Struck: Structured Output Tracking with Kernels

Sam Hare[1] Amir Saffari[1,2] Philip H. S. Torr[1]
[1]Oxford Brookes University, Oxford, UK [2]Sony Computer Entertainment Europe, London, UK
{sam.hare,philiptorr}@brookes.ac.uk amir@ymer.org

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 performance.

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 to allow successful tracking.

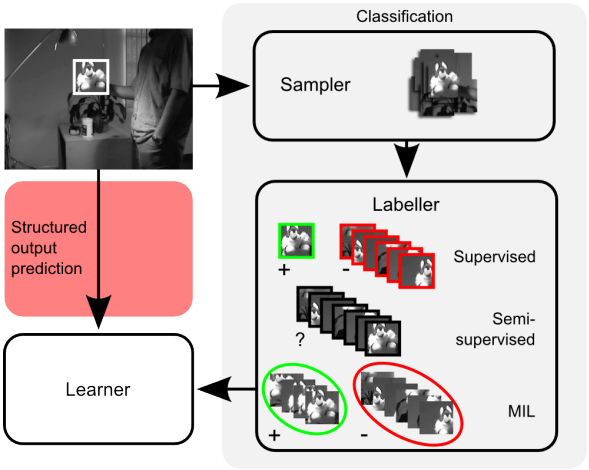

Figure 1. Different adaptive tracking-by-detection paradigms: given the current estimated object location, traditional approaches (shown on the right-hand side) generate a set of samples and, depending on the type of learner, produce training labels. Our approach (left-hand side) avoids these steps,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

An approach to tracking which has become particularly popular recently is tracking-by-detection [2], which treats the tracking problem as a detection task applied over time. 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 [2]. 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,

2011 IEEE International Conference on Computer Vision
978-1-4577-1102-2/11/$26.00 ©2011 IEEE

e.g. [3, 10, 16].

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.* in [22]). 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 [13, 14], semi-supervised learning [11, 17], or multiple-instance learning [3, 23].

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 [3, 10, 16]. In object detection, boosting has proved to be very successful for particular tasks, most notably face detection using the approach of [20]. 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 support vector machines (SVMs) rather than boosting, due to their good generalization ability, robustness to label noise, and flexibility in object representation through the use of kernels [4, 8, 19]. Because of this flexibility of SVMs and their natural generalization to structured output spaces, we make use of the structured output SVM framework of [18]. In particular, we extend the online structured output SVM learning method proposed in [5, 6] and adapt it to the tracking problem. We find experimentally that the use of our framework results in large performance gains over state-of-the-art tracking by detection approaches.

In [4], Blaschko and Lampert apply a structured output SVM to the task of object detection. 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 kernelization*, 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, meaning that the number of support vectors is constrained to remain within a specified limit, *e.g.* [7, 21]. We apply similar ideas in this work, 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.

2. Online structured output tracking

2.1. Tracking by detection

In the following 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 $\mathbf{f}_t \in \mathcal{F}$, 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 the 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 $\mathcal{Y} = \{(u, v) \mid u^2 + v^2 < r^2\}$, where r is a search radius. Mathematically, an estimate

is found for the change in position relative to the previous frame according to

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

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 samples $\{\mathbf{x}_t^{\mathbf{p}_t \circ \mathbf{y}_t^1}, \ldots, \mathbf{x}_t^{\mathbf{p}_t \circ \mathbf{y}_t^n}\}$. 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 updates 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 labels 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 overcome this problem by using robust loss functions [13, 14], semi-supervised learning [11, 17], or multiple-instance learning [3, 23]. 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 binarization 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 $\mathbf{y}_t^i, \mathbf{y}_t^j$, 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

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

where θ_u and θ_l are upper and lower thresholds, respectively. A binary classifier generally ignores the unlabelled examples [10], while those based on semi-supervised learning use them in their update phase [11, 17]. In approaches based on multiple-instance learning [3, 23], 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 θ_u and θ_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 generalization of these heuristic methods.

