# Active Structured Learning for High-Speed Object Detection

Christoph H. Lampert, Jan Peters

Max Planck Institute for Biological Cybernetics
Tübingen, Germany
{firstname.lastname}@tuebingen.mpg.de

**Abstract.** High-speed smooth and accurate visual tracking of objects in arbitrary, unstructured environments is essential for robotics and human motion analysis. However, building a system that can adapt to arbitrary objects and a wide range of lighting conditions is a challenging problem, especially if hard real-time constraints apply like in robotics scenarios. In this work, we introduce a method for learning a discriminative object tracking system based on the recent *structured regression* framework for object localization. Using a kernel function that allows fast evaluation on the GPU, the resulting system can process video streams at speed of 100 frames per second or more.
Consecutive frames in high speed video sequences are typically very redundant, and for training an object detection system, it is sufficient to have training labels from only a subset of all images. We propose an *active learning* method that select training examples in a data-driven way, thereby minimizing the required number of training labeling. Experiments on realistic data show that the active learning is superior to previously used methods for dataset subsampling for this task.

## 1 Introduction

Smooth high-speed tracking of arbitrary visual objects is essential in industrial automation, in many robot applications, e.g. visual servoing, high-speed ball games, and manipulation of dynamic objects in complex scenarios, as well as in a variety of other topics ranging from human motion analysis to automatic microscope operation. Due to the importance of the problem, many different solutions have been proposed both in academic research projects as well as for industrial applications.

Despite great progress in computer vision research, most solutions used in industry still rely upon controlled environmental conditions that can only be achieved on factory floors. Commercially available tracking solutions typically use active solutions such as pulsed LED's targtes or IR-reflecting markers. Basic research projects, on the other hand, have concentrated either on controlled setups with dark backgrounds, on complex marker patterns, or on systems that do not achieve pixel-exact tracking [1, 2]. Overall, tracking objects in semi-controlled, human inhabited environments at high frame rates with off-the-shelf hardware is still an open research problem. However, such components are essential in order to bring robots into human inhabited environments.

### 1.1 Object Tracking in Image Sequences

Most computer vision tracking technique, in particular *Kalman filters* [3] and *particle filters* [4], consists of two main components: a *detection step* that estimates an object's position in each individual frame and a *motion model* that performs temporal smoothing of the object trajectory, *e.g.* to suppress outliers. In probabilistic terms, the parts usually reflect a likelihood term and a prior.

In this work, we concentrate on the detection step: given an image from a sequence, we want to identify the location of a freely moving object with high accuracy and high speed. For maximal robustness, we avoid search space reductions, such as a *region of interest*, to be able to recover from misdetections without delay. For use in an interactive robotics system, the detections from two independent cameras are integrated in a Markov chain model, and 3D object trajectories are recovered, but these steps are beyond the scope of this paper.

### 1.2 Object Localization in Single Frames

To formalize the problem of object detection, or localization, we first introduce some notation. We treat images and object positions as random variables, denoting the image (the observed quantity) by $\mathbf{x}$, and the position of the object (the unknown quantity) by $\mathbf{y}$. *Object localization*, *i.e.* the task of predicting the object position from the image data, can be expressed as a *localization function*

$$f : \mathcal{X} \rightarrow \mathcal{Y}, \tag{1}$$

where $\mathcal{X}$ is the space of all images and $\mathcal{Y}$ is the space of possible object locations. For simplicity, we only treat the case where the output is parameterized by the object's center point in pixel coordinates, and where exactly one object location has to be predicted. Generalization to the possibility of predicting "no object", or multiple object locations are, of course, possible.

