the three different ways the problem of detecting small moving objects can be approached, as discussed in Section 2.



UAV dataset Aircraft dataset

Figure 13: Comparing against appearance-based approaches [16], [17], [19], [44] in terms of precision/recall. For both the UAV and Aircraft datasets, the blue curve depicts our approach and is significantly above the others.

- **Appearance-Based Approaches** rely on detection in individual frames. We will compare against Deformable Part Models (DPM) [16], single-frame based Convolutional Neural Networks (s-f CNN-based detector) [17], Random Forests [44], and the Aggregate Channel Features method (ACF) [19], the latter being widely considered to be among the best.
  Since our algorithm considers st-cubes, for a fair comparison with these single-frame algorithms, we proceed as follows. Similarly to our approach we divide the video sequence into a set of N-frame overlapping slices. We further extract st-cubes using a sliding window approach, but motion compensation is not applied. We then run the single frame based detector on each of the patches of these st-cubes and consider the whole st-cube b as positive if the weighted average of scores of the patches in b is positive. We use a simple Gaussian kernel G centered on the middle frame of b as a weighting function. G is defined as $G = \exp\left(-(i - s_t/2)^2/2\sigma^2\right)$, where s_t is the filter size and σ is taken as $\sigma = 0.3((s_t - 1)/2 - 1) + 0.8$ as often done. We tried simply averaging over the detection scores of the set of patches in the b, but it resulted in lower accuracy, because the detectors tend to give a higher score to the middle frame, in which the object appears to be close to the patch center.
- **Motion-based Approaches** do not use any appearance information and rely purely on the correct estimation of the background motion. Among those we experimented with MultiCue background subtraction [21], [22] and large displacement optical flow [26].
- **Hybrid approaches** are closer in spirit to ours and correct for motion using image-flow. Among those, the one presented in [11] is the most recent we know of and the one we compare against. The main difference is that it relies on optical flow for motion compensation whereas we use CNNs. To ensure a fair comparison, we used the same patches to construct the st-cubes both for our method and to extract the features [11] requires.

For all the motion-based [21], [22], [26] and single-frame-based [16], [17], [19], [44] approaches, the code was downloaded from publicly available sources. In particular, for ACF and Random Forests, we used the toolboxes of [45] and [41]