2.2. 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 ± 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 [4, 18], which introduces a discriminant function $F : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ that can be used for prediction according to

$$\mathbf{y}_t = f(\mathbf{x}_t^{\mathbf{p}_{t-1}}) = \arg\max_{\mathbf{y} \in \mathcal{Y}} F(\mathbf{x}_t^{\mathbf{p}_{t-1}}, \mathbf{y}). \tag{4}$$

Note the similarity between (4) and (1): we are performing a maximisation step in order to predict the object transformation, however now the discriminant function F includes

the label $\mathbf{y}$ explicitly, meaning it can be incorporated into the learning algorithm. To update the prediction function online, we supply a labelled example relative to the new tracker location $(\mathbf{x}_t^{\mathbf{p}_t}, \mathbf{y}^0)$.

F measures the compatibility between $(\mathbf{x}, \mathbf{y})$ pairs, and gives a high score to those which are well matched. By restricting this to be of the form $F(\mathbf{x}, \mathbf{y}) = \langle \mathbf{w}, \mathbf{\Phi}(\mathbf{x}, \mathbf{y}) \rangle$, 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 example pairs $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$ 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 optimization aims to ensure that the value of $F(\mathbf{x}_i, \mathbf{y}_i)$ is greater than $F(\mathbf{x}_i, \mathbf{y})$ for $\mathbf{y} \neq \mathbf{y}_i$, by a margin which depends on a loss function Δ. This loss function should satisfy $\Delta(\mathbf{y}, \bar{\mathbf{y}}) = 0$ iff $\mathbf{y} = \bar{\mathbf{y}}$ and decrease towards 0 as $\mathbf{y}$ and $\bar{\mathbf{y}}$ become more similar. 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. Note that in our definition, this loss function can also be expressed in terms of the transformation similarity function introduced in the previous section. For example, as in [4] we choose to base this loss function on bounding box overlap, and use

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

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

2.3. Online optimization

To optimize (5) in an online setting, we use the approach of [5, 6]. 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 can be expressed as $F(\mathbf{x}, \mathbf{y}) = \sum_{i, \bar{\mathbf{y}} \neq \mathbf{y}_i} \alpha_i^{\bar{\mathbf{y}}} \langle \delta\mathbf{\Phi}_i(\bar{\mathbf{y}}), \mathbf{\Phi}(\mathbf{x}, \mathbf{y}) \rangle$. As in the case of classification SVMs, a benefit of the dual representation is that because the joint kernel map $\mathbf{\Phi}(\mathbf{x}, \mathbf{y})$ only ever occurs inside inner products, it can be defined implicitly in terms of an appropriate joint kernel function

Algorithm 1 SMOSTEP

Require: $i, \mathbf{y}_+, \mathbf{y}_-$
$k_{00} = \langle \mathbf{\Phi}(\mathbf{x}_i, \mathbf{y}_+), \mathbf{\Phi}(\mathbf{x}_i, \mathbf{y}_+) \rangle$
$k_{11} = \langle \mathbf{\Phi}(\mathbf{x}_i, \mathbf{y}_-), \mathbf{\Phi}(\mathbf{x}_i, \mathbf{y}_-) \rangle$
$k_{01} = \langle \mathbf{\Phi}(\mathbf{x}_i, \mathbf{y}_+), \mathbf{\Phi}(\mathbf{x}_i, \mathbf{y}_-) \rangle$
$\lambda^u = \frac{g_i(\mathbf{y}_+) - g_i(\mathbf{y}_-)}{k_{00} + k_{11} - 2k_{01}}$
$\lambda = \max(0, \min(\lambda^u, C\delta(\mathbf{y}_+, \mathbf{y}_i) - \beta_i^{\mathbf{y}_+}))$
Update coefficients
$\beta_i^{\mathbf{y}_+} \leftarrow \beta_i^{\mathbf{y}_+} + \lambda$
$\beta_i^{\mathbf{y}_-} \leftarrow \beta_i^{\mathbf{y}_-} - \lambda$
Update gradients
for $(\mathbf{x}_j, \mathbf{y}) \in \mathcal{S}$ **do**
  $k_0 = \langle \mathbf{\Phi}(\mathbf{x}_j, \mathbf{y}), \mathbf{\Phi}(\mathbf{x}_i, \mathbf{y}_+) \rangle$
  $k_1 = \langle \mathbf{\Phi}(\mathbf{x}_j, \mathbf{y}), \mathbf{\Phi}(\mathbf{x}_i, \mathbf{y}_-) \rangle$
  $g_j(\mathbf{y}) \leftarrow g_j(\mathbf{y}) - \lambda(k_0 - k_1)$
end for