A huge number of possibilities to construct localizations functions have been proposed in the computer vision literature, either static model-based techniques, such as matched filters [5] and motion templates [6], or systems that learn from training examples, *e.g.* local classifiers [7, 8], voting procedures [9], mean-shift tracking [10], non-linear filters [11], and probabilistic random field models [12]. Most of these techniques are not applicable to our situation, because they are either not fast enough or do not achieve single pixel prediction accuracy. In this work, we build on *structured regression (SR)* [13], a flexible technique that treats Equation (1) as a (multidimensional) *regression problem*. Structured regression was originally introduced for performing object category localization with *bag of visual words* representations, thereby achieving strong invariance against within-class variation, but providing low spatial detection accuracy. In this work, we show how to adapt SR to the specifics of our problem, where accurate localization is crucial, because otherwise the subsequent stereo reconstruction breaks down. At the same time, we are able to work with a simpler object representation, because we only target the detection of specific objects, not of semantic object

classes. Variations in appearance therefore do not occur arbitrarily, but mainly due to varying illumination, a non-static background and partial occlusions.

In the following Section 2, we will recapitulate the concepts behind structured regression and explain our specific design choices. In Section 3, we introduce an improved training method based on active learning. Section 4 contains the experimental evaluation, where we show how active learning improves over other methods for efficient training, and Section 5 contains a summary of the paper and directions for future work .

## 2 Object Localization by Structured Regression

Structured regression in its general form consists of using a *structured support vector machine (S-SVM)* [14] to learn a kernelized *linear compatibility function*

$$F(x, y) = \langle w, \Phi(x, y) \rangle_{\mathcal{H}}, \tag{2}$$

where $\Phi : \mathcal{X} \times \mathcal{Y} \to \mathcal{H}$ is a joint feature function from $\mathcal{X}$ into a Hilbert space $\mathcal{H}$ that is implicitly given by a joint kernel function $k : (\mathcal{X} \times \mathcal{Y}) \times (\mathcal{X} \times \mathcal{Y}) \to \mathcal{H}$. From $F$ one obtains a *regression function* by maximization over the output space

$$f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} F(x, y). \tag{3}$$

For a fixed choice of kernel $k$, the function $f$ is completely determined by the weight vector $w \in \mathcal{H}$ that is obtained by solving an optimization problem [15]:

$$\min_{w \in \mathcal{H},\, \xi_1, \ldots, \xi_n \in \mathbb{R}^+} \quad \frac{1}{2}\|w\|^2 + \frac{C}{n} \sum\nolimits_{i=1}^{n} \xi_i \tag{4}$$

subject to weighted margin constraints[1] for $i = 1, \ldots, n$:

$$\forall y \in \mathcal{Y} \setminus \{y_i\} : \ \langle w, \Phi(x_i, y_i) \rangle_{\mathcal{H}} - \langle w, \Phi(x_i, y) \rangle_{\mathcal{H}}, \geq 1 - \frac{1}{\Delta(y_i, y)} \xi_i, \tag{5}$$

where $C > 0$ is a regularization parameter and $(x_1, y_1), \ldots, (x_n, y_n)$ are training examples, *i.e.* images $x_i$ with manually annotated correct object location $y_i$.

Remembering that $\langle w, \Phi(x, y) \rangle_{\mathcal{H}}$ is equal to the compatibility function $F(x, y)$, one sees that the optimization (4) is a maximum margin procedure: for each training image $x_i$, we would like to achieve a margin of 1 between the compatibility of the correct prediction $y_i$ to the compatibility of any other possible (and thereby suboptimal) prediction, *i.e.* $F(x_i, y_i) - F(x_i, y) \geq 1$ for $y \in \mathcal{Y} \setminus \{y_i\}$. The constraint set (5) expresses this fact, with two additions: each training image gets a slack variable $\xi_i$, because it might not be possible to fulfill all margin constraints simultaneously, and a weight function $\Delta(y_i, y)$ is introduced that reweights the slack variables to reflect the fact that in a regression setup some "wrong" predictions are less bad than others, and therefore not all slack variables should be penalized equally strong. $\Delta$ is also called the *loss function*, because it it proportional to the loss one has to pay in the objective function (4) when not achieving a sufficient margin for any of the training examples.

[1] In contrast to [13], we use the *slack rescaling* formulation of the S-SVM, which is generally considered more robust than the computationally easier *margin rescaling* [14].