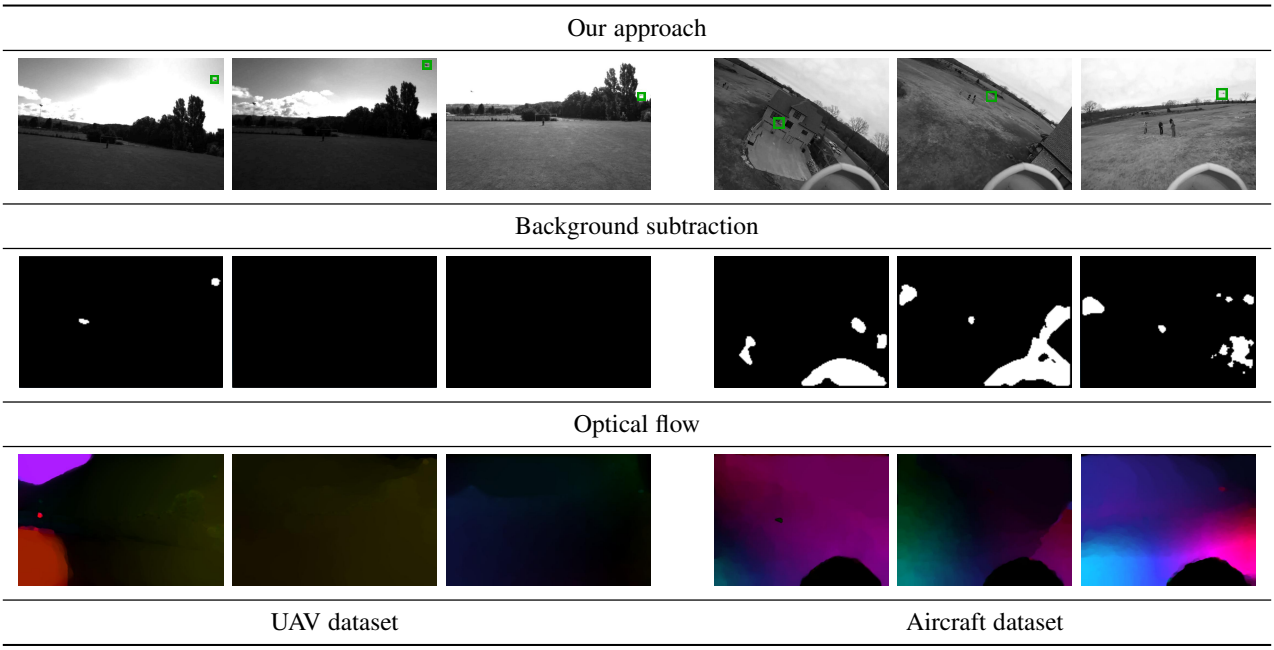


Figure 14: Comparing against motion-based methods [21], [26]. (**First row**) Our detector detects the objects by relying on motion and appearance, as evidenced by the green rectangles. (**Middle row**) Background subtraction results of [21]. Only in the leftmost frame of the three on the left, is there a blob that corresponds to a UAV, along with one that does not. Similarly, there is a small blob that corresponds to a plane in the central frame of the three right-most ones and many large ones in the others that do not clearly correspond to anything. (**Bottom row**) Optical flow computed using the algorithm of [26]. The plane and UAV generate a distinctly visible pattern in 2 or the 3 right-most images but in none of the three left-most ones. (best seen in color)

respectively. The DPM implementation is publicly available [16]. We also used the open source BGSLibrary [22] for state-of-the-art background subtraction. To compute features, we used default parameter configurations much as we did in our own pipeline for HoG3D. For algorithms relying on Random Forest, we tried varying the number of trees, and kept the number yielding the best results, again much as we did to find the best CNN configurations in our pipeline. For [11], we did not find a publicly available implementation and reimplemented the algorithm ourselves.

Method	Average Precision	
	UAV dataset	Aircraft dataset
Single-frame based approaches		
DPM [16]	0.573	0.470
Random Forests [44]	0.618	0.563
s-f CNN-based detector [17]	0.682	0.647
ACF [19]	0.652	0.648
Hybrid approaches		
Park [11]	0.568	0.705
Ours	**0.849**	**0.864**

Table 2: Average precision of detection methods on our datasets. We can see that in both cases our approach is able to reach higher detection accuracy. We achieve about 15% increase comparing to the best competing algorithms for the UAV and Aircraft datasets.

6.2 Evaluation against Competing Approaches

We used the same video sequences to train all the methods from the three classes described above. We compare here their results against ours.

6.2.1 Appearance-Based Methods.

In Fig. 13, we compare our method with appearance-based ones on our two datasets in terms of precision/recall. Table 2 summarizes the results in terms of Average Precision. For both the UAV and Aircraft datasets we improve on average by $15-20\%$ over ACF [19], which itself outperforms the others.

The CNN approach, provided by [17] yields scores comparable to those of the Random Forests and ACF methods. The structure of the network is the one depicted by Fig. 5, except for the fact that we replaced 3D convolutions by standard 2D ones. To boost CNN performance, we used Local Contrast Normalization (LCN) [46] after every convolutional layer and minimize the Hinge Loss at the final layer of the network, which was shown to be effective [47], [48].

The DPM [16] performs worst on average. This likely happens because it depends on using the correct size of the bins for HoG estimation, which makes it hard to generalize for a large variety of flying objects.

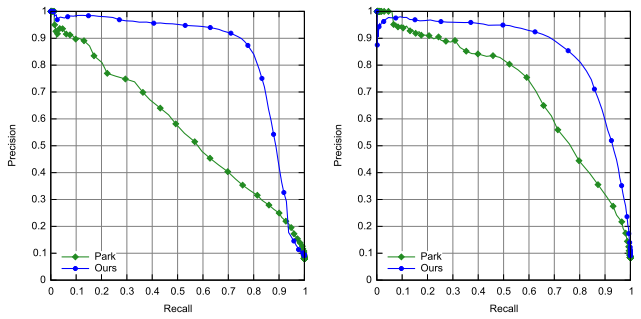
6.2.2 Motion-Based Methods

Fig. 14 depicts cases where background subtraction [21] and optical flow computation [26] algorithms, even though they are state-of-the-art, do not work well enough for detecting UAVs or planes in the challenging conditions we consider.

