# Struck: Structured Output Tracking with Kernels

Sam Hare, Amir Saffari, Stuart Golodetz, Vibhav Vineet, Ming-Ming Cheng, Stephen L. Hicks and Philip H. S. Torr, *Senior Member, IEEE*

**Abstract**—Adaptive tracking-by-detection methods are widely used in computer vision for tracking arbitrary objects. Current approaches treat the tracking problem as a classification task and use online learning techniques to update the object model. However, for these updates to happen one needs to convert the estimated object position into a set of labelled training examples, and it is not clear how best to perform this intermediate step. Furthermore, the objective for the classifier (label prediction) is not explicitly coupled to the objective for the tracker (accurate estimation of object position). In this paper, we present a framework for adaptive visual object tracking based on structured output prediction. By explicitly allowing the output space to express the needs of the tracker, we are able to avoid the need for an intermediate classification step. Our method uses a kernelized structured output support vector machine (SVM), which is learned online to provide adaptive tracking. To allow for real-time application, we introduce a budgeting mechanism which prevents the unbounded growth in the number of support vectors which would otherwise occur during tracking. Experimentally, we show that our algorithm is able to outperform state-of-the-art trackers on various benchmark videos. Additionally, we show that we can easily incorporate additional features and kernels into our framework, which results in increased tracking performance. As a further contribution, we show that an optimised GPU implementation of Struck can be made to run at high frame-rates without sacrificing tracking performance.

**Index Terms**—tracking-by-detection, structured output SVMs, budget maintenance, GPU-based tracking

## 1 Introduction

Visual object tracking is one of the core problems of computer vision, with wide-ranging applications including human-computer interaction, surveillance and augmented reality, to name just a few. For other areas of computer vision which aim to perform higher-level tasks such as scene understanding and action recognition, object tracking provides an essential component.

For some applications, the object to be tracked is known in advance and it is possible to incorporate prior knowledge when designing the tracker. There are other cases, however, where it is desirable to be able to track arbitrary objects, which may only be specified at runtime. In these scenarios, the tracker must be able to model the appearance of the object on-the-fly and adapt this model during tracking to take into account changes caused by object motion, lighting conditions and occlusion. Even when prior information about the object is known, having a framework with the flexibility to adapt to appearance changes and incorporate new information during tracking is attractive, and in real-world scenarios is often essential for successful tracking.

An approach to tracking which has become particularly popular recently is tracking-by-detection [1], which treats the tracking problem as a detection task applied over time.

- *S. Hare was with the Department of Computing, Oxford Brookes University, and is now a co-founder of Obvious Engineering.*
- *A. Saffari was with Sony Europe Ltd., and is now with Affectv.*
- *S. Golodetz and S.L. Hicks are with the Nuffield Department of Clinical Neurosciences, and V. Vineet, M.-M. Cheng and P.H.S. Torr are with the Department of Engineering Science, University of Oxford.*

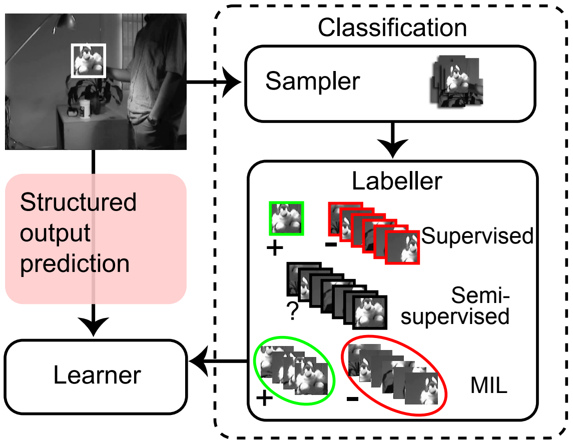


Fig. 1: Different adaptive tracking-by-detection paradigms: given the current estimated object location, traditional approaches (shown on the right-hand side) generate a set of samples and, depending on the type of learner, produce training labels. Our approach (left-hand side) avoids these steps and operates directly on the tracking output.

This popularity is due in part to the great deal of progress made recently in object detection, with many of the ideas being directly transferable to tracking. Another key factor is the development of methods which allow the classifiers used by these approaches to be trained online, providing a natural mechanism for adaptive tracking [2]–[4].

Adaptive tracking-by-detection approaches maintain a classifier trained online to distinguish the target object from its surrounding background. During tracking, this classifier is used to estimate object location by searching for the maximum classification score in a local region around the estimate from the previous frame, typically using a sliding-window approach. Given the estimated object location, traditional algorithms generate a set of binary labelled training samples with which to update the classifier online. As such, these algorithms separate the adaptation phase of the tracker into two distinct parts: (i) the generation and labelling of samples; and (ii) the updating of the classifier.

While widely used, this separation raises a number of issues. Firstly, it is necessary to design a strategy for generating and labelling samples, and it is not clear how this should be done in a principled manner. The usual approaches rely on predefined rules such as the distance of a sample from the estimated object location to decide whether a sample should be labelled positive or negative. Secondly, the objective for the classifier is to predict the binary label of a sample correctly, while the objective for the tracker is to estimate object location accurately. Because these two objectives are not explicitly coupled during learning, the assumption that the maximum classifier confidence corresponds to the best estimate of object location may not hold (a similar point was raised by Williams *et al.* [5]). State-of-the-art adaptive tracking-by-detection methods mainly focus on improving tracking performance by increasing the robustness of the classifier to poorly labelled samples resulting from this approach. Examples of this include using robust loss functions [6], [7], semi-supervised learning [8], [9], or multiple-instance learning [3], [10].

In this paper, we take a different approach and frame the overall tracking problem as one of structured output prediction, in which the task is to directly predict the change in object location between frames. We present a novel and principled adaptive tracking-by-detection framework which integrates the learning and tracking, avoiding the need for ad-hoc update strategies (see Figure 1).

Most recent tracking-by-detection approaches have used variants of online boosting-based classifiers [2]–[4]. In object detection, boosting has proved to be very successful for particular tasks, most notably face detection using the approach of Viola and Jones [11]. Elements of this approach, in particular the Haar-like feature representation, have become almost standard in tracking-by-detection research. The most successful research in object detection, however, has tended to make use of SVMs rather than boosting, due to their good generalisation ability, robustness to label noise, and flexibility in object representation through the use of kernels [12]–[14]. Because of this flexibility of SVMs and their natural generalisation to structured output spaces, we make use of the structured output SVM framework of Tsochantaridis *et al.* [15]. In particular, we extend the online structured output SVM learning method proposed by Bordes *et al.* [16], [17] and adapt it to the task of adaptive object tracking. We find experimentally that the use of our framework results in large performance gains over state-of-the-art tracking-by-detection approaches.

A structured output SVM framework has previously been applied to the task of object detection by Blaschko and Lampert [12]. In contrast to their work, in our setting there is no offline labelled data available for training (except the first frame, which is assumed to be annotated) and instead online learning is used. However, online learning with kernels suffers from the *curse of kernelisation*, whereby the number of support vectors increases with the amount of training data. Therefore, in order to allow for real-time operation, there is a need to control the number of support vectors. Recently, approaches have been proposed for online learning of classification SVMs on a fixed budget [18], [19], meaning that the number of support vectors is constrained to remain within a specified limit. We apply similar ideas in this paper and introduce a novel approach for budgeting which is suitable for use in an online structured output SVM framework. We find empirically that the introduction of a budget brings large gains in terms of computational efficiency, without impacting significantly on the tracking performance of our system.

An earlier version of this paper appeared in [20]. We extend it here in the following ways:

1) We perform additional experiments to quantify the effects of structured learning in comparison with a baseline classification SVM (§4.2).
2) We show how Struck can be implemented on the GPU and demonstrate how high frame-rates can be achieved without sacrificing tracking performance (§5).
3) We discuss recent benchmarks in which Struck achieved state-of-the-art performance (§6).

**This paper is organised as follows**: in §2, we briefly review related work; in §3, we describe the Struck tracker; in §4, we perform experiments to compare Struck to other trackers, and evaluate the effects on performance of structured learning and kernel combination; in §5, we present our GPU implementation of Struck; in §6 we discuss third-party tracking benchmarks and in §7 we conclude.

## 2 Related Work

Due to the importance of the tracking problem, a wide variety of different approaches have been proposed to solve it over the years. Whilst a comprehensive review of tracking techniques is beyond the scope of this paper, we direct the reader to [21] for a survey, and also to [22], [23] for some benchmarks that compare a significant number of trackers on large datasets (see also §6). We focus here on a representative selection of recent trackers.

*Dictionary-based trackers* maintain dictionaries of object templates and aim to represent candidate object regions in a new frame using combinations of these templates. A popular idea is to try and represent the candidates sparsely using $\ell_1$-norm minimization [24]–[26]. Prediction from one frame to the next is often done using particle filtering [27], with a sensor model that assigns higher confidence to candidates that are more easily represented by the templates. For example, Xing *et al.* [26] describe an approach that combines short-term, medium-term and long-term dictionaries to achieve a compromise between adaptivity (the short-term dictionary will adapt more quickly to new data) and robustness (the long-term dictionary will remember more of what the object originally looked like). Wang *et al.* [25] describe another interesting dictionary-based approach that aims to learn templates that capture distinct aspects of the object.

*Ensemble-based trackers* combine the results of a set of individual 'weak' classifiers to form a strong classifier that can be used to predict an object's bounding box in a new frame. For example, Cao and Xue [28] describe an adaptive random forest method that maintains a collection of candidate decision trees and picks half of them each frame to form an ensemble. (Other techniques that incorporate random forests can be found in [29], [30].) Bai *et al.* [31] combine weak classifiers trained on $8 \times 8$ patches within the object's bounding box using weight vectors sampled from a Dirichlet distribution that is updated over time. Wang *et al.* [32] show how to combine an ensemble of different trackers (including Struck) using a conditional particle filter approach to try and meld the best features of the trackers.

*Segmentation-based trackers* [33]–[35] actually segment (possibly coarsely) the object being tracked in each frame, so as to try and avoid the problem of drift that occurs when trackers inadvertently incorporate parts of the background in their object representation. For example, Duffner and Garcia [33] describe PixelTrack, an approach that co-trains a probabilistic segmentation model alongside a pixel-based Hough model so as to better handle non-rigid deformations of the tracked object between frames.

*Circulant trackers* are an interesting recent type of tracker that exploit the circulant structure of adjacent sub-windows in an image to achieve extremely fast tracking. The original such tracker, CSK [36], works by evaluating a classifier trained using kernel regularised least squares (KRLS) quickly at all sub-windows around the estimated target location and maximising the response. Danelljan *et al.* [37] build on this by introducing colour attributes to achieve superior performance on colour sequences.

A number of trackers we survey do not fall into any of the above categories. For example, Pernici and Del Bimbo [38] describe a tracker called ALIEN based on Nearest Neighbour classifiers that tracks using an oriented rather than axis-aligned bounding box, handles occlusions well and is designed for long-term tracking. Lu *et al.* [39] describe an interesting approach based on And-Or graphs that achieves good tracking performance in exchange for some speed. Finally, Zhang and van der Maaten [40] describe an appealing multi-object tracker based on structured SVMs that can be co-opted for single-object, part-based tracking.