$k(\mathbf{x}, \mathbf{y}, \bar{\mathbf{x}}, \bar{\mathbf{y}}) = \langle \mathbf{\Phi}(\mathbf{x}, \mathbf{y}), \mathbf{\Phi}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \rangle$. The kernel functions we use during tracking are discussed in Section 2.5.

As in [5], by reparametrising (7) according to

$$\beta_i^{\mathbf{y}} = \begin{cases} -\alpha_i^{\mathbf{y}} & \text{if } \mathbf{y} \neq \mathbf{y}_i \\ \sum_{\bar{\mathbf{y}} \neq \mathbf{y}_i} \alpha_i^{\bar{\mathbf{y}}} & \text{otherwise,} \end{cases} \tag{8}$$

the dual can be considerably simplified to

$$\begin{aligned} \max_{\boldsymbol{\beta}} \quad & -\sum_{i, \mathbf{y}} \Delta(\mathbf{y}, \mathbf{y}_i)\beta_i^{\mathbf{y}} - \frac{1}{2} \sum_{i, \mathbf{y}, j, \bar{\mathbf{y}}} \beta_i^{\mathbf{y}} \beta_j^{\bar{\mathbf{y}}} \langle \mathbf{\Phi}(\mathbf{x}_i, \mathbf{y}), \mathbf{\Phi}(\mathbf{x}_j, \bar{\mathbf{y}}) \rangle \\ \text{s.t.} \quad & \forall i, \forall \mathbf{y}: \ \beta_i^{\mathbf{y}} \leq \delta(\mathbf{y}, \mathbf{y}_i)C \\ & \forall i: \ \sum_{\mathbf{y}} \beta_i^{\mathbf{y}} = 0 \end{aligned} \tag{9}$$

where $\delta(\mathbf{y}, \bar{\mathbf{y}}) = 1$ if $\mathbf{y} = \bar{\mathbf{y}}$ and 0 otherwise. Note that this also simplifies the discriminant function to $F(\mathbf{x}, \mathbf{y}) = \sum_{i, \bar{\mathbf{y}}} \beta_i^{\bar{\mathbf{y}}} \langle \mathbf{\Phi}(\mathbf{x}_i, \bar{\mathbf{y}}), \mathbf{\Phi}(\mathbf{x}, \mathbf{y}) \rangle$. 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 [5, 6] is an SMO-style step [15] which monotonically improves (9) 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 λ and 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 (9) with respect to a single coefficient $\beta_i^{\mathbf{y}}$ is given by

$$\begin{aligned} g_i(\mathbf{y}) &= -\Delta(\mathbf{y}, \mathbf{y}_i) - \sum_{j,\bar{\mathbf{y}}} \beta_j^{\bar{\mathbf{y}}} \langle \mathbf{\Phi}(\mathbf{x}_i, \mathbf{y}), \mathbf{\Phi}(\mathbf{x}_j, \bar{\mathbf{y}}) \rangle \\ &= -\Delta(\mathbf{y}, \mathbf{y}_i) - F(\mathbf{x}_i, \mathbf{y}). \end{aligned} \tag{10}$$

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 $\mathbf{y}_- = \arg\min_{\mathbf{y} \in \mathcal{Y}} g_i(\mathbf{y})$. 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 adjust the $\beta_i^{\mathbf{y}}$ away from 0.