We did not compute precision-recall curves using these motion-based methods because it is unclear how big the moving part of the frame should be considered as an aircraft. We have tested several potential sizes and the resulting average precision values were much lower than those in Table 2 in all cases.

6.2.3 Hybrid approaches

In Fig. 15, we compare our method against the hybrid approach of [11], which relies on motion compensation using Lucas-Kanade



(a) UAV dataset (b) Aircraft dataset

Figure 15: Comparing against the hybrid method of [11]. Our approach performs better for both UAVs and Planes.

optical flow method, and yields state-of-the-art performance for pedestrian detection. As shown in Fig. 2, optical flow motion compensation cannot achieve good performance in our case, mostly because the target object is rather small and its appearance can significantly change due to illumination and background changes.

As a result, our regression-based approach allows to achieve higher performance for both the UAV and aircraft datasets. This suggests that accurate localization of the object in the patch is essential and leads to significant improvement in detection accuracy. Fig. 16 shows several frames to illustrate the performance of our approach.

6.3 Collision Courses

Motion compensation can be seen as a way to make the st-cube invariant from the motion of the aircraft, as it keeps flying object in the center, for all the patches of the st-cube. To evaluate whether enforcing this kind of invariance negatively impacts performance in the situations when it is not required, we applied our approach to the case of aircrafts on collision courses.

As shown in Fig. 17, if the aircraft A_1, observed from the camera of another aircraft A_2, is on collision course with A_2 then its behavior can be characterized by two important properties:

- A_1 remains at constant angle with respect to A_2
- the apparent size of A_1 increases from the point of view of A_2

These properties are invariant from the actual positions of the aircrafts in the 3D environment, the only constraint is that the paths of the aircrafts should intersect, which effectively means collision. In the scope of this paper only the first property is important, which means that motion stabilization is not needed, as A_1 will always occupy the same position in the image from the camera of A_2, provided A_1 and A_2 are on collision course.

We therefore searched publicly available sources for video sequences in which airplanes appear to be on a collision course for a substantial amount of time. We found fourteen, which vary in length from tens to several hundreds of frames. As before, we used half of them to train the detector and the others to test it.

In Fig. 18, we compare our results with and without motion stabilization. As expected, even though the non-stabilized results were poor in the general case, they are much better in this specific scenario. Incorporating motion stabilization very slightly degrades performance, which could be expected because enforcing *any* kind of invariance always loses some amount of information and is penalizing when such invariance is not required. However, in this case, the loss is almost negligible.

This is significant because, in a practical on-board system, detecting aircrafts on a collision course, which present a clear and immediate danger, would probably take priority over detecting all others. The former does not require motion compensation while the latter does. However, since nothing is lost by having motion compensation on, we can detect all aircrafts, whether on a collision course or not, without performance loss in the crucial case of those that are.

6.4 Scale Adjustment

As discussed in Section 4.3, we must run our detection scheme at different image resolutions to accommodate rapid size changes. This additional computational burden can be reduced by compensating not only for motion but also for size, which makes it possible to reduce the number of scales the system needs to check.

More specifically, we trained a regressor $\phi_{sc}(\cdot)$ to adjust for scale so that the bounding box fits the object of interest, much in the same way as we learned a regressor to compensate for motion. Fig. 19 illustrates this process in two separate cases. Note that in the case of Fig. 19(b), there were originally two different detections, which were collapsed into the same one after adjustment without having to perform non-maximum suppression.

Since CNNs have proved more effective for motion compensation than HoG based regressors, we used them to implement scale adjustment as well. We found out experimentally that using just a single patch to predict the true scale of the object is not enough. As in [49], we therefore used several scales as inputs to the CNN. Fig. 20 illustrates its structure.