Whilst some of the trackers we surveyed achieve better tracking performance than Struck on the video sequences they tested, there remain trade-offs involved, e.g. in terms of speed [31], [32], [39], difficulties with fast motion [32] or dependence on colour features [37]. As a result, we believe that our GPU-based implementation of Struck, which achieves high frame-rates, copes comparatively well with fast motion and works equally well on either greyscale or colour sequences, is state-of-the-art.

## 3 Online Structured Output Tracking

### 3.1 Tracking by detection

In this section we provide an overview of traditional adaptive tracking-by-detection algorithms, which attempt to learn a classifier to distinguish a target object from its local background.

Typically, the tracker maintains an estimate of the position $\mathbf{p} \in \mathcal{P}$ of a 2D bounding box containing the target object within a frame of a video sequence $\mathbf{f}_t \in \mathcal{F}$, where $t = 1, \ldots, T$ is the time. Given a bounding box position $\mathbf{p}$, a classifier is applied to features extracted from an image patch within the bounding box $\mathbf{x}_t^{\mathbf{P}} \in \mathcal{X}$. The classifier is trained with example pairs $(\mathbf{x}, z)$, where $z = \pm 1$ is a binary label, and makes its predictions according to $\hat{z} = \text{sign}(h(\mathbf{x}))$, where $h : \mathcal{X} \to \mathbb{R}$ is the classification confidence function.

During tracking, it is assumed that a change in position of the target can be estimated by maximising $h$ in a local region around the position in the previous frame. Let $\mathbf{p}_{t-1}$ be the estimated bounding box at time $t-1$. The objective for the tracker is to estimate a transformation (*e.g.* translation) $\mathbf{y}_t \in \mathcal{Y}$ such that the new position of the object is approximated by the composition $\mathbf{p}_t = \mathbf{p}_{t-1} \circ \mathbf{y}_t$. $\mathcal{Y}$ denotes our *search space* and its form depends on the type of motion to be tracked. For most tracking-by-detection approaches, this is 2D translation, in which case $\mathcal{Y} = \{(\Delta u, \Delta v) \mid \Delta u^2 + \Delta v^2 < r^2\}$, where $r$ is a search radius. In this case, the composition $\mathbf{p}_t = \mathbf{p}_{t-1} \circ \mathbf{y}_t$ is given by $(u_t, v_t) = (u_{t-1}, v_{t-1}) + (\Delta u, \Delta v)$.

Mathematically, an estimate is found for the change in position relative to the previous frame according to

$$\mathbf{y}_t = \underset{\mathbf{y} \in \mathcal{Y}}{\text{argmax}} \; h(\mathbf{x}_t^{\mathbf{p}_{t-1} \circ \mathbf{y}}), \tag{1}$$

and the tracker position is updated as $\mathbf{p}_t = \mathbf{p}_{t-1} \circ \mathbf{y}_t$.

After estimating the new object position, a set of training examples from the current frame is generated. We separate this process into two components: the *sampler* and the *labeller*. The sampler generates a set of $n$ different transformations $\{\mathbf{y}_t^1, \ldots, \mathbf{y}_t^n\}$, resulting in a set of training examples $\{\mathbf{x}_t^{\mathbf{p}_t \circ \mathbf{y}_t^1}, \ldots, \mathbf{x}_t^{\mathbf{p}_t \circ \mathbf{y}_t^n}\}$. After this process, depending on the classifier type, the labeller chooses labels $\{z_t^1, \ldots, z_t^n\}$ for these training examples. Finally, the classifier is updated using these training examples and labels.

There are a number of issues which are raised by this approach to tracking. Firstly, the assumption made in (1) that the classification confidence function provides an accurate estimate of object position is not explicitly incorporated into the learning algorithm, since the classifier is trained only with binary examples and has no information about transformations. Secondly, examples used for training the classifier are all equally weighted, meaning that a negative example which overlaps significantly with the tracker bounding box is treated the same as one which overlaps very little. One implication of this is that slight inaccuracy during tracking can lead to poorly labelled examples, which are likely to reduce the accuracy of the classifier, in turn leading to further tracking inaccuracy. Thirdly, the labeller is usually chosen based on intuitions and heuristics, rather than having a tight coupling with the classifier. Mistakes made by the labeller manifest themselves as *label noise*, and many current state-of-the-art approaches try to mitigate this problem by using robust loss functions [6], [7], semi-supervised learning [8], [9], or multiple-instance learning [3], [10]. We argue that all of these techniques, though justified in increasing the robustness of the classifier to label noise, are not addressing the real problem, which stems from separating the labeller from the learner. The algorithm which we present does not depend on a labeller and tries to overcome all these problems within a coherent framework by directly linking the learning to tracking and avoiding an artificial binarisation step. Sample selection is fully controlled by the learner itself, and relationships between samples such as their relative similarity are taken into account during learning.

To conclude this section, we describe how a conventional labeller works, as this provides further insight into our algorithm. Traditional labellers use a *transformation similarity function* to determine the label of a sample positioned at $\mathbf{p}_t \circ \mathbf{y}_t^i$. This function can be expressed as $s_{\mathbf{p}_t}(\mathbf{y}_t^i, \mathbf{y}_t^j) \in \mathbb{R}$ which, given a reference position $\mathbf{p}_t$ and two transformations $\mathbf{y}_t^i$ and $\mathbf{y}_t^j$, determines how similar the resulting samples are. For example, the overlap function defined by

$$s^o_{\mathbf{p}_t}(\mathbf{y}_t^i, \mathbf{y}_t^j) = \frac{(\mathbf{p}_t \circ \mathbf{y}_t^i) \cap (\mathbf{p}_t \circ \mathbf{y}_t^j)}{(\mathbf{p}_t \circ \mathbf{y}_t^i) \cup (\mathbf{p}_t \circ \mathbf{y}_t^j)} \tag{2}$$

measures the degree of overlap between two bounding boxes. Another example of such a function is based on the distance of two transformations $s^d_{\mathbf{p}_t}(\mathbf{y}_t^i, \mathbf{y}_t^j) = -d(\mathbf{y}_t^i, \mathbf{y}_t^j)$.

Let $\mathbf{y}^0$ denote the identity (or null) transformation, *i.e.* $\mathbf{p} = \mathbf{p} \circ \mathbf{y}^0$. Given a transformation similarity function, the labeller determines the label $z_t^i$ of a sample generated by transformation $\mathbf{y}_t^i$ by applying a *labelling function* $z_t^i = \ell(s_{\mathbf{p}_t}(\mathbf{y}^0, \mathbf{y}_t^i))$. Most commonly, this can be expressed as

$$\ell(s_{\mathbf{p}_t}(\mathbf{y}^0, \mathbf{y}_t^i)) = \begin{cases} +1 & \text{for} \quad s_{\mathbf{p}_t}(\mathbf{y}^0, \mathbf{y}_t^i) \geq \theta_u \\ -1 & \text{for} \quad s_{\mathbf{p}_t}(\mathbf{y}^0, \mathbf{y}_t^i) < \theta_l \\ 0 & \text{for} \quad \text{otherwise} \end{cases} \tag{3}$$

where $\theta_u$ and $\theta_l$ are upper and lower thresholds, respectively. A binary classifier generally ignores the unlabelled examples [2], while those based on semi-supervised learning use them in their update phase [8], [9]. In approaches based on multiple-instance learning [3], [10], the labeller collects all the positive examples in a *bag* and assigns a positive label to the bag instead. Most, if not all, variants of adaptive tracking-by-detection algorithms use a labeller which can be expressed in a similar fashion. However, it is not clear how the labelling parameters (*e.g.* the thresholds $\theta_u$ and $\theta_l$ in the previous example) should be estimated in an online learning framework. Additionally, such heuristic approaches are often prone to noise and it is not clear why such a function is in fact suitable for tracking. In the subsequent section, we will derive our algorithm based on a structured output approach which fundamentally addresses these issues and can be thought of as a generalisation of these heuristic methods.

### 3.2 Structured output SVM

Rather than learning a classifier, we propose learning a prediction function $f : \mathcal{X} \to \mathcal{Y}$ to directly estimate the object transformation between frames. Our output space is thus the space of all transformations $\mathcal{Y}$ instead of the binary labels $\pm 1$. In our approach, a labelled example is a pair $(\mathbf{x}, \mathbf{y})$ where $\mathbf{y}$ is the desired transformation of the target. We learn $f$ in a structured output SVM framework [12], [15], which introduces a discriminant function $g : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ that can be used for prediction according to

$$\mathbf{y}_t = f(\mathbf{x}_t^{\mathbf{p}_{t-1}}) = \underset{\mathbf{y} \in \mathcal{Y}}{\operatorname{argmax}} \; g(\mathbf{x}_t^{\mathbf{p}_{t-1}}, \mathbf{y}). \tag{4}$$

Note the similarity between (4) and (1): we are still performing a maximisation step in order to predict the object transformation, but the discriminant function $g$ now includes the label $\mathbf{y}$ explicitly, meaning it can be incorporated into the learning algorithm. In our framework, rather than using the tracker position to generate binary examples for training a classifier, we instead provide the single labelled example $(\mathbf{x}_t^{\mathbf{p}_t}, \mathbf{y}^0)$, which is then used to update the learner.

The discriminant function $g$ measures the compatibility between $(\mathbf{x}, \mathbf{y})$ pairs and gives a high score to those that are well-matched. By restricting this to be a linear function $g(\mathbf{x}, \mathbf{y}) = \langle \mathbf{w}, \mathbf{\Phi}(\mathbf{x}, \mathbf{y}) \rangle$, where $\mathbf{\Phi}(\mathbf{x}, \mathbf{y})$ is a joint kernel map (to be defined later), it can be learned in a large-margin framework from a set of examples $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$ by minimising the convex objective function

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \xi_i \\ \text{s.t.} \quad & \forall i : \; \xi_i \geq 0 \\ & \forall i, \forall \mathbf{y} \neq \mathbf{y}_i : \; \langle \mathbf{w}, \delta\mathbf{\Phi}_i(\mathbf{y}) \rangle \geq \Delta(\mathbf{y}_i, \mathbf{y}) - \xi_i \end{aligned} \tag{5}$$

where $\delta\mathbf{\Phi}_i(\mathbf{y}) = \mathbf{\Phi}(\mathbf{x}_i, \mathbf{y}_i) - \mathbf{\Phi}(\mathbf{x}_i, \mathbf{y})$. This optimisation aims to ensure that the value of $g(\mathbf{x}_i, \mathbf{y}_i)$ for the training example $(\mathbf{x}_i, \mathbf{y}_i)$ is greater than $g(\mathbf{x}_i, \mathbf{y})$ for $\mathbf{y} \neq \mathbf{y}_i$, by a margin which depends on a loss function $\Delta$. This loss function should satisfy $\Delta(\mathbf{y}, \bar{\mathbf{y}}) = 0$ iff $\mathbf{y} = \bar{\mathbf{y}}$ and increase as $\mathbf{y}$ and $\bar{\mathbf{y}}$ become more dissimilar. The loss function plays an important role in our approach, as it allows us to address the issue raised previously of all samples being treated equally. This can be achieved by making use of the transformation similarity function introduced in Section 3.1. For example, as suggested by Blaschko and Lampert [12], we choose to base the loss function on bounding box overlap according to

$$\Delta(\mathbf{y}, \bar{\mathbf{y}}) = 1 - s^o_{\mathbf{p}_t}(\mathbf{y}, \bar{\mathbf{y}}), \tag{6}$$

where $s^o_{\mathbf{p}_t}(\mathbf{y}, \bar{\mathbf{y}})$ is the overlap function (2).