- PROCESSOLD Processes an existing support pattern $\mathbf{x}_i$ chosen at random. $\mathbf{y}_+ = \arg\max_{\mathbf{y} \in \mathcal{Y}} g_i(\mathbf{y})$, but a feasible search direction requires $\beta_i^{\mathbf{y}} < \delta(\mathbf{y}, \mathbf{y}_i)C$, meaning this maximization will only involve existing support vectors. As for PROCESSNEW, $\mathbf{y}_- = \arg\min_{\mathbf{y} \in \mathcal{Y}} g_i(\mathbf{y})$. 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 $\mathbf{y}_- = \arg\min_{\mathbf{y} \in \mathcal{Y}_i} g_i(\mathbf{y})$, where $\mathcal{Y}_i = \{\mathbf{y} \in \mathcal{Y} \mid \beta_i^{\mathbf{y}} \neq 0\}$.

Of these cases, PROCESSNEW and PROCESSOLD both have the ability 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 $g_i(\mathbf{y})$, which may be a relatively expensive operation. In practice, we found 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 in [6], 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 coefficients $\beta_i^{\mathbf{y}}$ and gradients $g_i(\mathbf{y})$, 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 $\mathcal{S}$.

2.4. 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 $F(\mathbf{x}, \mathbf{y})$ requires evaluating inner 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 (10) involves evaluating F, 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 [7, 21], 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 the previous section. Similar to [21], 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 optimization, 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

$$\bar{\mathbf{w}} = \mathbf{w} - \beta_r^{\mathbf{y}} \mathbf{\Phi}(\mathbf{x}_r, \mathbf{y}) + \beta_r^{\mathbf{y}} \mathbf{\Phi}(\mathbf{x}_r, \mathbf{y}_r), \tag{11}$$

meaning

Algorithm 2 Struck: Structured Output Tracking

Require: $\mathbf{f}_t, \mathbf{p}_{t-1}, \mathcal{S}_{t-1}$

Estimate change in object location

$\mathbf{y}_t = \arg\max_{\mathbf{y}\in\mathcal{Y}} F(\mathbf{x}_t^{\mathbf{p}_{t-1}}, \mathbf{y})$

$\mathbf{p}_t = \mathbf{p}_{t-1} \circ \mathbf{y}_t$

Update discriminant function

$(i, \mathbf{y}_+, \mathbf{y}_-) \leftarrow$ PROCESSNEW$(\mathbf{x}_t^{\mathbf{p}_t}, \mathbf{y}^0)$

SMOSTEP$(i, \mathbf{y}_+, \mathbf{y}_-)$

BUDGETMAINTENANCE()

for $j = 1$ to n_R **do**

  $(i, \mathbf{y}_+, \mathbf{y}_-) \leftarrow$ PROCESSOLD()

  SMOSTEP$(i, \mathbf{y}_+, \mathbf{y}_-)$

  BUDGETMAINTENANCE()

  for $k = 1$ to n_O **do**

    $(i, \mathbf{y}_+, \mathbf{y}_-) \leftarrow$ OPTIMIZE()

    SMOSTEP$(i, \mathbf{y}_+, \mathbf{y}_-)$

  end for

end for

return $\mathbf{p}_t, \mathcal{S}_t$

$$\|\Delta\mathbf{w}\|^2 = \beta_r^{\mathbf{y}^2}\big\{\langle\mathbf{\Phi}(\mathbf{x}_r,\mathbf{y}),\mathbf{\Phi}(\mathbf{x}_r,\mathbf{y})\rangle + \langle\mathbf{\Phi}(\mathbf{x}_r,\mathbf{y}_r),\mathbf{\Phi}(\mathbf{x}_r,\mathbf{y}_r)\rangle - 2\langle\mathbf{\Phi}(\mathbf{x}_r,\mathbf{y}),\mathbf{\Phi}(\mathbf{x}_r,\mathbf{y}_r)\rangle\big\}. \quad (12)$$

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 unoptimized C++ implementation of Struck is publicly available[1].