### 2.1 S-SVM Training by Delayed Constraint Generation

The S-SVM training step is a *convex* optimization problem. Therefore, one can aim for finding the globally optimal solution vector without the risk of converging only to a local minimum. However, generic optimization packages do not handle the optimization well, because the number of constraints is extremely high: for each training instance, there are as many constraints as there possible object locations in the image. In order to derive a specialized solution procedure, Joachims [15] observed that only very few of the constraint will be active at the optimal solution, which allows the use of a *delayed constraint generation* technique. One iterates between solving (4) for a subset of constraints, which is nearly the same *quadratic program (QP)* as solving an ordinary SVM, and a verification step that checks if the resulting solution violates any element of the full constraint set (5). If it does not, one has found the globally optimal vector $w$. Otherwise, one adds one or several violated constraints to the constraint subset and restarts to the iteration[2]. Theoretic results guarantee only polynomial time convergence [14] of this procedure, but practical experience shows that typically only few iteration are required until the optimal solution is found, see *e.g.* [17].

In many applications, including structured regression, the time critical part of the S-SVM training procedure is not the QP solution, but the check for violated constraints. This step requires answering the following argmax problem

$$i^*, y^* = \operatorname*{argmax}_{i \in \{1,\ldots,n\}, y \in \mathcal{Y} \setminus \{y_i\}} \Delta(y, y_i) \left( 1 + \langle w, \Phi(x_i, y) \rangle - \langle w, \Phi(x_i, y_i) \rangle \right) . \quad (6)$$

Because $w$ is kept fixed in this expression, $\langle w, \Phi(x_i, y_i) \rangle$ is constant, and the maximization is nearly the same as Equation (3), except for an additional weighting by the loss function. Being able to solve (6) quickly is a crucial prerequisite to building an efficient S-SVM training procedure.

### 2.2 Fast Object Localization

S-SVM based structured regression can be adapted to localization problems of very different nature by choosing a suitable joint kernel function $k$ and loss function $\Delta$. For our system, the main requirements are *high spatial accuracy*, because the triangulation of $3D$ positions would otherwise fail, and *high speed at test time*, because the robotic system needs to operate in 100 Hz real-time. Our choice of $\Delta$ and $k$ reflects this: we use the *robust quadratic loss*

$$\Delta(y, y') = \min(\frac{1}{\sigma^2} \|y - y'\|^2_{L^2}, 1) \quad (7)$$

[2] In a computer vision context, the iterated algorithm is similar to *bootstrapping* methods for training object detection systems. These iteratively improve a detection function by searching for false positive detections and adding them as negative training examples [16]. S-SVM differs from this as it requires no *non-maximum suppression*, because the loss function allows arbitrary regions to be included instead of only false positives, and no *early stopping*, because the margin conditions prevent overfitting.

where $y$ encodes the object center in pixel coordinates and $\sigma$ is a tolerance parameter that we set to one third of the expected radius of the object. The locally quadratic part enforces high spatial accuracy, whereas the cutoff reflects that all predictions too far from the correct one are equally wrong, thereby making the measure robust to outliers.

Because the kernel enters the compatibility function (2), which is evaluated repeatedly at test time, we cannot afford expensive feature extraction steps like previous applications of S-SVMs to computer vision problems (*e.g.* [13, 18, 19]). Instead, we resort to an explicit kernel function

$$k\big(\,(x,y)\,,(x',y')\,\big) = \sum\nolimits_{(u,v)\in W} \phi\big(x,\, y+(u,v)\big)^t\, \phi\big(x',\, y'+(u,v)\big) \tag{8}$$

based on a per-pixel feature map $\phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^k$ where $W$ is a fixed shape region, *e.g.* a square, centered at relative coordinate $(0,0)$, such that $y+(u,v)$ runs over all positions of $W$ translated to the center point $y$. $\phi(x,y)$ can be vector valued with simplest choice $\phi(x,y) = (x_y^R, x_y^G, x_y^B)$, where $x_y^R, x_y^G, x_y^B$ are the values of the red, green and blue channel of the pixel at position $y$ in the image $x$. One can easily imagine more powerful representations, *e.g.* working in other color spaces, or using non-linear operations like *gamma correction*.