## 3.3 Online optimisation

To optimise (5) in an online setting, we use the approach of Bordes *et al.* [16], [17]. Using standard Lagrangian duality techniques, (5) can be converted into its equivalent dual form

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \sum_{i,\mathbf{y}\neq\mathbf{y}_i} \Delta(\mathbf{y},\mathbf{y}_i)\alpha_i^{\mathbf{y}} - \frac{1}{2}\sum_{\substack{i,\mathbf{y}\neq\mathbf{y}_i\\ j,\bar{\mathbf{y}}\neq\mathbf{y}_j}} \alpha_i^{\mathbf{y}}\alpha_j^{\bar{\mathbf{y}}}\langle\delta\mathbf{\Phi}_i(\mathbf{y}),\delta\mathbf{\Phi}_j(\bar{\mathbf{y}})\rangle \\ \text{s.t.} \quad & \forall i, \forall \mathbf{y}\neq\mathbf{y}_i: \ \alpha_i^{\mathbf{y}} \geq 0 \\ & \forall i: \sum_{\mathbf{y}\neq\mathbf{y}_i}\alpha_i^{\mathbf{y}} \leq C \end{aligned} \tag{7}$$

and the discriminant function expressed as

$$g(\mathbf{x},\mathbf{y}) = \sum_{i,\bar{\mathbf{y}}\neq\mathbf{y}_i} \alpha_i^{\bar{\mathbf{y}}}\langle\delta\mathbf{\Phi}_i(\bar{\mathbf{y}}),\mathbf{\Phi}(\mathbf{x},\mathbf{y})\rangle. \tag{8}$$

As in the case of classification SVMs, a benefit of this dual representation is that because the joint kernel map $\mathbf{\Phi}(\mathbf{x},\mathbf{y})$ only ever occurs inside scalar products, it can be defined implicitly in terms of an appropriate joint kernel function $k(\mathbf{x},\mathbf{y},\bar{\mathbf{x}},\bar{\mathbf{y}}) = \langle\mathbf{\Phi}(\mathbf{x},\mathbf{y}),\mathbf{\Phi}(\bar{\mathbf{x}},\bar{\mathbf{y}})\rangle$. The kernel functions we use during tracking are discussed in Section 3.5. By reparametrising (7) [16] according to

$$\beta_i^{\mathbf{y}} = \begin{cases} -\alpha_i^{\mathbf{y}} & \text{if } \mathbf{y}\neq\mathbf{y}_i \\ \sum_{\bar{\mathbf{y}}\neq\mathbf{y}_i}\alpha_i^{\bar{\mathbf{y}}} & \text{otherwise,} \end{cases} \tag{9}$$

the dual can be considerably simplified to

$$\begin{aligned} \max_{\boldsymbol{\beta}} \quad & -\sum_{i,\mathbf{y}}\Delta(\mathbf{y},\mathbf{y}_i)\beta_i^{\mathbf{y}} - \frac{1}{2}\sum_{i,\mathbf{y},j,\bar{\mathbf{y}}}\beta_i^{\mathbf{y}}\beta_j^{\bar{\mathbf{y}}}\langle\mathbf{\Phi}(\mathbf{x}_i,\mathbf{y}),\mathbf{\Phi}(\mathbf{x}_j,\bar{\mathbf{y}})\rangle \\ \text{s.t.} \quad & \forall i,\forall\mathbf{y}: \ \beta_i^{\mathbf{y}} \leq \delta(\mathbf{y},\mathbf{y}_i)C \\ & \forall i: \sum_{\mathbf{y}}\beta_i^{\mathbf{y}} = 0 \end{aligned} \tag{10}$$

where $\delta(\mathbf{y},\bar{\mathbf{y}}) = 1$ if $\mathbf{y} = \bar{\mathbf{y}}$ and 0 otherwise. This also simplifies the discriminant function to

$$g(\mathbf{x},\mathbf{y}) = \sum_{i,\bar{\mathbf{y}}}\beta_i^{\bar{\mathbf{y}}}\langle\mathbf{\Phi}(\mathbf{x}_i,\bar{\mathbf{y}}),\mathbf{\Phi}(\mathbf{x},\mathbf{y})\rangle. \tag{11}$$

In this form we refer to those pairs $(\mathbf{x}_i,\mathbf{y})$ for which $\beta_i^{\mathbf{y}} \neq 0$ as *support vectors* and those $\mathbf{x}_i$ included in at least one support vector as *support patterns*. Note that for a given support pattern $\mathbf{x}_i$, only the support vector $(\mathbf{x}_i,\mathbf{y}_i)$ will have $\beta_i^{\mathbf{y}_i} > 0$, while any other support vectors $(\mathbf{x}_i,\mathbf{y})$, $\mathbf{y}\neq\mathbf{y}_i$, will have $\beta_i^{\mathbf{y}} < 0$. We refer to these as positive and negative support vectors respectively.

The core step in the optimisation algorithm of Bordes *et al.* [16], [17] is an SMO-style step [41] which monotonically improves (10) with respect to a pair of coefficients $\beta_i^{\mathbf{y}+}$ and $\beta_i^{\mathbf{y}-}$. Because of the constraint $\sum_{\mathbf{y}}\beta_i^{\mathbf{y}} = 0$, the coefficients must be modified by opposite amounts, $\beta_i^{\mathbf{y}+} \leftarrow \beta_i^{\mathbf{y}+} + \lambda$, $\beta_i^{\mathbf{y}-} \leftarrow \beta_i^{\mathbf{y}-} - \lambda$, leading to a one-dimensional maximisation in $\lambda$ which can be solved in closed form (Algorithm 1).

The remainder of the online learning algorithm centres around how to choose the triplet $(i,\mathbf{y}_+,\mathbf{y}_-)$ which should be optimised by this SMO step. For a given $i$, $\mathbf{y}_+$ and $\mathbf{y}_-$ are chosen to define the feasible search direction with the highest gradient, where the gradient of (10) with respect to a single coefficient $\beta_i^{\mathbf{y}}$ is given by

$$\begin{aligned} \nabla_i(\mathbf{y}) &= -\Delta(\mathbf{y},\mathbf{y}_i) - \sum_{j,\bar{\mathbf{y}}}\beta_j^{\bar{\mathbf{y}}}\langle\mathbf{\Phi}(\mathbf{x}_i,\mathbf{y}),\mathbf{\Phi}(\mathbf{x}_j,\bar{\mathbf{y}})\rangle \\ &= -\Delta(\mathbf{y},\mathbf{y}_i) - g(\mathbf{x}_i,\mathbf{y}). \end{aligned} \tag{12}$$

**Algorithm 1** SMOSTEP

```
Require: $i$, $\mathbf{y}_+$, $\mathbf{y}_-$
 1: $k_{00} = \langle\mathbf{\Phi}(\mathbf{x}_i,\mathbf{y}_+),\mathbf{\Phi}(\mathbf{x}_i,\mathbf{y}_+)\rangle$
 2: $k_{11} = \langle\mathbf{\Phi}(\mathbf{x}_i,\mathbf{y}_-),\mathbf{\Phi}(\mathbf{x}_i,\mathbf{y}_-)\rangle$
 3: $k_{01} = \langle\mathbf{\Phi}(\mathbf{x}_i,\mathbf{y}_+),\mathbf{\Phi}(\mathbf{x}_i,\mathbf{y}_-)\rangle$
 4: $\lambda^u = \frac{g_i(\mathbf{y}_+)-g_i(\mathbf{y}_-)}{k_{00}+k_{11}-2k_{01}}$
 5: $\lambda = \max(0, \min(\lambda^u, C\delta(\mathbf{y}_+,\mathbf{y}_i) - \beta_i^{\mathbf{y}+}))$
 6: Update coefficients
 7: $\beta_i^{\mathbf{y}+} \leftarrow \beta_i^{\mathbf{y}+} + \lambda$
 8: $\beta_i^{\mathbf{y}-} \leftarrow \beta_i^{\mathbf{y}-} - \lambda$
 9: Update gradients
10: for $(\mathbf{x}_j,\mathbf{y}) \in \mathcal{S}$ do
11:     $k_0 = \langle\mathbf{\Phi}(\mathbf{x}_j,\mathbf{y}),\mathbf{\Phi}(\mathbf{x}_i,\mathbf{y}_+)\rangle$
12:     $k_1 = \langle\mathbf{\Phi}(\mathbf{x}_j,\mathbf{y}),\mathbf{\Phi}(\mathbf{x}_i,\mathbf{y}_-)\rangle$
13:     $\nabla_j(\mathbf{y}) \leftarrow \nabla_j(\mathbf{y}) - \lambda(k_0 - k_1)$
14: end for
```

Three different update steps are considered, which map very naturally onto a tracking framework:

- PROCESSNEW Processes a new example $(\mathbf{x}_i,\mathbf{y}_i)$. Because all the $\beta_i^{\mathbf{y}}$ are initially 0, and only $\beta_i^{\mathbf{y}_i} \geq 0$, $\mathbf{y}_+ = \mathbf{y}_i$. $\mathbf{y}_-$ is found according to $\mathbf{y}_- = \operatorname{argmin}_{\mathbf{y}\in\mathcal{Y}}\nabla_i(\mathbf{y})$. During tracking, this corresponds to adding the true label $\mathbf{y}_i$ as a positive support vector and searching for the most important sample to become a negative support vector according to the current state of the learner, taking into account the loss function. Note, however, that this step does not necessarily add new support vectors, since the SMO step may not need to adjust the $\beta_i^{\mathbf{y}}$ away from 0.
- PROCESSOLD Processes an existing support pattern $\mathbf{x}_i$ chosen at random. $\mathbf{y}_+ = \operatorname{argmax}_{\mathbf{y}\in\mathcal{Y}}\nabla_i(\mathbf{y})$, but a feasible search direction requires $\beta_i^{\mathbf{y}} < \delta(\mathbf{y},\mathbf{y}_i)C$, meaning this maximisation will only involve existing support vectors. As for PROCESSNEW, $\mathbf{y}_- = \operatorname{argmin}_{\mathbf{y}\in\mathcal{Y}}\nabla_i(\mathbf{y})$. During tracking, this corresponds to revisiting a frame for which we have retained some support vectors and potentially adding another sample as a negative support vector, as well as adjusting the associated coefficients. Again, this new sample is chosen to take into account the current learner state and loss function.
- OPTIMIZE Processes an existing support pattern $\mathbf{x}_i$ chosen at random, but only modifies coefficients of existing support vectors. $\mathbf{y}_+$ is chosen as for PROCESSOLD, and $\mathbf{y}_- = \operatorname{argmin}_{\mathbf{y}\in\mathcal{Y}_i}\nabla_i(\mathbf{y})$, where $\mathcal{Y}_i = \{\mathbf{y}\in\mathcal{Y} \mid \beta_i^{\mathbf{y}} \neq 0\}$.