2.5. Kernel functions and image features

The use of a structured output SVM framework provides great flexibility in how images are actually represented. As in [4], we propose using a restriction kernel, which uses the relative bounding box location $\mathbf{y}$ to crop a patch from a frame $\mathbf{x}_t^{\mathbf{p}\circ\mathbf{y}}$, and then applies a standard image kernel between pairs of such patches,

$$k_{xy}(\mathbf{x},\mathbf{y},\bar{\mathbf{x}},\bar{\mathbf{y}}) = k(\mathbf{x}^{\mathbf{p}\circ\mathbf{y}}, \bar{\mathbf{x}}^{\bar{\mathbf{p}}\circ\bar{\mathbf{y}}}). \quad (13)$$

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

3. Experiments

3.1. 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 in [20]. 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 4×4 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 $\mathcal{Y} = \{(u, v) \,|\, u^2 + v^2 < r^2\}$. 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 2.3, we found empirically that searching $\mathcal{Y}$ exhaustively during online learning was unnecessary, and it is sufficient to sample from $\mathcal{Y}$ on a 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 in [16], 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, Struck, for various budget sizes B, along with published results from existing state-of-the-art approaches [1, 3, 10, 12, 16]. It can be seen from these results that the proposed system 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 2 we show some examples of the support vec-

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
Face 1	0.86	0.86	0.86	0.81	0.77	0.80	0.60	**0.88**	0.48
Face 2	**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
Tiger 1	**0.70**	**0.70**	0.69	0.68	0.55	0.53	0.52	0.19	0.23
Tiger 2	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 the 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 unoptimized C++ implementation of our method.

tor set $\mathcal{S}$ at the end of tracking with $B = 64$[2]. 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[3].

(a) *girl* (b) *david*

Figure 2. Visualisation of the support vector set $\mathcal{S}$ at the end of tracking with $B = 64$. 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.

3.2. 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 [9] 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 3.1, we also consider the following features:

- Raw pixel features obtained by scaling a patch to 16×16 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 3.1. 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.* [10], feature selection plays a major role in online tracking. Therefore, further investigation into full MKL could potentially result in further improvements.

4. Conclusions

In this paper, we have presented a new adaptive tracking-by-detection framework based on structured output prediction. Unlike existing methods based on classification, our

[2]Chosen to organize the support vectors in a square image.

[3]Please refer to supplementary material for illustrative videos.

Seq.	A	B	C	AB	AC	BC	ABC
Coke	0.57	0.67	0.69	0.62	0.65	0.68	0.63
Dav.	0.80	0.83	0.67	0.84	0.68	**0.87**	**0.87**
Face1	0.86	0.82	0.86	0.82	**0.87**	0.82	0.83
Face2	0.86	0.79	0.79	0.83	**0.86**	0.78	0.84
Girl	0.80	0.77	0.68	0.79	**0.80**	0.79	0.79
Sylv.	0.68	0.75	0.72	0.73	0.72	**0.77**	0.73
Tig.1	0.70	0.69	0.77	0.69	0.74	0.74	0.72
Tig.2	0.57	0.50	0.61	0.53	**0.63**	0.57	0.56
Av.	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 rows are in the same order as in Table 1. The bold shows when multiple kernels improve over the best of individual kernels, while the underline shows the best performance within the individual kernels. The last row shows the average of each column.

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 generalization 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 also introduced a budget maintenance mechanism for online structured output SVMs. We showed experimentally that our algorithm gives superior performance compared to state-of-the-art trackers.

We believe that the structured output framework we presented provides a very rich platform for incorporating advanced concepts into tracking. For example, it is relatively easy to extend the output space to include rotation and scale transformations. We also plan to incorporate object dynamics into our model. While these extensions focus on the output space, we can also enrich our input space through the use of alternative image features and multiple kernel learning.