The input to this CNN is a set of images of the object at different scales, which are provided as separate channels. Its output is the estimated scale of the object. Since there is no pooling layer after the first convolutional layer, we can estimate the scale with high precision. Furthermore, this CNN can be combined with the motion stabilization one of Section 4 to increase the accuracy of both motion compensation and scale adjustment. The structure of the resulting composite CNN is similar to the one depicted by Fig. 20. However, the output of its fully-connected layer has 3 floating point values instead of only 2. The first two are the shifts from the center of the patch in the spatial domain and the last one is the estimated scale. This replaces NMS in scale space, as described in Section 3, and yields precise object localization. Fig. 21 depicts some scale-adjustment results.

Table 3 compares the time required to process a single st-cube using our approach with and without scale adjustment. In this case, we have used st-cubes of size $(40, 40, 4)$ and 7 scales for the scale adjustment algorithm. Note that the number of scales can be selected with respect to the desired localization quality. Thus having many scales will yield more precise estimation of the object size, at the cost of a computation time increase. In our experiments we selected 7 scales, which results in high localization precision, as depicted by Fig. 22, while keeping the processing time relatively low. Even though adding scale adjustment to motion compensation increases the processing time per st-cube, it reduces the overall computation time by a factor of about 4. This is because it replaces the need of doing NMS across 7 different scales, which takes $0.123 * 7 = 0.861$ seconds, by processing one st-cube while accounting for scale, which takes 0.193 seconds.

In Table 4, we evaluate our approach on the UAV dataset with and without scale adjustment. Even though HBT-Detection with

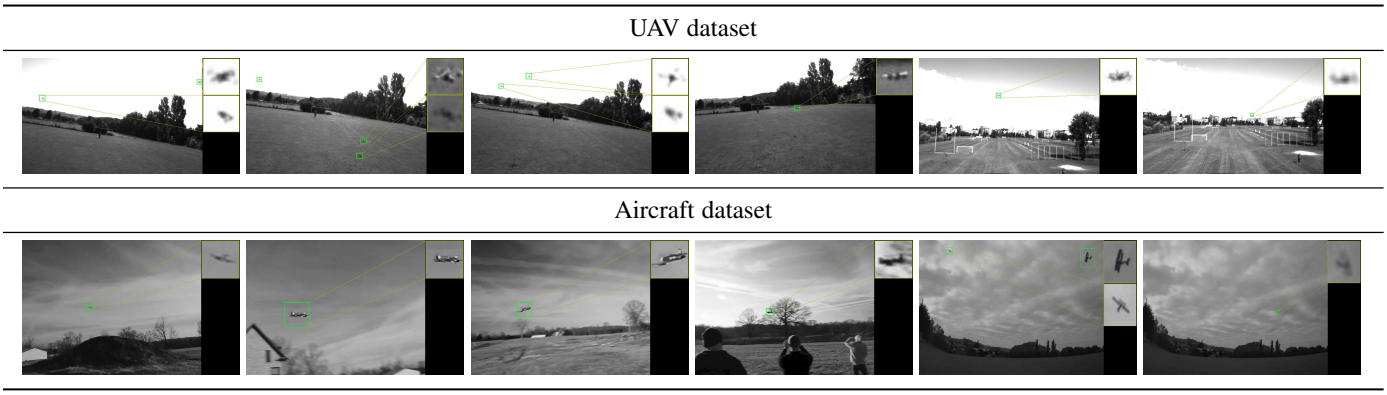


Figure 16: Some detection results. Thumbnails at the side of each figure show the zoomed-in versions of the detections made by our algorithm.

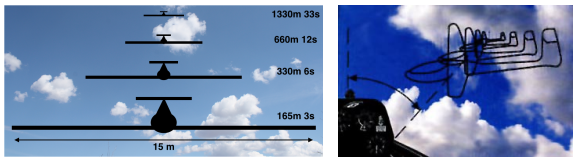
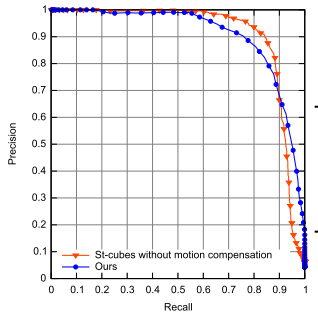