Of these cases, PROCESSNEW and PROCESSOLD are both able to add new support vectors, which gives the learner the ability to perform sample selection during tracking and discover important background elements. This selection involves searching over $\mathcal{Y}$ to minimise $\nabla_i(\mathbf{y})$, which may be a relatively expensive operation. In practice, we found that for the 2D translation case it was sufficient to sample from $\mathcal{Y}$ on a polar grid, rather than considering every pixel offset. The OPTIMIZE case only considers existing support vectors, so is a much less expensive operation.

As suggested by Bordes *et al.* [17], we schedule these update steps as follows. A REPROCESS step is defined as a single PROCESSOLD step followed by $n_O$ OPTIMIZE steps. Given a new training example $(\mathbf{x}_i, \mathbf{y}_i)$ we call a single PROCESSNEW step followed by $n_R$ REPROCESS steps. In practice we typically use $n_O = n_R = 10$.

During tracking, we maintain a set of support vectors $\mathcal{S}$. For each $(\mathbf{x}_i, \mathbf{y}) \in \mathcal{S}$ we store the coefficient $\beta_i^{\mathbf{y}}$ and gradient $\nabla_i(\mathbf{y})$, which are both incrementally updated during an SMO step. If the SMO step results in a $\beta_i^{\mathbf{y}}$ becoming 0, the corresponding support vector is removed from $\mathcal{S}$.

## 3.4 Incorporating a budget

An issue with the approach described thus far is that the number of support vectors is not bounded and in general will increase over time. Evaluating $g(\mathbf{x}, \mathbf{y})$ requires evaluating scalar products (or kernel functions) between $(\mathbf{x}, \mathbf{y})$ and each support vector, which means that both the computational and storage costs grow linearly with the number of support vectors. Additionally, since (12) involves evaluating $g$, both the PROCESSNEW and PROCESSOLD update steps will become more expensive as the number of support vectors increases. This issue is particularly important in the case of tracking, as in principle we could be presented with an infinite number of training examples.

Recently a number of approaches have been proposed for online learning of classification SVMs on a fixed budget [18], [19], meaning the number of support vectors cannot exceed a specified limit. If the budget is already full and a new support vector needs to be added, these approaches identify a suitable support vector to remove and potentially adjust the coefficients of the remaining support vectors to compensate for the removal.

We now propose an approach for incorporating a budget into the algorithm presented in Section 3.3. Similar to Wang *et al.* [19], we choose to remove the support vector which results in the smallest change to the weight vector $\mathbf{w}$, as measured by $\|\Delta\mathbf{w}\|^2$. However, as with the SMO step used during optimisation, we must also ensure that the constraint $\sum_{\mathbf{y}} \beta_i^{\mathbf{y}} = 0$ remains satisfied. Because of the fact that there only exists one positive support vector for each support pattern, it is sufficient to only consider the removal of negative support vectors during budget maintenance. In the case that a support pattern has only two support vectors, then this will result in them both being removed. Removing the negative support vector $(\mathbf{x}_r, \mathbf{y})$ results in the weight vector changing according to

**Algorithm 2** Struck tracking loop.

```
Require: $\mathbf{f}_t$, $\mathbf{p}_{t-1}$, $\mathcal{S}_{t-1}$
 1: Estimate change in object location
 2: $\mathbf{y}_t = \mathrm{argmax}_{\mathbf{y}\in\mathcal{Y}}\ g(\mathbf{x}_t^{\mathbf{p}_{t-1}}, \mathbf{y})$
 3: $\mathbf{p}_t = \mathbf{p}_{t-1} \circ \mathbf{y}_t$
 4: Update discriminant function
 5: $(i, \mathbf{y}_+, \mathbf{y}_-) \leftarrow$ PROCESSNEW$(\mathbf{x}_t^{\mathbf{p}_t}, \mathbf{y}^0)$
 6: SMOSTEP$(i, \mathbf{y}_+, \mathbf{y}_-)$
 7: BUDGETMAINTENANCE()
 8: for $j = 1$ to $n_R$ do
 9:     $(i, \mathbf{y}_+, \mathbf{y}_-) \leftarrow$ PROCESSOLD()
10:     SMOSTEP$(i, \mathbf{y}_+, \mathbf{y}_-)$
11:     BUDGETMAINTENANCE()
12:     for $k = 1$ to $n_O$ do
13:         $(i, \mathbf{y}_+, \mathbf{y}_-) \leftarrow$ OPTIMIZE()
14:         SMOSTEP$(i, \mathbf{y}_+, \mathbf{y}_-)$
15:     end for
16: end for
17: return $\mathbf{p}_t$, $\mathcal{S}_t$
```

$$\bar{\mathbf{w}} = \mathbf{w} - \beta_r^{\mathbf{y}} \mathbf{\Phi}(\mathbf{x}_r, \mathbf{y}) + \beta_r^{\mathbf{y}} \mathbf{\Phi}(\mathbf{x}_r, \mathbf{y}_r), \tag{13}$$

meaning

$$\begin{aligned} \|\Delta\mathbf{w}\|^2 = &\beta_r^{\mathbf{y}\,2} \{ \langle \mathbf{\Phi}(\mathbf{x}_r, \mathbf{y}), \mathbf{\Phi}(\mathbf{x}_r, \mathbf{y}) \rangle + \\ &\langle \mathbf{\Phi}(\mathbf{x}_r, \mathbf{y}_r), \mathbf{\Phi}(\mathbf{x}_r, \mathbf{y}_r) \rangle - \\ &2 \langle \mathbf{\Phi}(\mathbf{x}_r, \mathbf{y}), \mathbf{\Phi}(\mathbf{x}_r, \mathbf{y}_r) \rangle \}. \end{aligned} \tag{14}$$

Each time the budget is exceeded we remove the support vector resulting in the minimum $\|\Delta\mathbf{w}\|^2$. We show in the experimental section that this does not impact significantly on tracking performance, even with modest budget sizes, and improves the efficiency. We name the proposed algorithm *Struck* and show the overall tracking loop in Algorithm 2. Our unoptimised C++ implementation of Struck is publicly available[1].

## 3.5 Kernel functions and image features

The use of a structured output SVM framework provides great flexibility in how images are actually represented. In practice we choose to use a restriction kernel [12] which uses the relative bounding box location $\mathbf{y}$ to crop a patch from a frame $\mathbf{x}_t^{\mathbf{p}\circ\mathbf{y}}$, allowing a standard image kernel to be applied between pairs of such patches

$$k_{xy}(\mathbf{x}, \mathbf{y}, \bar{\mathbf{x}}, \bar{\mathbf{y}}) = k(\mathbf{x}^{\mathbf{p}\circ\mathbf{y}}, \bar{\mathbf{x}}^{\bar{\mathbf{p}}\circ\bar{\mathbf{y}}}). \tag{15}$$

The use of kernels makes it straightforward to incorporate different image features into our approach, and in our experiments we consider a number of examples. We also investigate using multiple kernels in order to combine different image features together.

1. http://www.samhare.net/research

| Sequence | Struck$_{\infty}$ | Struck$_{100}$ | Struck$_{50}$ | Struck$_{20}$ | MIForest | OMCLP | MIL | Frag | OAB |
|---|---|---|---|---|---|---|---|---|---|
| *coke* | **0.57** | **0.57** | 0.56 | 0.52 | 0.35 | 0.24 | 0.33 | 0.08 | 0.17 |
| *david* | 0.80 | 0.80 | **0.81** | 0.35 | 0.72 | 0.61 | 0.57 | 0.43 | 0.26 |
| *face1* | 0.86 | 0.86 | 0.86 | 0.81 | 0.77 | 0.80 | 0.60 | **0.88** | 0.48 |
| *face2* | **0.86** | **0.86** | **0.86** | 0.83 | 0.77 | 0.78 | 0.68 | 0.44 | 0.68 |
| *girl* | **0.80** | **0.80** | **0.80** | 0.79 | 0.71 | 0.64 | 0.53 | 0.60 | 0.40 |
| *sylvester* | **0.68** | **0.68** | 0.67 | 0.58 | 0.59 | 0.67 | 0.60 | 0.62 | 0.52 |
| *tiger1* | **0.70** | **0.70** | 0.69 | 0.68 | 0.55 | 0.53 | 0.52 | 0.19 | 0.23 |
| *tiger2* | 0.56 | **0.57** | 0.55 | 0.39 | 0.53 | 0.44 | 0.53 | 0.15 | 0.28 |
| Average FPS | 12.1 | 13.2 | 16.2 | 21.4 | | | | | |

TABLE 1: Average bounding box overlap on benchmark sequences. The first four columns correspond to our method with different budget size indicated by the subscript, and the rest of the columns show published results from state-of-the-art approaches. The best performing method is shown in bold. We also show underlined the cases when Struck with the smallest budget size ($B = 20$) outperforms the state-of-the-art. The last row gives the average number of frames per second for an unoptimised C++ implementation of our method.