The reason we choose the kernel $k$ based on a per-pixel feature map is that it allows efficient inference, because it turns the operation of $w$ in Equation (2) into a *linear shift-invariant (LSI) filter* [20] on $x$. We write $k$ in Equation (8) as a linear kernel $k(\,(x,y)\,,(x',y')\,) = \Phi(x,y)^t\Phi(x',y')$ with explicit feature map

$$\Phi(x,y) = \Big(\,\phi\big(x,\, y+(u_1,v_1)\big)\;,\ldots,\;\phi\big(x,\, y+(u_s,v_s)\big)\Big) \tag{9}$$

for $W = \{(u_1,v_1),\ldots,(u_s,v_s)\}$, such that $\mathcal{H} = \mathbb{R}^K$ for $K = sk$. Decomposing $w$ into per-pixel contributions $w = (w_{(u_1,v_1)},\ldots,w_{(u_s,v_s)})$ in the same way as $\Phi$, we can rewrite the compatibility function as

$$F(x,y) = \langle w, \Phi(x,y)\rangle = \sum\nolimits_{c=1}^{k}\ \sum\nolimits_{i=1}^{s}\ w^c_{(u_i,v_i)}\phi^c\big(x, y+(u_i,v_i)\big) \tag{10}$$

where the index $c$ denotes the vector components of $\phi$. Writing $\hat{w}^c$ for the mirrored and padded pattern of $w^c$, *i.e.* $\hat{w}^c_{(x_i,y_i)} = w^c_{(-x_i,-y_i)}$ where defined, and $\hat{w}^c_{(x_i,y_i)} = 0$ elsewhere, we can write the inner sum as a 2D convolution

$$= \sum\nolimits_{c=1}^{k}\ [\hat{w}^c * \phi^c(x)](y) \tag{11}$$

where $\phi^c(x)$ denotes the $c$-th channels of the per-pixel feature representation of the whole image $x$. Now each summand in Equation (11) can be calculated efficiently even for large regions $W$ using the *convolution theorem* [20]. Denoting the Fourier transform by $\mathcal{F}$, we obtain

$$= \mathcal{F}^{-1}\Big(\sum\nolimits_{c=1}^{k}\ \mathcal{F}\hat{w}^c \odot \mathcal{F}\phi^c(x)\Big)[y] \tag{12}$$

where $\odot$ is the point-wise complex multiplication in Fourier space, and we were able to exchange the order of $\mathcal{F}^{-1}$ and the summation, because both are linear operations. The result is a score map of the same size as $x$, in which we can identify the argmax by a single scan through the elements.

The same trick allows us to speed up the training procedure, where we have to repeatedly solve Equation (6). After calculating the scalar product by the convolution theorem, multiplying with the loss function is just a point-wise operation, and we identify the argmax by scanning the array.

### 2.3 Implementation with GPU support

We implement the described S-SVM training procedure using the Python interface of `SVMstruct`[3]. Since training takes only seconds or few minutes, it is currently not a computational bottleneck for our object tracking system. Test time speed, however, is the crucial quantity we need to optimize for, because we have only milliseconds to evaluate the detection function (3) in a 100 Hz object detection system. We meet these requirement by calculating the Fourier transform on the GPU using the CUDA framework[4]. Using the FFT implementation provided by the CUDA SDK, the convolutions in (11) require less than $3ms$ to compute on an *NVIDIA GeForce GTX 280* graphics card.

## 3 Training with Active Learning

Structured regression provides us with a method to train an object detection system from a set of given training examples. However, because the training labels for high accuracy object localization need to be very accurate, ideally to the pixel level, creating training examples is a tedious task, and we would like to get away with as few labeled examples as possible.

To achieve this, we propose an *active learning* method, *i.e.* a setup in which the detector itself "decides" which images it would like to have labeled. While active learning is a well-established technique in the area of binary and multiclass classification [21], for the problem of structured prediction only perceptron-like classifiers have so far been studied in an active learning context [22]. This is despite the fact that because of its inherent sparsity, the S-SVM's is much better suited to this idea than the perceptron: due to the maximum-margin framework the optimal S-SVM solution vector will not depend on all training samples, but only on a subset of *support vectors*, which in our case are pairs of the training images $x_i$ with correct or incorrect labels $y \in \mathcal{Y}$. The set of support vectors is typically much smaller than the number of training instances, and particularly so for very redundant data sources like high framerate video streams. Therefore, it makes sense not to label all images of a sequence, but only some *relevant* ones. If we dropped only images that are not support vectors, we would still obtain exactly the optimal S-SVM solution. Unfortunately, the support vector are *a priori* unknown, so in practice, heuristics subsampling methods are used, *e.g.* labelling only every $k$-th frame, or labelling a random subset.