Figure 17: Collision courses. (**Left**) The apparent size of a standard glider and its 15 m wingspan flying towards another aircraft at a relatively slow speed (100 km/h) is very small 33s before impact, but the glider completely fills the field of view only half a minute later, 3s before impact. (**Right**) An aircraft on a collision course is seen in a constant direction but its apparent size grows, slowly at first and then faster.



st-cube	AveP (Average Precision)
W/o compensation	0.907
With compensation	0.904

Figure 18: Performance for aircrafts on a collision course. (**Left**) Precision/recall with and without motion compensation. (**Right**) Average Precision with and without motion compensation.

motion compensation + detection	0.123s
motion and scale adjustment + detection	0.193s

Table 3: Speed comparison of the motion and scale adjustment methods with motion compensation. We provide the time needed to process a single st-cube using an Intel® Xeon® CPU E5-2650 v2 running at 2.60GHz.

scale adjustment allows for faster computation, its performance is slightly lower than without scale adjustment. This is mainly due to the artifacts that appear when resizing small noisy images. Greater scale numbers improve detection accuracy at the cost of increased computation time.

In the experiments of Section 6.2, we rely on 50% overlap between detected and ground-truth bounding boxes. Thus, it is unnecessary to localize the target objects very precisely. We therefore use our method without scale adjustment on 8 distinctive scales, which yields a good balance between accuracy and computational time.

(a) (b)

Figure 19: Scale adjustment. The red bounding box shows the original detection and the green one the position adjusted for scale and motion. The thumbnails on the right are zoomed-in versions of the detections, with the top one illustrating the original detection and the bottom one showing the one after being motion and scale are adjusted. (best seen in color)

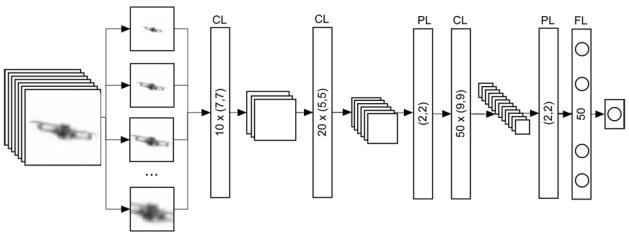


Figure 20: Structure of the scale adjustment Convolutional Neural Network. Several input channels contain object at different scales. The output of the CNN is a number, which characterizes the true scale of the object. 'CL' denotes a convolutional layer, 'PL' a pooling layer, and 'FL' a fully connected layer.

Fig. 22 illustrates the performance of our detection method in combination with motion compensation and scale adjustment. Our algorithm localizes the flying object with a great accuracy and yields trajectories that are smooth both in the spatial domain and in scale space. Provided that the camera is calibrated and given the true size of the object, we can estimate its distance to the camera, which is critical for collision avoidance purposes.

Different other examples that illustrate the performance of our motion compensation and detection approaches can be found at the

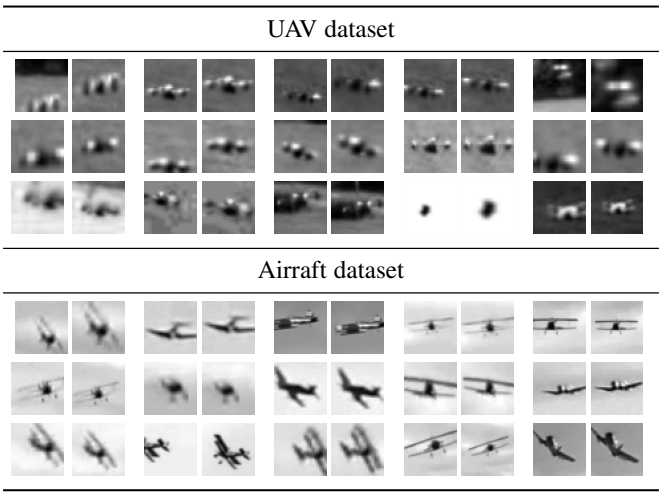


Figure 21: Sample results for simultaneous scale and motion compensation. The left image of each pair contains the original patch, where neither scale nor position are corrected. The right patch depicts the resulting patch after scale and motion correction.