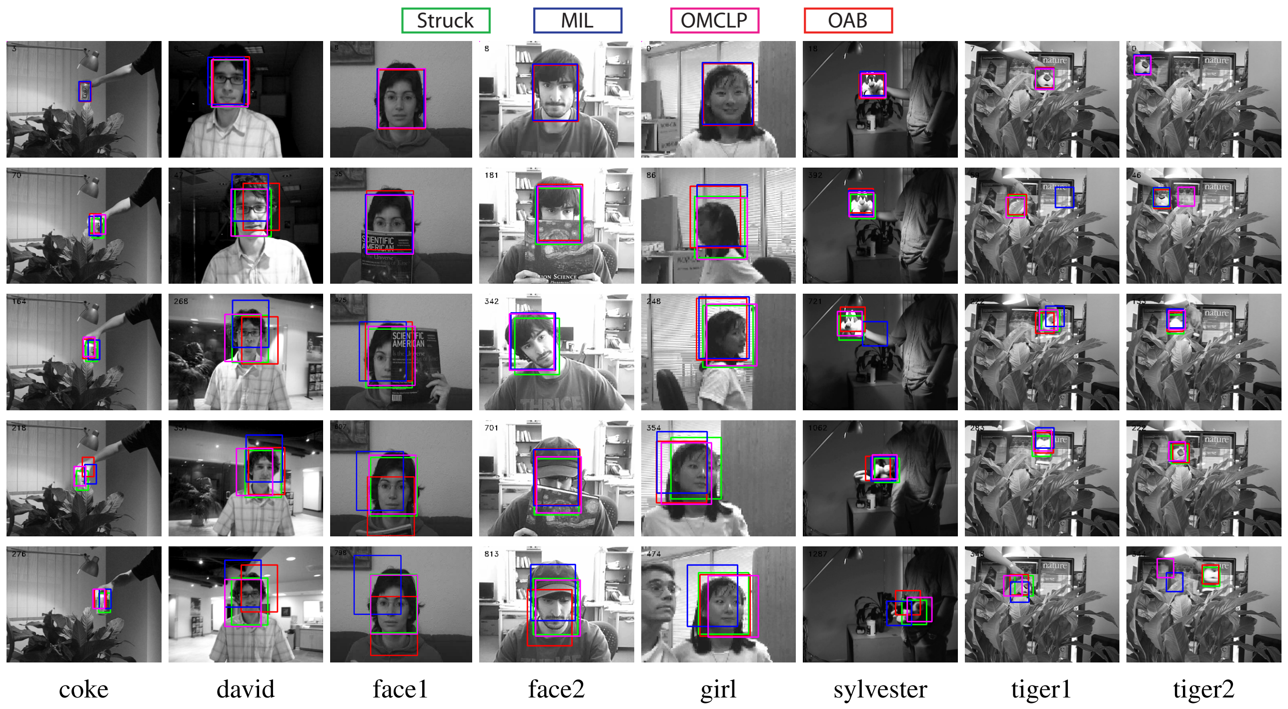


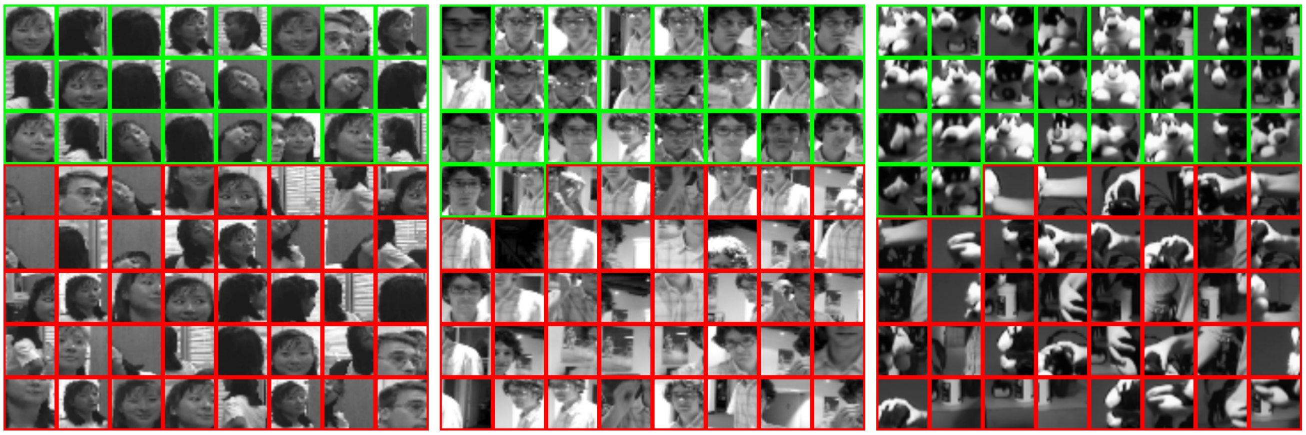
Fig. 2: Example frames from benchmark tracking sequences, showing the results of Struck compared with MILTrack [3], OMCLP [4] and OAB [2]. Videos of these results can be found at http://www.samhare.net/research.

## 4 Experiments

### 4.1 Tracking-by-detection benchmarks

Our first set of experiments aims to compare the results of the proposed approach with existing tracking-by-detection approaches. The majority of these are based around boosting or random forests and use simple Haar-like features as their image representation. We use similar features for our evaluation in order to provide a fair comparison and isolate the effect of the learning framework, but note that these features were specifically designed to work with the feature-selection capability of boosting, having been originally introduced by Viola and Jones [11]. Even so, we find that with our framework we are able to significantly outperform the existing state-of-the-art results.

We use 6 different types of Haar-like feature arranged on a grid at 2 scales on a $4 \times 4$ grid, resulting in 192 features, with each feature normalised to give a value in the range $[-1, 1]$. The reason for using a grid, as opposed to random locations, is partly to limit the number of random factors in the tracking algorithm, since the learner itself has a random element, and partly to compensate for the fact that we do not perform feature selection. Note, however, that the number of features we use is lower than systems against which we compare, which use at least 250. We concatenate the feature responses into a feature vector $\mathbf{x}$ and apply a Gaussian kernel $k(\mathbf{x}, \bar{\mathbf{x}}) = \exp(-\sigma\|\mathbf{x}-\bar{\mathbf{x}}\|^2)$, with $\sigma = 0.2$ and $C = 100$ which is fixed for all sequences. Like the systems against which we compare, we track 2D translation $\mathcal{Y} = \{(\Delta u, \Delta v) \mid \Delta u^2 + \Delta v^2 < r^2\}$. During tracking we use a search radius $r = 30$ pixels, though when updating the classifier we take a larger radius $r = 60$ to ensure stability. As mentioned in Section 3.3, we found empirically that searching $\mathcal{Y}$ exhaustively when performing online learning was unnecessary, and it is sufficient to sample from $\mathcal{Y}$ on a



(a) *girl* (b) *david* (c) *sylvester*

Fig. 3: Visualisation of the support vector set $\mathcal{S}$ at the end of tracking with $B = 64$ (chosen for illustrative purposes). Each patch shows $\mathbf{x}_t^{\mathbf{p}\circ\mathbf{y}}$, and positive and negative support vectors have green and red borders respectively. Notice that the positive support vectors capture the change in appearance of the target object during tracking.

polar grid (we use 5 radial and 16 angular divisions, giving 81 locations).

To assess tracking performance, we use the Pascal VOC overlap criterion as suggested by Saffari *et al.* [4] and report the average overlap between estimated and ground truth throughout each sequence. Because of the randomness involved in our learning algorithm, we repeat each sequence 5 times with different random seeds and report the median result.

Table 1 shows the results obtained by our tracking framework for various budget sizes $B$, along with published results from existing state-of-the-art approaches [2]–[4], [42], [43], and example frames can be seen in Figure 2. It can be seen from these results that Struck outperforms the current state-of-the-art on almost every sequence, often by a considerable margin. These results also demonstrate that the proposed budgeting mechanism does not impact significantly on tracking results. Even when the budget is reduced as low as $B = 20$ we outperform the state-of-the-art on 4 out of 8 sequences.

In Figure 3 we show some examples of the support vector set $\mathcal{S}$ at the end of tracking. An interesting property which can be observed is that the positive support vectors (shown with green borders) provide a compact summary of the change in object appearance observed during tracking. In other words, our tracker is able to identify distinct appearances of the object over time. Additionally, it is clear that the algorithm automatically chooses more negative support vectors than positive. This is mainly because the foreground can be expressed more compactly than the background, which has higher diversity. We also see from these figures that the budgeting mechanism we use maintains support vectors from the entire tracking sequence and does not discard old appearance information. We believe that this contributes to the strong performance of our tracker, as it helps prevent drift during tracking which could occur if old information was discarded.

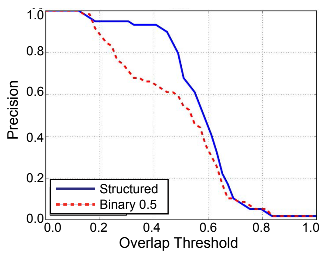
## 4.2 Effect of structured learning

To investigate the importance of structured learning on our results, we next perform a set of experiments against a baseline classification SVM. To achieve this we modify our tracking framework such that the learner is no longer trained using structured examples, but rather using a set of binary examples. Each frame a single positive example is generated using the current tracker state, and negative examples are generated by sampling from $\mathcal{Y}$ as in Section 4.1 and taking those which have an overlap of less than 0.5 with the tracker state (*i.e.* $\theta_u = 1$ and $\theta_l = 0.5$ using the labelling function (3)). All other factors are kept the same, meaning both approaches use the same image features as in Section 4.1 and both use a budget size $B = 100$.

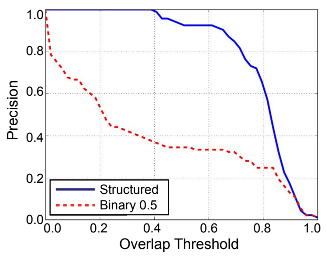
Figure 4 shows precision plots for these two tracking approaches on each of the benchmark test sequences from Section 4.1. These plots show the percentage of frames for which the overlap between the ground truth bounding box and tracker bounding box is greater then a particular threshold, which provides a more detailed view of the tracker performance than the average overlap used in the previous section. As before, we run each tracker 5 times on the sequence and compute the median precision for a given overlap threshold to produce these plots.

We can see from these results that overall the precision curves for the structured SVM are better than or roughly equivalent to those for the classification SVM, which demonstrate that the structured learning framework we use is able to produce gains in accuracy over a traditional classification-based approach. These gains are most notable on the more challenging sequences such as *coke*, *david* and *tiger2*, for which the classification SVM does not perform particularly well.

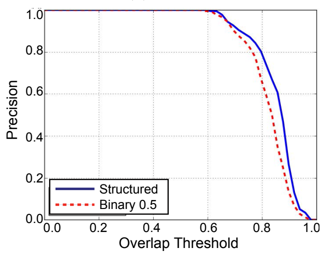
In many cases, however, we see that the performance of the two tracking approaches are quite similar. This indicates that a large part of the performance gains observed in Section 4.1 can be attributed to our use of a kernelised



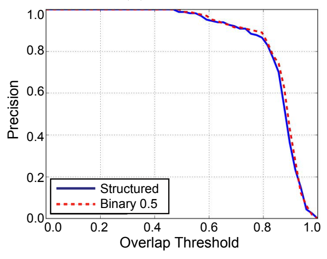
(a) *coke*



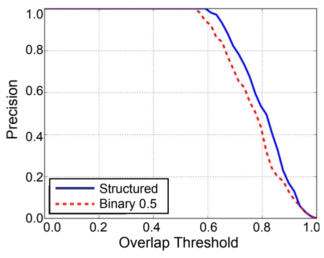
(b) *david*



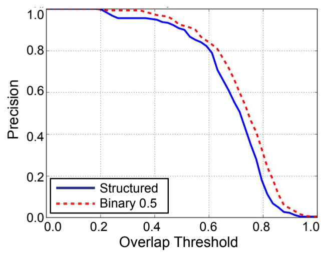
(c) *face1*



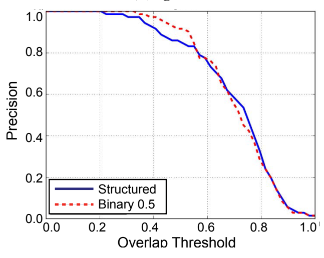
(d) *face2*



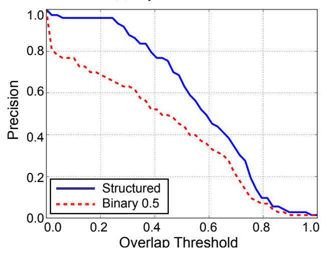
(e) *girl*



(f) *sylvester*



(g) *tiger1*



(h) *tiger2*

Fig. 4: Precision plots comparing the results of tracking using our structured SVM framework with a baseline classification SVM. These plots show the percentage of frames for which the overlap between the ground truth bounding box and tracker bounding box is greater than a particular threshold.

| Sequence | A | B | C | A+B | A+C | B+C | A+B+C |
|---|---|---|---|---|---|---|---|
| *coke* | 0.57 | 0.67 | $\underline{0.69}$ | 0.62 | 0.65 | 0.68 | 0.63 |
| *david* | 0.80 | $\underline{0.83}$ | 0.67 | 0.84 | 0.68 | **0.87** | **0.87** |
| *face1* | $\underline{0.86}$ | 0.82 | $\underline{0.86}$ | 0.82 | **0.87** | 0.82 | 0.83 |
| *face2* | $\underline{0.86}$ | 0.79 | 0.79 | 0.83 | **0.86** | 0.78 | 0.84 |
| *girl* | $\underline{0.80}$ | 0.77 | 0.68 | 0.79 | **0.80** | 0.79 | 0.79 |
| *sylvester* | 0.68 | $\underline{0.75}$ | 0.72 | 0.73 | 0.72 | **0.77** | 0.73 |
| *tiger1* | 0.70 | 0.69 | $\underline{0.77}$ | 0.69 | 0.74 | 0.74 | 0.72 |
| *tiger2* | 0.57 | 0.50 | $\underline{0.61}$ | 0.53 | **0.63** | 0.57 | 0.56 |
| Average | 0.73 | 0.73 | 0.72 | 0.73 | 0.74 | **0.75** | **0.75** |

TABLE 2: Combining kernels. A: Haar features with Gaussian kernel ($\sigma = 0.2$); B: Raw features with Gaussian kernel ($\sigma = 0.1$); C: Histogram features with intersection kernel. The bold shows when multiple kernels improve over the best performance of individual kernels, while the underline shows the best performance within the individual kernels. The last row shows the average of each column.