[3] `http://svmlight.joachims.org/svm_struct.html`
[4] `http://www.nvidia.com/object/cuda_home.html`

In this work, we instead propose the *active learning* setup illustrated in Algorithm 1. It can be seen as a generalization of the *delayed constraint generation* procedure [15]. Instead of only iteratively adding training regions for each image, we iteratively add the images themselves. For each working set of training examples, we train the S-SVM, and then sequentially classifying all available training images, including the unlabeled ones, until an *outlier* criterion is raised. If no outliers are found, the procedure terminates. Otherwise, we ask the user to label the first outlier image, add it to the training set and re-iterate until convergence. Note that $w$ is always a valid weight vector for object detection, so we also could interrupt the procedure at any time, *e.g.* after a fixed number of training examples.

**Require:** image sequence $x_1, \ldots, x_n$
  $S \leftarrow \emptyset$
  **repeat**
    $w \leftarrow$ S-SVM trained with $S$
    **for** $t = 1, \ldots, n$ **do**
      $\tilde{y}_t \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}} \langle w, \Phi(x_t, y) \rangle$
      **if** $\mathit{outlier}(\tilde{y}_t) \wedge (x_t, .) \notin S$ **then**
        ask for label $y_t$
        $S \leftarrow S \cup \{(x_t, y_t)\}$
        **break** from loop over $t$
      **end if**
    **end for**
  **until** *no outlier was detected.*

**Fig. 1.** Active S-SVM Training

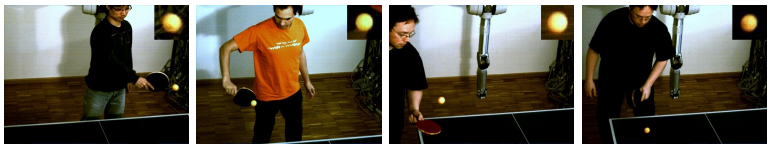

**Fig. 2.** Example frames from the image sequences with varying lighting conditions and players. The task is to detect the table tennis ball (enlarged in top right excerpts) that is of known size and color, but undergoes appearance changes due to non-homogeneous illumination conditions and occlusions.

The concept of an *outlier* serves as a proxy for a *mistake*, that would be the ideal criterion whether to include an image into the training set. However, to decide whether a predicted label differs from the correct label, we would require all images to be labeled, which is exactly what we want to avoid. A predicate *outlier*, in contrast, we can define by looking only at object detection in previous frames, using either a physical motion model, or a simper criterion like the distance between subsequent detections. Since in our practical experiments, all outlier criteria tested coincided almost perfectly with a true *mistakes* criterion, we settled for the simplest setup, declaring a prediction an outlier if its distance to the previous prediction is more than 4 object radii.

## 4 Experimental Evaluation

We show the performance of the proposed active learning setup on a realistic high-speed object detection task. With a static *Prosilica GE640C* Gigabit Ethernet camera, we captured sequences of people playing table tennis at a resolution

of $640 \times 480$ with 100 frames per second. Figure 2 shows example images from four different test sequences. The task consists of robustly detecting the position of the ball in each frame, for which we use a $33 \times 33$ rectangular region $W$.

From an pure object detection point of view, this task can be considered relatively easy, as the table tennis ball is a homogeneously textured spherical object of known color and size. However, there are numerous practical complications because we have to work in a human inhabited environments that we cannot fully control: different distractors may enter the image, *e.g.* due to peoples' different clothing. The image background can partly change due to people or objects entering and leaving the field of view. Additionally, a large window front causes strong variation in the lighting conditions that are non-homogeneous within the room and over time. Overall, classical non-adaptive method for *blob detection*, in particular *difference of Gaussians (DoG)* filters, have proven unreliable under these conditions, as can also be seen from the subsequent experiments.