Method:	UAV dataset	
	number of scales processed per frame	average miss-rate for FPPI $= 1$
HBT-Detection		
without scale adjustment	4	51%
without scale adjustment	8	50%
with scale adjustment	8	54%
with scale adjustment	16	52%
with scale adjustment	32	48%

Table 4: Evaluation of the HBT-Detection method on the UAV dataset with and without scale adjustment. Both method perform better when more scales are used, at the cost of increasing the computation time.

following link: http://cvlab.epfl.ch/research/unmanned/detection.

7 Conclusion

We showed that temporal information from a sequence of frames plays a vital role in detection of small fast moving objects like UAVs or aircrafts in complex outdoor environments. We therefore developed an object-centric learning-based motion compensation approach that is robust to changes in the appearance of both object and background. Both CNN and Boosted trees methods allow us to outperform state-of-the-art techniques on two challenging datasets. The CNN proved to be more suitable for motion compensation than the Boosted trees introduced in our previous work [15].

To evaluate our algorithms, we collected two challenging datasets for UAVs and Aircrafts detection. We hope that these datasets will become used as a new benchmark for improving flying objects detection and visual-based aerial collision avoidance.

8 Acknowledgments

This work was conducted in the context of the "Visual detection and tracking of flying objects in Unmanned Aerial Vehicles" project, funded by Honeywell International, Inc.

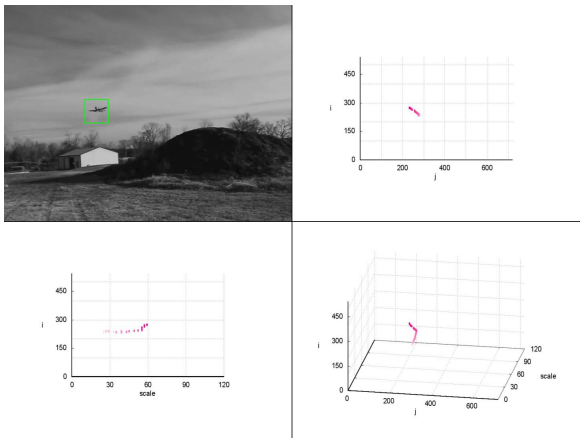


Figure 22: Precise estimation of the scale of the object allows us to localize it in 3-D space. (**Top Left**) Scale and motion adjusted detection of the aircraft in one frame of a video sequence. (**Top Right**) Projection of the points of the 3D trajectory throughout the previous 20 frames to the image plane. (**Bottom Left**) Changes of object scale. (**Bottom Right**) Trajectory of the object in 3D space is quite smooth due to the motion compensation algorithm, while neither tracking nor additional smoothing is applied.

References

[1] "Mercedes-Benz Intelligent Drive," http://techcenter.mercedes-benz.com/en/intelligent_drive/detail.html/.

[2] "Mobileeye Inc." http://us.mobileye.com/technology/.

[3] G. Conte and P. Doherty, "An Integrated UAV Navigation System Based on Aerial Image Matching," in *IEEE Aerospace Conference*, 2008, pp. 3142–3151.

[4] C. Martínez, I. F. Mondragón, M. Olivares-Méndez, and P. Campoy, "On-Board and Ground Visual Pose Estimation Techniques for UAV Control," *Journal of Intelligent and Robotic Systems*, vol. 61, no. 1-4, pp. 301–320, 2011.

[5] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "PIXHAWK: A System for Autonomous Flight Using Onboard Computer Vision," in *IEEE International Conference on Robotics and Automation*, 2011.

[6] C. Hane, C. Zach, J. Lim, A. Ranganathan, and M. Pollefeys, "Stereo Depth Map Fusion for Robot Navigation," in *Proceedings of International Conference on Intelligent Robots and Systems*, 2011, pp. 1618–1625.

[7] S. Weiss, M. Achtelik, S. Lynen, M. Achtelik, L. Kneip, M. Chli, and R. Siegwart, "Monocular Vision for Long-Term Micro Aerial Vehicle State Estimation: A Compendium," *Journal of Field Robotics*, vol. 30, pp. 803–831, 2013.

[8] S. Lynen, M. Achtelik, S. Weiss, M. Chli, and R. Siegwart, "A Robust and Modular Multi-Sensor Fusion Approach Applied to MAV Navigation," in *Conference on Intelligent Robots and Systems*, 2013.