SVM rather than a boosting-based classifier. Nevertheless, we can still observe that structured learning is able to bring additional performance gains, and importantly it removes the need for introducing a binary labelling strategy, providing a more tightly integrated approach to learning in a tracking context.

### 4.3 Combining kernels

A benefit of the framework we have presented is that it is straightforward to use different image features by modifying the kernel function used for evaluating patch similarity. In addition, different features can be combined by averaging multiple kernels: $k(\mathbf{x}, \bar{\mathbf{x}}) = \frac{1}{N_k} \sum_{i=1}^{N_k} k^{(i)}(\mathbf{x}^{(i)}, \bar{\mathbf{x}}^{(i)})$. Such an approach can be considered a basic form of multiple kernel learning (MKL), and indeed it has been shown [44] that in terms of performance full MKL (in which the relative weighting of the different kernels is learned from training data) does not provide a great deal of improvement over this simple approach.

In addition to the Haar-like features and Gaussian kernel used in Section 4.1, we also consider the following features:

- Raw pixel features obtained by scaling a patch to $16 \times 16$ pixels and taking the greyscale value (in the range $[0, 1]$). This gives a 256-D feature vector, which is combined with a Gaussian kernel with $\sigma = 0.1$.
- Histogram features obtained by concatenating 16-bin intensity histograms from a spatial pyramid of 4 levels. At each level $L$, the patch is divided into $L \times L$ cells, resulting in a 480-D feature vector. This is combined with an intersection kernel: $k(\mathbf{x}, \bar{\mathbf{x}}) = \frac{1}{D} \sum_{i=1}^{D} \min(x_i, \bar{x}_i)$.

Table 2 shows tracking results on the same benchmark videos, with $B = 100$ and all other parameters as specified in Section 4.1. It can be seen that the behaviour of the individual features are somewhat complementary. In many cases, combining multiple kernels seems to improve results. However, it is also noticeable that the performance gains are not significant for some sequences. This could be because of our naïve kernel combination strategy and as has been shown by other researchers, *e.g.* [2], feature selection plays a major role in online tracking. Therefore, further investigation into full MKL could potentially result in further improvements.

## 5 GPU-Based Tracking

To investigate the speed potential of an optimised implementation of Struck, we implemented a CUDA-based version of it called *ThunderStruck*. As with the original Struck, code for ThunderStruck is publicly available[2]. In this section, we describe some of the details of this implementation and compare its speed and performance with the original version.

2. https://bitbucket.org/sgolodetz/thunderstruck

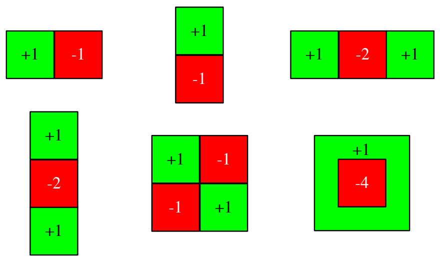


Fig. 5: The different types of Haar feature used by Struck. The numbers in the boxes are the (unnormalised) weights used when calculating the features. Note that no feature requires more than four boxes.

## 5.1 Feature Calculation

Prior to calculating features, we transfer the current frame and any derived images that are best computed on the CPU (e.g. an integral image, for computing Haar features) across to the GPU as CUDA textures. To calculate raw features, we use a variant of the approach in the original Struck that is a better fit for the GPU architecture. Rather than scaling a patch to $16 \times 16$ pixels and then densely sampling the result, we instead sample from a $16 \times 16$ uniform grid placed over the unscaled patch: this allows us to avoid resizing patches on the GPU, whilst producing equivalent results. We compute each raw feature on a separate CUDA thread and calculate the features for all patches in parallel.

To calculate Haar features, we observe that each Haar feature used in the original Struck (see §4.1) can be calculated as the weighted combination of at most four box sums over the pixels of the current frame (see Figure 5). The sum of each box can be calculated efficiently using the integral image for the frame [11]. We can thus compute all of the Haar features without needing to branch on feature type on the GPU by making each CUDA thread calculate a single feature and assigning zero weights to boxes that are unnecessary for features of particular types.

## 5.2 SVM Representation

A Struck-style SVM maintains two separate sets of data: the features computed for patches associated with the support patterns (some of the previously-seen frames), and a record of which patches are in the current set of support vectors, together with their corresponding $\beta$ coefficients and gradient values. Since GPU code is easier to optimise when using fixed-size arrays and we have seen that it is possible to use a finite budget of support vectors (specifically, 100) without degrading tracking performance, we chose to use a fixed-size representation for our SVM data in ThunderStruck (see Figure 6). We use a single large GPU-based array to store all of the features for every patch within every support pattern. We use a smaller array of indices to specify the current support vectors: each element of this array either refers to a patch or is $-1$ to indicate the absence of a support vector. The corresponding $\beta$ coefficients and

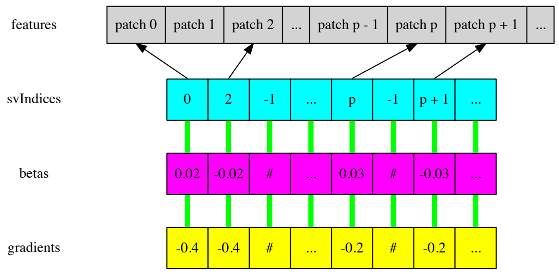


Fig. 6: The representation of the SVM in ThunderStruck.

gradient values are stored in the similarly-sized arrays *betas* and *gradients*, such that the $\beta$ value for support vector $k$ is stored in $betas[k]$ and the gradient value is stored in $gradients[k]$. Since access to the three support vector arrays is required on both the CPU and GPU, we store mirrored copies of them in both places to minimise costly memory transfers over the CPU-GPU bus (note that the storage cost involved is low due to the small size of the arrays).

Addition and removal of support vectors can be implemented via a simple ID allocator that maintains a set of used IDs and a set of free ones. The IDs index into the *svIndices*, *betas* and *gradients* arrays. To add a support vector, we allocate a free ID and use the corresponding elements in the arrays; to remove one, we deallocate the ID, causing it to be returned to the free set, and set the corresponding element of *svIndices* to $-1$.

## 5.3 SVM Evaluation

We implemented both linear and Gaussian kernels for SVM evaluation in ThunderStruck. Linear kernels yield worse tracking performance than Gaussian ones (in particular, they do not cope well with changes in the pose of the tracked object), but offer roughly three times greater speed even on the CPU and are a natural fit to the GPU architecture. We consider the implementation of each type of kernel separately.

*Linear kernels.* A kernelised SVM (as in Struck) that uses a linear kernel can be evaluated efficiently on a set of samples by first calculating the SVM's weight vector and then computing a straightforward dot product of the weight vector with each of the samples. To achieve this, we can rewrite the computation of our discriminant function $g$ on a particular sample as:

$$\begin{aligned} g(\mathbf{x},\mathbf{y}) &= \sum_{i,\bar{\mathbf{y}}} \beta_i^{\bar{\mathbf{y}}} \left\langle \mathbf{\Phi}(\mathbf{x_i},\bar{\mathbf{y}}), \mathbf{\Phi}(\mathbf{x},\mathbf{y}) \right\rangle \\ &= \sum_{i,\bar{\mathbf{y}}} \beta_i^{\bar{\mathbf{y}}} \sum_j \mathbf{\Phi}(\mathbf{x_i},\bar{\mathbf{y}})_j \mathbf{\Phi}(\mathbf{x},\mathbf{y})_j \\ &= \sum_j \mathbf{\Phi}(\mathbf{x},\mathbf{y})_j \left( \underbrace{\sum_{i,\bar{\mathbf{y}}} \beta_i^{\bar{\mathbf{y}}} \mathbf{\Phi}(\mathbf{x_i},\bar{\mathbf{y}})}_{\mathbf{w}} \right)_j \\ &= \langle \mathbf{w}, \mathbf{\Phi}(\mathbf{x},\mathbf{y}) \rangle \end{aligned} \tag{16}$$

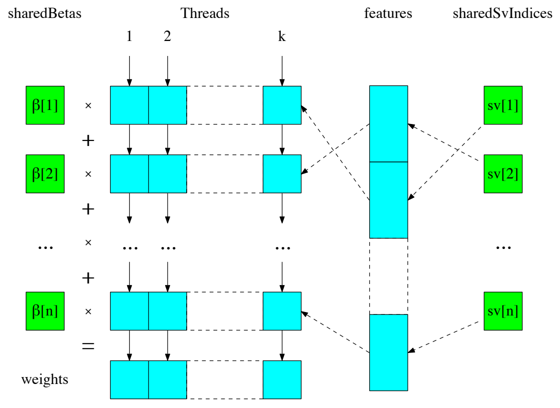


Fig. 7: To compute the weight vector $\mathbf{w}$ for an SVM with a linear kernel efficiently using CUDA, we use a single thread block and compute a single element of $\mathbf{w}$ on each thread. The indices and $\beta$ coefficients of the support vectors are loaded into shared memory at the start of the computation so that they can be accessed quickly by all threads. The computation is structured so that all reads from global memory are *coalesced* (threads access consecutive locations in global memory). The coloured boxes indicate where the data is stored (cyan = global memory, green = shared memory).

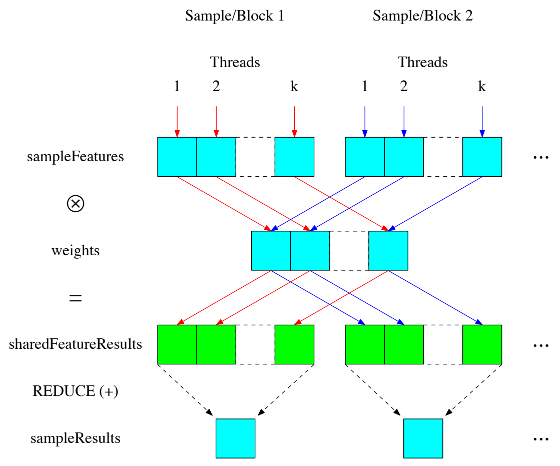


Fig. 8: To evaluate an SVM with a linear kernel efficiently using CUDA, we use a thread block for each sample and compute a dot product between the sample's features and the SVM's weight vector. Each dot product is computed using a pointwise multiplication followed by a reduction in shared memory. The coloured boxes indicate where the data is stored (cyan = global memory, green = shared memory). The coloured arrows distinguish between different thread blocks.