Besides the DoG filter, we compare the proposed active learning setup with two frequently used baseline methods for creating reduced training sets: *uniform subsampling (H)* and *random subsampling (R)*. For all methods, we measure the detection accuracy on four video sequences consisting of 452, 505, 405 and 268 frames. For the trained methods, all frames not used as training samples are used for testing.

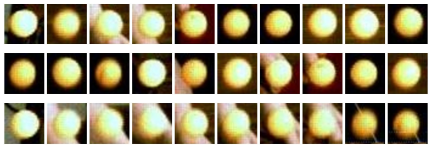

**Fig. 3.** First 10 training examples from *uniform* (top), and *random* (center) and *active learning* (bottom) selection. Active learning chooses more *difficult*, and thereby *informative*, training examples.

Figure 4 visualizes the detection performance of the three learning methods and the best performing different of Gaussian filter for the four sequence of Figure 2. As one can see, all trained method are able to learn a detector that is better than a predefined DoG filter, even when given only few training examples. We explain this by the fact that the appearance of the table tennis ball in the sequences we use is not rotationally symmetric due to asymmetric illumination conditions, and it is therefore not well modeled by Gaussian model. The plots also show that the *active* methods of example selection consistently requires fewer training examples to reach an acceptable level of accuracy than the other methods. Table 1 shows that one reason for this is that it produces much fewer strong outliers. The reason for this can be seen from Figure 3, which shows examples of the training sets resulting from the different selection strategies. Because the baseline methods select their training example regardless of their difficulty, they require labels for samples that are "easy" and unlikely to become support vectors anyway. Active learning adds mainly difficult example to the training set (*e.g.* the orange ball in front of varying amounts of skin color). Thus, the labels it requests are more likely to influence the decision function.

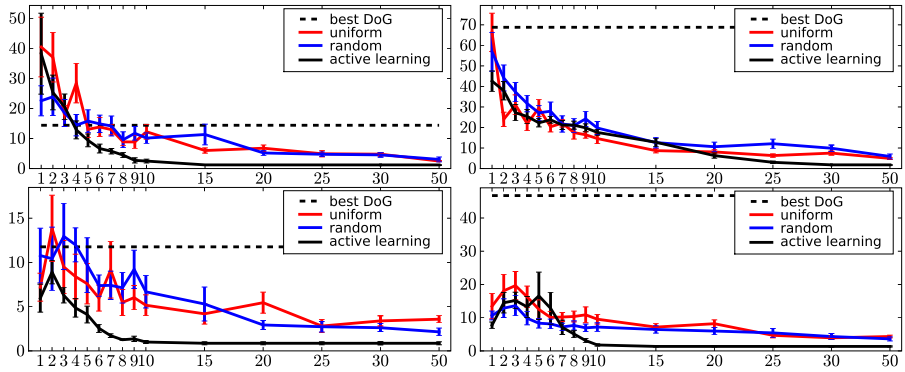


**Fig. 4.** Detection accuracy ($L^2$ distance between prediction and ground truth) against the number of training samples used. Each figure corresponds to one test sequences, and each data points depicts the mean and standard error over 10 runs with different start states.

| sequence | best DoG | random | uniform | active training |
|---|---|---|---|---|
| 1 | 0.24 | 0.15 / 0.10 / 0.02 | 0.12 / 0.11 / 0.02 | 0.08 / 0.02 / 0.01 |
| 2 | 0.84 | 0.26 / 0.19 / 0.08 | 0.28 / 0.15 / 0.07 | 0.21 / 0.15 / 0.02 |
| 3 | 0.11 | 0.08 / 0.06 / 0.02 | 0.07 / 0.05 / 0.04 | 0.05 / 0.01 / 0.01 |
| 4 | 0.53 | 0.07 / 0.06 / 0.03 | 0.13 / 0.08 / 0.03 | 0.13 / 0.01 / 0.01 |

**Table 1.** Fraction of outliers (defined by $\Delta(y_i, y_i^{pred}) = 1$) for the different detection methods at 5 / 10 / 50 training examples.