[9] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast Semi-Direct Monocular Visual Odometry," in *International Conference on Robotics and Automation*, 2014.

[10] T. Zsedrovits, A. Zarándy, B. Vanek, T. Peni, J. Bokor, and T. Roska, "Visual Detection and Implementation Aspects of a UAV See and Avoid System," in *European Conference on Circuit Theory and Design*, 2011.

[11] D. Park, C. L. Zitnick, D. Ramanan, and P. Dollár, "Exploring Weak Stabilization for Motion Feature Extraction," in *Conference on Computer Vision and Pattern Recognition*, 2013.

[12] P. Dollar, V. Rabaud, G. Cottrell, and S. Belongie, "Behavior Recognition via Sparse Spatio-Temporal Features," in *VS-PETS*, October 2005, pp. 65–72.

[13] I. Laptev, "On Space-Time Interest Points," *International Journal of Computer Vision*, 2005.

[14] D. Weinland, M. Ozuysal, and P. Fua, "Making Action Recognition Robust to Occlusions and Viewpoint Changes," in *European Conference on Computer Vision*, 2010.

[15] A. Rozantsev, V. Lepetit, and P. Fua, "Flying Objects Detection from a Single Moving Camera," in *Conference on Computer Vision and Pattern Recognition*, 2015.

[16] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan, "Object Detection with Discriminatively Trained Part Based Models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010.

[17] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio, "Theano: new features and speed improvements," 2012.

[18] A. Bosch, A. Zisserman, and X. Munoz, "Image Classification Using Random Forests and Ferns," in *International Conference on Computer Vision*, 2007.

[19] P. Dollár, Z. Tu, P. Perona, and S. Belongie, "Integral Channel Features," in *British Machine Vision Conference*, 2009.

[20] N. Oliver, B. Rosario, and A. Pentland, "A Bayesian Computer Vision System for Modeling Human Interactions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 831–843, 2000.

[21] N. Seungjong and J. Moongu, "A New Framework for Background Subtraction Using Multiple Cues," in *Asian Conference on Computer Vision*. Springer Berlin Heidelberg, 2013, pp. 493–506.

[22] A. Sobral, "BGSLibrary: An OpenCV C++ Background Subtraction Library," in *IX Workshop de Visao Computacional*, 2013.

[23] D. Zamalieva and A. Yilmaz, "Background Subtraction for the Moving Camera: A Geometric Approach," *Computer Vision and Image Understanding*, vol. 127, pp. 73–85, 2014.

[24] T. Brox and J. Malik, "Object Segmentation by Long Term Analysis of Point Trajectories," in *European Conference on Computer Vision*, 2010, pp. 282–295.

[25] B. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," in *International Joint Conference on Artificial Intelligence*, 1981, pp. 674–679.

[26] T. Brox and J. Malik, "Large Displacement Optical Flow: Descriptor Matching in Variational Motion Estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011.

[27] Y. Zhang, S.-J. Kiselewich, W.-A. Bauson, and R. Hammoud, "Robust Moving Object Detection at Distance in the Visible Spectrum and Beyond Using a Moving Camera," in *Conference on Computer Vision and Pattern Recognition*, 2006.

[28] S.-W. Kim, K. Yun, K.-M. Yi, S.-J. Kim, and J.-Y. Choi, "Detection of Moving Objects with a Moving Camera Using Non-Panoramic Background Model," *Machine Vision and Applications*, vol. 24, pp. 1015–1028, 2013.

[29] S. Kwak, T. Lim, W. Nam, B. Han, and J. Han, "Generalized Background Subtraction Based on Hybrid Inference by Belief Propagation and Bayesian Filtering," in *International Conference on Computer Vision*, 2011, pp. 2174–2181.

[30] A. Elqursh and A. Elgammal, "Online Moving Camera Background Subtraction," in *European Conference on Computer Vision*, 2012, pp. 228–241.

[31] M. Narayana, A. Hanson, and E. Learned-miller, "Coherent Motion Segmentation in Moving Camera Videos Using Optical Flow Orientations," in *International Conference on Computer Vision*, 2013.