To compute $\mathbf{w}$ efficiently using CUDA, we use a single thread block containing a number of threads equal to the size of our feature vectors (see Figure 7). Each thread then computes one element of $\mathbf{w}$, i.e. thread $j$ computes:

$$\mathbf{w}_j = \sum_{i,\bar{\mathbf{y}}} \beta_i^{\bar{\mathbf{y}}} \mathbf{\Phi}(\mathbf{x_i}, \bar{\mathbf{y}})_j \tag{17}$$

Since all of the threads are in the same thread block, we can load the indices and $\beta$ coefficients of the support vectors into shared memory at the start of the computation: this dramatically reduces the number of costly reads from global memory that each thread would otherwise have to perform (e.g. for 100 support vectors, each thread performs 102 global reads instead of 300). Note also that all the global reads we do perform are *coalesced* (that is, the threads access consecutive locations in global memory): this is important because reading from consecutive locations allows the reads to be grouped into a smaller number of transactions, improving performance.

Having computed $\mathbf{w}$, the SVM can be evaluated on a set of samples by computing the dot product between $\mathbf{w}$ and the features for each sample. Each thread block in our implementation evaluates the SVM for a single sample (see Figure 8). Individual threads multiply corresponding elements of the SVM's weight vector and the sample's feature vector and store the results in shared memory. The result of the dot product is then computed using a reduction.

*Gaussian kernels.* Since computing the SVM's weight vector is reliant on knowing the feature mapping $\mathbf{\Phi}$, we are unable to apply the approach we just used when using a non-linear kernel. Instead, we must evaluate the SVM directly in terms of its support vectors. For a Gaussian kernel, our discriminant function takes the form

$$g(\mathbf{x}, \mathbf{y}) = \sum_{i,\bar{\mathbf{y}}} \beta_i^{\bar{\mathbf{y}}} \exp(-\sigma \left\|\mathbf{x} - \mathbf{x_i}\right\|^2). \tag{18}$$

To parallelise this, we retain the approach of letting each thread block evaluate a single sample, and proceed in multiple passes, each of which calculates the contribution made by a single support vector to the result for the sample. In each pass, thread $j$ computes $(\mathbf{x}[j] - \mathbf{x_i}[j])^2$, i.e. the square of the difference between the $j$'th elements of $\mathbf{x}$ and $\mathbf{x_i}$. These results are summed using a reduction [45] at the end of each pass and the sum is used to calculate the contribution made by that pass's support vector.

## 5.4 Budget Maintenance

As a result of the fixed-size SVM representation we use, the way in which we maintain our support vector budget for ThunderStruck has to differ slightly from that in the original Struck. In particular, instead of removing a support vector when the budget is exceeded, we now remove a support vector at the point at which we need to add a new one but have no available space in the arrays. We found this to be an equivalent scheme that made little difference to the results; however, it would nevertheless be possible to implement a version of the original scheme for fixed-size arrays by adding additional space at the ends of the arrays and maintaining the budget after adding a support vector.

## 5.5 Multi-Threading on the CPU

Whilst it does not represent an improvement to the speed of the tracker itself, it is worth observing that we were able to obtain a further improvement in the speed of the ThunderStruck system as a whole by running the tracker on one CPU thread whilst rendering the output on another. This allows the tracker to process the next frame whilst the current one is still being rendered. This improvement could clearly also have been made to the original version of Struck.

## 5.6 Comparison with Struck

We compare Struck$_{100}$, the best-performing version of the original Struck, with an equivalent version of ThunderStruck (Haar features, a Gaussian kernel with $\sigma = 0.2$ and a budget of 100 support vectors) that we call ThunderStruck$_{HG}$. To perform the experiments, we ran ThunderStruck on the same sequences used in §4.1 and computed the tracking performance (the average bounding box overlap, as before) and speed (in frames per second). The machine used to run ThunderStruck had a hyper-threaded quad-core Intel i7-2600 CPU, running at 3.4 GHz, and an NVIDIA GTX 680 GPU. The results are shown in Table 3, in which we also show the results of an alternative version of ThunderStruck (raw features, a linear kernel and a budget of 100 support vectors) that we call ThunderStruck$_{RL}$. Whilst the tracking performance of ThunderStruck$_{RL}$ is slightly lower than that of ThunderStruck$_{HG}$, it runs significantly faster in practice due to its use of a linear kernel.

As expected, the results illustrate that ThunderStruck$_{HG}$ has similar tracking performance to Struck$_{100}$ (the differences are due to randomness in the method and the different budgeting strategy used by ThunderStruck) but is significantly faster, running at an average of more than 50 frames per second. ThunderStruck$_{RL}$ is faster again, running at an average of 78 frames per second. It is worth noting that even higher average frame-rates (57.8 frames per second for ThunderStruck$_{HG}$ and 112.2 frames per second for ThunderStruck$_{RL}$) can be obtained when rendering is suppressed (e.g. when running the tracker for reasons other than direct output).

| Sequence | Struck$_{100}$ | TS$_{HG}$ | TS$_{RL}$ |
|---|---|---|---|
| *coke* | 0.57 | **0.61** | **0.61** |
| *david* | **0.80** | 0.78 | 0.62 |
| *face1* | 0.86 | **0.88** | 0.84 |
| *face2* | **0.86** | 0.84 | 0.77 |
| *girl* | **0.80** | 0.78 | 0.77 |
| *sylvester* | **0.68** | 0.62 | 0.63 |
| *tiger1* | 0.70 | **0.73** | 0.64 |
| *tiger2* | 0.57 | 0.47 | **0.64** |
| Average Performance | **0.73** | 0.71 | 0.69 |
| Average FPS | 13.2 | 50.9 | **78.0** |

TABLE 3: Comparing ThunderStruck (TS) with the original Struck based on tracking performance and speed. The subscripts indicate the support vector budget for Struck and the feature/kernel combination for ThunderStruck (see main text). The best-performing method is shown in bold. ThunderStruck was run in Ubuntu on a machine with a hyper-threaded quad-core Intel i7-2600 CPU, running at 3.4 GHz, and an NVIDIA GTX 680 GPU.

## 6 Third-Party Benchmarks

Since the original version of Struck was published [20], both Pang and Ling [22] and Wu et al. [23] have published third-party benchmarks that compare it against other state-of-the-art trackers. In this section, we briefly summarise some of their key findings.

### 6.1 Pang and Ling (2013)

Pang and Ling [22] presented a methodology for performing a fairer comparison between the tracking performance of single-target trackers by aggregating results from other comparison papers. Their key idea was to make use of pairwise comparisons that authors performed between third-party algorithms whilst ignoring those that involved the authors' own algorithms, on the basis that the former may be less likely to reflect unconscious subjective bias. They extracted pairwise results for 15 recent trackers (including Struck) from 45 comparison papers in the existing literature. The trackers used included MILTrack [3], MTT [46] and ColorPF [47].

The results were separately aggregated using various aggregation algorithms (rank aggregation, a PageRank-like algorithm, Elo's rating and Glicko's rating) to produce four different full rankings for the trackers. Whilst the rankings differed in the details, they were broadly consistent in their assessment of the best and worst trackers. In particular, the Struck tracker was consistently ranked highest amongst the trackers they studied.

It is important to note that this study explicitly excluded trackers that appeared in fewer than 10 pairwise results or only a single comparison paper: this clearly had the effect of excluding some recent trackers for which there had not yet been sufficient time for a thorough evaluation by the community. In spite of this, it is encouraging that Struck performed consistently well against a significant number of other trackers using a variety of different ranking approaches.

### 6.2 Wu et al. (2013)

Wu et al. [23] performed a large-scale evaluation comparing 29 trackers (including SCM [48], TLD [49] and ASLA [50]) on 50 fully-annotated sequences. Their key ideas were (a) to perturb the initialisation of the trackers in both time and space to improve the robustness of the evaluation, and (b) to evaluate the trackers on sequences that had been annotated to highlight tracking challenges, e.g. fast motion, occlusion, or changes in scale or illumination. Trackers were evaluated using a variety of different tests, in each case using *location error* (the percentage of frames whose predicted bounding boxes were within a fixed pixel threshold of the ground truth bounding boxes) as a precision measure, and *overlap* (the percentage of frames whose predicted bounding boxes overlapped the ground truth bounding boxes by more than a fixed threshold) as a success measure. These measures were calculated for a range of thresholds in each case. The trackers were ranked in terms of precision using a fixed location error threshold of 20 pixels, and in terms of success using the area under the curve (AUC) approach.

There different types of test were performed to compare the trackers' performance: (a) one-pass evaluation (OPE), which initialises the trackers with the ground truth bounding box from the first frame of each sequence, (b) temporal robustness evaluation (TRE), which initialises the trackers at another starting frame in the sequence, and (c) spatial robustness evaluation (SRE), which shifts or scales the ground truth bounding box in the first frame. Overall performance tests of all three types were performed for each tracker on all of the available sequences; further tests were also performed to compare the trackers' performance when restricted to specific types of sequence, e.g. those with a significant amount of fast motion or occlusion.

Struck was the top-performing tracker for all of the overall performance tests except the OPE success test, in which it ranked a narrow second. For the TRE and SRE overall success tests, it performed better than other trackers for small overlap thresholds but less well for large ones, due to not handling scale variation. For the TRE and SRE overall precision tests, it performed significantly better than other trackers at the 20 pixel threshold. For the SRE fast motion tests, dense sampling trackers performed best, with Struck being the top performer among such methods. Similarly, Struck performed strongly on the SRE occlusion and scale variation tests, ranking first for all the tests except the SRE success test for scale variation, in which it ranked a close second to the ASLA tracker. For the SRE tests that initialised trackers at various different scales, Struck also performed well, ranking either first or second in 6 of the 8 tests shown. This is encouraging, since the implementation tested only sampled at a single scale.

## 7 Conclusion

In this paper, we have extended Struck, our adaptive tracking-by-detection framework [20], to the GPU, demonstrating that it is possible for Struck to achieve high framerates without sacrificing tracking performance. Struck is based on structured output prediction, and we have performed additional experiments to quantify the effects of structured learning in comparison with a baseline classification SVM.

Unlike prior methods based on classification, our algorithm does not rely on a heuristic intermediate step for producing labelled binary samples with which to update the classifier, which is often a source of error during tracking. Our approach uses an online structured output SVM learning framework, making it easy to incorporate image features and kernels. From a learning point of view, we take advantage of the well-studied large-margin theory of SVMs, which brings benefits in terms of generalisation and robustness to noise (both in the input and output spaces). To prevent unbounded growth in the number of support vectors, and allow real-time performance, we introduced a budget maintenance mechanism for online structured output SVMs. We have shown experimentally that our algorithm gives superior performance compared to state-of-the-art trackers. We have also discussed recent third-party benchmarks in which Struck achieved state-of-the-art performance.