Acknowledgements This work is supported by EPSRC CASE and KTP research grants and the IST Programme of the European Community, under the PASCAL2 Network of Excellence, IST-2007-216886. P. H. S. Torr is in receipt of Royal Society Wolfson Research Merit Award.

References

[1] A. Adam, E. Rivlin, and I. Shimshoni. Robust Fragments-based Tracking using the integral Histogram. In *Proc. CVPR*, 2006. 6

[2] S. Avidan. Support Vector Tracking. *IEEE Trans. on PAMI*, 26:1064–1072, 2004. 1

[3] B. Babenko, M. H. Yang, and S. Belongie. Visual Tracking with Online Multiple Instance Learning. In *Proc. CVPR*, 2009. 2, 3, 6

[4] M. B. Blaschko and C. H. Lampert. Learning to Localize Objects with Structured Output Regression. In *Proc. ECCV*, 2008. 2, 3, 4, 6

[5] A. Bordes, L. Bottou, P. Gallinari, and J. Weston. Solving multiclass support vector machines with LaRank. In *Proc. ICML*, 2007. 2, 4

[6] A. Bordes, N. Usunier, and L. Bottou. Sequence Labelling SVMs Trained in One Pass. In *Proc. ECML-PKDD*, 2008. 2, 4, 5

[7] K. Crammer, J. Kandola, R. Holloway, and Y. Singer. Online Classification on a Budget. In *NIPS*, 2003. 2, 5

[8] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object Detection with Discriminatively Trained Part-Based Models. *IEEE Trans. on PAMI*, 32(9):1627–1645, Sept. 2010. 2

[9] P. Gehler and S. Nowozin. On Feature Combination for Multiclass Object Classification. In *Proc. ICCV*, 2009. 7

[10] H. Grabner, M. Grabner, and H. Bischof. Real-time tracking via on-line boosting. In *Proc. BMVC*, 2006. 2, 3, 6, 7

[11] H. Grabner, C. Leistner, and H. Bischof. Semi-supervised on-line boosting for robust tracking. In *Proc. ECCV*, 2008. 2, 3

[12] C. Leistner, A. Saffari, and H. Bischof. MIForests: Multiple-Instance Learning with Randomized Trees. In *Proc. ECCV*, 2010. 6

[13] C. Leistner, A. Saffari, P. M. Roth, and H. Bischof. On Robustness of On-line Boosting - A Competitive Study. In *Proc. ICCV-OLCV*, 2009. 2, 3

[14] H. Masnadi-shirazi, V. Mahadevan, and N. Vasconcelos. On the design of robust classifiers for computer vision. In *Proc. CVPR*, 2010. 2, 3

[15] J. C. Platt. *Fast training of support vector machines using sequential minimal optimization*, pages 185–208. MIT Press, Cambridge, MA, USA, 1999. 4

[16] A. Saffari, M. Godec, T. Pock, C. Leistner, and H. Bischof. Online Multi-Class LPBoost. In *Proc. CVPR*, 2010. 2, 6

[17] A. Saffari, C. Leistner, M. Godec, and H. Bischof. Robust multi-view boosting with priors. In *Proc. ECCV*, 2010. 2, 3

[18] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large Margin Methods for Structured and Interdependent Output Variables. *JMLR*, 6:1453–1484, Dec. 2005. 2, 3

[19] A. Vedaldi, V. Gulshan, M. Varma, and A. Zisserman. Multiple kernels for object detection. In *Proc. ICCV*, 2009. 2

[20] P. Viola and M. J. Jones. Robust real-time face detection. *IJCV*, 57:137–154, 2004. 2, 6

[21] Z. Wang, K. Crammer, and S. Vucetic. Multi-Class Pegasos on a Budget. In *Proc. ICML*, 2010. 2, 5

[22] O. Williams. A sparse probabilistic learning algorithm for real-time tracking. In *Proc. ICCV*, 2003. 2

[23] B. Zeisl, C. Leistner, A. Saffari, and H. Bischof. On-line Semi-Supervised Multiple-Instance Boosting. In *Proc. CVPR*, 2010. 2, 3