## 5 Summary and Future Work

We have presented a learning framework for efficient object detection. Using a structured regression setup, we showed how to construct the kernel function in a way that allows evaluation on the GPU, thereby achieving detection speed of more than 100 frames per second, where previous S-SVM based methods requires seconds or even minutes per test image [13, 18, 19]. We also extended the usual structured SVM training procedure to an *active learning* setup. By this we were able to also strongly reduce the number of labeled training examples necessary.

A strong advantage of the proposed systems is its flexibility. While we applied it in a relative straight-forward setting, working directly with the images' RGB components, other explicit per-pixel feature are easily integrated to yield more powerful classifiers. This includes elementary operations like *gamma correction* or color space transforms, but also non-linear features like *locally binary pattern*. Using temporal differences, one can incorporate *background subtraction*.

An interesting step in this direction that would also further increase the speed would be the use of the cameras' raw *Bayer pattern* as input features. A further direction of study would be the question if we can develop other kernel functions besides convolution-based ones that allow fast GPU-based evaluation.

**Acknowledgements**
This work was funded in part by the EU project CLASS, IST 027978.

## References

1. Hu, W., Tan, T., Wang, L., Maybank, S.: A survey on visual surveillance of object motion and behaviors. Systems, Man, and Cybernetics **34**(3) (2004)
2. Yilmaz, A., Javed, O., Shah, M.: Object tracking: A survey. ACM Computing Surveys **38**(4) (2006)
3. Kalman, R.E.: A new approach to linear filtering and prediction problems. Transaction of the ASME (1960)
4. Tanizaki, H.: Non-gaussian state-space modeling of nonstationary time series. J. Amer. Statist. Assoc. **82** (1987)
5. Tsatsanis, M.K., Giannakis, G.: Object detection and classification using matched filtering and higher-order statistics. In: Multidimensional Signal Processing. (1989)
6. Hager, G.D., Belhumeur, P.N.: Efficient region tracking with parametric models of geometry and illumination. IEEE Pattern Analysis and Machine Intelligence **20**(10) (1998)
7. Viola, P.A., Jones, M.J.: Robust real-time face detection. In: ICCV. (2001)
8. Grabner, H., Bischof, H.: On-line boosting and vision. In: CVPR. (2006)
9. Leibe, B., Leonardis, A., Schiele, B.: Robust object detection with interleaved categorization and segmentation. IJCV **77**(1) (2008)
10. Bajramovic, F., Graßl, C., Denzler, J.: Efficient combination of histograms for real-time tracking using mean-shift and trust-region optimization. In: DAGM. (2005)
11. Reisert, M., Burkhardt, H.: Equivariant holomorphic filters for contour denoising and rapid object detection. IEEE Image Processing **17**(2) (2008)
12. Shotton, J., Winn, J., Rother, C., Criminisi, A.: Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In: ECCV. (2006)
13. Blaschko, M.B., Lampert, C.H.: Learning to localize objects with structured output regression. In: ECCV. (2008)
14. Tsochantaridis, I., Joachims, T., Hofmann, T., Altun, Y.: Large margin methods for structured and interdependent output variables. JMLR **6**(2) (2006) 1453
15. Joachims, T., Finley, T., Yu, C.N.: Cutting-plane training of structural SVMs. Machine Learning (2009)
16. Rowley, H.A., Baluja, S., Kanade, T.: Neural network-based face detection. In: CVPR. (1996)
17. Joachims, T.: Training linear SVMs in linear time. In: KDD. (2006)
18. Szummer, M., Kohli, P., Hoiem, D.: Learning CRFs using graph cuts. In: ECCV. (2008)
19. Li, Y., Huttenlocher, D.P.: Learning for stereo vision using the structured support vector machine. In: CVPR. (2008)
20. Jähne, B.: Digital Image Processing. Springer (2005)
21. Cohn, D., Ghahramani, Z., Jordan, M.: Active Learning with Statistical Models. Journal of Artificial Intelligence Research **4** (1996) 129–145
22. Roth, D., Small, K.: Active learning with perceptron for structured output. In: ICML Workshop on Learning in Structured Output Spaces. (2006)