## Acknowledgements

Financial support was provided by ERC grant ERC-2012-AdG 321162-HELIOS. Stuart Golodetz is funded via a Royal Society Brian Mercer Award for Innovation awarded to Stephen L. Hicks. Philip H. S. Torr is in receipt of a Royal Society Wolfson Research Merit Award and acknowledges support from the Leverhulme Trust and EPSRC.

## References

[1] S. Avidan, "Support vector tracking," *IEEE TPAMI*, vol. 26, no. 8, pp. 1064–72, 2004.

[2] H. Grabner, M. Grabner, and H. Bischof, "Real-Time Tracking via On-line Boosting," in *British Machine Vision Conference*, 2006.

[3] B. Babenko, M.-H. Yang, and S. Belongie, "Robust Object Tracking with Online Multiple Instance Learning," *IEEE TPAMI*, 2011.

[4] A. Saffari, M. Godec, T. Pock, C. Leistner, and H. Bischof, "Online Multi-Class LPBoost," in *CVPR*, 2010.

[5] O. Williams, A. Blake, and R. Cipolla, "A Sparse Probabilistic Learning Algorithm for Real-Time Tracking," in *ICCV*, 2003.

[6] C. Leistner, A. Saffari, P. M. Roth, and H. Bischof, "On Robustness of On-line Boosting - A Competitive Study," in *ICCVW*, 2009.

[7] H. Masnadi-Shirazi, V. Mahadevan, and N. Vasconcelos, "On the design of robust classifiers for computer vision," in *CVPR*, 2010.

[8] H. Grabner, C. Leistner, and H. Bischof, "Semi-Supervised On-Line Boosting for Robust Tracking," in *ECCV*, 2008.

[9] A. Saffari, C. Leistner, M. Godec, and H. Bischof, "Robust multi-view boosting with priors," in *ECCV*, 2010.

[10] B. Zeisl, C. Leistner, A. Saffari, and H. Bischof, "Online Semi-Supervised Multiple-Instance Boosting," in *CVPR*, 2010.

[11] P. Viola and M. J. Jones, "Robust Real-Time Face Detection," *IJCV*, vol. 57, no. 2, pp. 137–154, 2004.

[12] M. B. Blaschko and C. H. Lampert, "Learning to Localize Objects with Structured Output Regression," in *ECCV*, 2008.

[13] A. Vedaldi, V. Gulshan, M. Varma, and A. Zisserman, "Multiple kernels for object detection," in *ICCV*, 2009.

[14] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object Detection with Discriminatively Trained Part-Based Models," *IEEE TPAMI*, vol. 32, no. 9, pp. 1627–1645, 2010.

[15] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, "Large Margin Methods for Structured and Interdependent Output Variables," *JMLR*, vol. 6, pp. 1453–1484, 2005.

[16] A. Bordes, L. Bottou, P. Gallinari, and J. Weston, "Solving multiclass support vector machines with LaRank," in *ICML*, 2007.

[17] A. Bordes, N. Usunier, and L. Bottou, "Sequence Labelling SVMs Trained in One Pass," in *Proc. ECML-PKDD*, 2008.

[18] K. Crammer, J. Kandola, R. Holloway, and Y. Singer, "Online Classification on a Budget," in *NIPS*, 2003.

[19] Z. Wang, K. Crammer, and S. Vucetic, "Multi-Class Pegasos on a Budget," in *ICML*, 2010.

[20] S. Hare, A. Saffari, and P. H. S. Torr, "Struck: Structured Output Tracking with Kernels," in *ICCV*, 2011.

[21] A. Yilmaz, O. Javed, and M. Shah, "Object Tracking: A Survey," *ACM Computing Surveys*, vol. 38, no. 4, December 2006.

[22] Y. Pang and H. Ling, "Finding the Best from the Second Bests – Inhibiting Subjective Bias in Evaluation of Visual Tracking Algorithms," in *CVPR*, 2013.

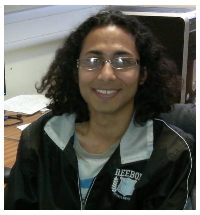
[23] Y. Wu, J. Lim, and M.-H. Yang, "Online Object Tracking: A Benchmark," in *CVPR*, 2013, pp. 2411–2418.

[24] X. Mei, H. Ling, Y. Wu, E. Blasch, and L. Bai, "Minimum Error Bounded Efficient $\ell_1$ Tracker with Occlusion Detection," in *CVPR*, 2011, pp. 1257–1264.
[25] N. Wang, J. Wang, and D.-Y. Yeung, "Online Robust Non-negative Dictionary Learning for Visual Tracking," in *ICCV*, 2013.
[26] J. Xing, J. Gao, B. Li, W. Hu, and S. Yan, "Robust Object Tracking with Online Multi-lifespan Dictionary Learning," in *ICCV*, 2013.
[27] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking," *IEEE TSP*, vol. 50, no. 2, pp. 174–188, February 2002.
[28] S. Cao and W. Xue, "Robust Visual Tracking via Adaptive Forest," in *ICICIP*, June 2013, pp. 30–34.
[29] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof, "Online Random Forests," in *Proc. ICCV-OLCV*, 2009.
[30] J. Santner, C. Leistner, A. Saffari, T. Pock, and H. Bischof, "PROST: Parallel Robust Online Simple Tracking," in *CVPR*, 2010, pp. 723–730.
[31] Q. Bai, Z. Wu, S. Sclaroff, M. Betke, and C. Monnier, "Randomized Ensemble Tracking," in *ICCV*, 2013, pp. 2040–2047.
[32] N. Wang and D.-Y. Yeung, "Ensemble-Based Tracking: Aggregating Crowdsourced Structured Time Series Data," in *ICML*, 2014.
[33] S. Duffner and C. Garcia, "PixelTrack: a fast adaptive algorithm for tracking non-rigid objects," in *ICCV*, 2013, pp. 2480–2487.
[34] X. Ren and J. Malik, "Tracking as Repeated Figure/Ground Segmentation," in *CVPR*, 2007.
[35] S. Wang, H. Lu, F. Yang, and M.-H. Yang, "Superpixel Tracking," in *ICCV*, 2011, pp. 1323–1330.
[36] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "Exploiting the Circulant Structure of Tracking-by-detection with Kernels," in *ECCV*. Springer Berlin Heidelberg, 2012, pp. 702–715.
[37] M. Danelljan, F. S. Khan, M. Felsberg, and J. van de Weijer, "Adaptive Color Attributes for Real-Time Visual Tracking," in *CVPR*, 2014.
[38] F. Pernici and A. D. Bimbo, "Object Tracking by Oversampling Local Features," *IEEE TPAMI*, vol. PP, no. 99, 2013.
[39] Y. Lu, T. Wu, and S.-C. Zhu, "Online Object Tracking, Learning and Parsing with And-Or Graphs," in *CVPR*, 2014.
[40] L. Zhang and L. van der Maaten, "Preserving Structure in Model-Free Tracking," *IEEE TPAMI*, vol. 36, no. 4, pp. 756–769, 2014.
[41] J. C. Platt, *Fast Training of Support Vector Machines Using Sequential Minimal Optimization*. MIT Press, 1999, pp. 185–208.
[42] A. Adam, E. Rivlin, and I. Shimshoni, "Robust Fragments-Based Tracking using the Integral Histogram," in *CVPR*, 2006.
[43] C. Leistner, A. Saffari, and H. Bischof, "MIForests: Multiple-Instance Learning with Randomized Trees," in *ECCV*, 2010.
[44] P. Gehler and S. Nowozin, "On Feature Combination for Multiclass Object Classification," in *ICCV*, 2009.
[45] M. Harris, "Optimizing Parallel Reduction in CUDA," *NVIDIA Developer Technology*, vol. 2, p. 45, 2007.
[46] T. Zhang, B. Ghanem, S. Liu, and N. Ahuja, "Robust Visual Tracking via Structured Multi-Task Sparse Learning," *IJCV*, vol. 101, 2013.
[47] P. Pérez, C. Hue, J. Vermaak, and M. Gangnet, "Color-Based Probabilistic Tracking," in *ECCV*, 2002, pp. 661–675.
[48] W. Zhong, H. Lu, and M.-H. Yang, "Robust Object Tracking via Sparsity-based Collaborative Model," in *CVPR*, 2012.
[49] Z. Kalal, J. Matas, and K. Mikolajczyk, "P-N Learning: Bootstrapping Binary Classifiers by Structural Constraints," in *CVPR*, 2010.
[50] X. Jia, H. Lu, and M.-H. Yang, "Visual Tracking via Adaptive Structural Local Sparse Appearance Model," in *CVPR*, 2012.

**Sam Hare** received his PhD from Oxford Brookes University in 2012 as a member of the Brookes Vision Group. His PhD research focused on online structured learning for real-time computer vision applications, and was carried out in collaboration with Sony Computer Entertainment Europe. He is now co-founder and CTO of Obvious Engineering, developing 3D scene reconstruction technology for mobile devices.

**Amir Saffari** obtained his PhD on Online and Semi-supervised Learning from Graz University of Technology in 2010, and in the same year joined Sony Computer Entertainment Europe's London Studio to work on computer vision-based technology for video games – notably Wonderbook, an interactive augmented reality book for PS3. In 2013, he moved to Affectv as the Director of Research and leads the data science team working on big data machine learning algorithms.

**Stuart Golodetz** obtained his PhD in Computer Science at the University of Oxford in 2011, working on 3D image segmentation and feature identification. After working in industry for both SunGard and Semmle, he is currently a research associate at the University of Oxford. His areas of interest include image processing and computer games. He was a session chair for ISPA 2009.



**Vibhav Vineet** is a research fellow at the University of Oxford and closely collaborates with the I3D group at the Microsoft Research Cambridge. His research interests are in computer vision and machine learning.

**Ming-Ming Cheng** received his PhD degree from Tsinghua University in 2012. He is currently a research fellow at the University of Oxford. His research interests include computer graphics, computer vision, and image processing. He has received the Google PhD fellowship award and the IBM PhD fellowship award. He reviews papers regularly for IEEE TPAMI, ACM SIGGRAPH, etc.

**Stephen L. Hicks** received his PhD from the University of Sydney and is a Research Fellow in neuroscience and visual prosthetics at the University of Oxford. He leads a program of research to develop and validate non-invasive sight enhancement techniques based on computer vision for legally blind individuals. He won the 2013 Royal Society Brian Mercer Award for Innovation and the 2014 SET for Britain prize for Engineering. Stephen is funded by the NIHR i4i program.

**Philip H. S. Torr** received the PhD degree from Oxford University. After working for another three years at Oxford, he worked for six years as a research scientist for Microsoft Research, first in Redmond, then in Cambridge, founding the vision side of the Machine Learning and Perception Group. He is now a professor at Oxford University. He has won awards from several top vision conferences, including ICCV, CVPR, ECCV, NIPS and BMVC. He is a senior member of the IEEE, Royal Society Wolfson Research Merit Award holder, and program co-chair of ICCV 2013.