[32] A. Papazoglou and V. Ferrari, "Fast Object Segmentation in Unconstrained Video," in *International Conference on Computer Vision*, 2013.

[33] S. Walk, N. Majer, K. Schindler, and B. Schiele, "New Features and Insights for Pedestrian Detection," in *Conference on Computer Vision and Pattern Recognition*, 2010.

[34] J. Friedman, "Stochastic Gradient Boosting," *Computational Statistics & Data Analysis*, 2002.

[35] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in *Conference on Computer Vision and Pattern Recognition*, 2005.

[36] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *IEEE*, 1998.

[37] X. Glorot, A. Bordes, and Y. Bengio, "Deep Sparse Rectifier Neural Networks," in *International Conference on Artificial Intelligence and Statistics*, 2011.

[38] M. D. Zeiler, "ADADELTA: an Adaptive Learning Rate Method," *Computing Research Repository*, 2012.

[39] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[40] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, 2009.

[41] R. Sznitman, C. Becker, F. Fleuret, and P. Fua, "Fast Object Detection with Entropy-Driven Evaluation," in *Conference on Computer Vision and Pattern Recognition*, 2013, pp. 3270–3277.

[42] A. Vedaldi and B. Fulkerson, "VLFeat: An open and portable library of computer vision algorithms," http://www.vlfeat.org/, 2008.

[43] R. Benenson, O. Mohamed, J. Hosang, and B. Schiele, "Ten Years of Pedestrian Detection, What Have We Learned?" in *European Conference on Computer Vision*, 2014.

[44] L. Breiman, "Random Forests," *Machine Learning*, 2001.

[45] P. Dollár, "Piotr's Computer Vision Matlab Toolbox (PMT)," http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html.

[46] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the Best Multi-Stage Architecture for Object Recognition?" in *Conference on Computer Vision and Pattern Recognition*, 2009.

[47] J. Jin, K. Fu, and C. Zhang, "Traffic Sign Recognition with Hinge Loss Trained Convolutional Neural Networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, pp. 1991–2000, 2014.

[48] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial Transformer Networks," *arXiv Preprint*, 2015.

[49] M. Oberweger, P. Wohlhart, and V. Lepetit, "Hands Deep in Deep Learning for Hand Pose Estimation," *arXiv Preprint*, 2015.

Artem Rozantsev joined EPFL in 2012 as Ph.D. candidate at Computer Vision Laboratory, under the supervision of Prof. Pascal Fua and Prof. Vincent Lepetit. He received his Specialist degree in Mathematics and Computer Science in 2012 from Lomonosov Moscow State University. His research interests include object detection, synthetic data generation and machine learning.

Vincent Lepetit is a Professor at the Institute for Computer Graphics and Vision, TU Graz and a Visiting Professor at the Computer Vision Laboratory, EPFL. He received the PhD degree in Computer Vision in 2001 from the University of Nancy, France, after working in the ISA INRIA team. He then joined the Virtual Reality Lab at EPFL as a post-doctoral fellow and became a founding member of the Computer Vision Laboratory. He became a Professor at TU GRAZ in February 2014. His research interests include vision-based Augmented Reality, 3D camera tracking, Machine Learning, object recognition, and 3D reconstruction. He often serves as program committee member and area chair of major vision conferences (CVPR, ICCV, ECCV, ACCV, BMVC). He is an editor for the International Journal of Computer Vision (IJCV) and the Computer Vision and Image Understanding (CVIU) journal.

Pascal Fua received an engineering degree from Ecole Polytechnique, Paris, in 1984 and the Ph.D. degree in Computer Science from the University of Orsay in 1989. He then worked at SRI International and INRIA Sophia-Antipolis as a Computer Scientist. He joined EPFL in 1996 where he is now a Professor in the School of Computer and Communication Science and heads the Computer Vision Laboratory. His research interests include shape modeling and motion recovery from images, analysis of microscopy images, and Augmented Reality. He has (co)authored over 200 publications in refereed journals and conferences. He is an IEEE Fellow and has been an associate editor of IEEE journal Transactions for Pattern Analysis and Machine Intelligence. He often serves as program committee member, area chair, and program chair of major vision conferences.