Struck: Supplementary Material

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

DERIVATIONS

Converting to the Dual SVM

Starting from the primal SVM, we introduce a Lagrange multiplier $\alpha_i^y \geq 0$ for every i and $y \neq y_i$ that corresponds to the appropriate constraint. We then define the Lagrangian function:

$$\begin{aligned} L(\mathbf{w}, \alpha) &= \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ &\quad - \sum_{i,y\neq y_i} \alpha_i^y \left[\langle \mathbf{w}, \delta\mathbf{\Phi}_i(y)\rangle - \Delta(y, y_i) + \xi_i\right], \end{aligned}$$

in which $\alpha = (\alpha_i^y)$. We are trying to minimise this with respect to $\mathbf{w}$, so we differentiate w.r.t. $\mathbf{w}$ and set to 0:

$$\nabla_{\mathbf{w}} L = \mathbf{w} - \sum_{i,y\neq y_i} \alpha_i^y \delta\mathbf{\Phi}_i(y) = 0$$

Thus:

$$\mathbf{w} = \sum_{i,y\neq y_i} \alpha_i^y \delta\mathbf{\Phi}_i(y)$$

We now substitute this back into L:

$$\begin{aligned} & L(\mathbf{w}, \alpha) \\ = \; & \frac{1}{2}\left\langle \sum_{i,y\neq y_i} \alpha_i^y \delta\mathbf{\Phi}_i(y), \sum_{j,\bar{y}\neq y_j} \alpha_j^{\bar{y}} \delta\mathbf{\Phi}_j(\bar{y}) \right\rangle + C\sum_{i=1}^{n}\xi_i \\ & - \sum_{i,y\neq y_i} \alpha_i^y \left[\left\langle \sum_{j,\bar{y}\neq y_j} \alpha_j^{\bar{y}} \delta\mathbf{\Phi}_j(\bar{y}), \delta\mathbf{\Phi}_i(y) \right\rangle - \Delta(y, y_i) + \xi_i \right] \\ = \; & \frac{1}{2} \sum_{\substack{i,y\neq y_i \\ j,\bar{y}\neq y_j}} \alpha_i^y \alpha_j^{\bar{y}} \delta\mathbf{\Phi}_i(y) \delta\mathbf{\Phi}_j(\bar{y}) + C\sum_{i=1}^{n}\xi_i - \sum_{i,y\neq y_i} \alpha_i^y \xi_i \\ & - \sum_{\substack{i,y\neq y_i \\ j,\bar{y}\neq y_j}} \alpha_i^y \alpha_j^{\bar{y}} \delta\mathbf{\Phi}_i(y) \delta\mathbf{\Phi}_j(\bar{y}) + \sum_{i,y\neq y_i} \alpha_i^y \Delta(y, y_i) \\ = \; & \sum_{i,y\neq y_i} \Delta(y, y_i)\alpha_i^y - \frac{1}{2} \sum_{\substack{i,y\neq y_i \\ j,\bar{y}\neq y_j}} \alpha_i^y \alpha_j^{\bar{y}} \delta\mathbf{\Phi}_i(y) \delta\mathbf{\Phi}_j(\bar{y}) \\ & + \sum_i \left(C - \sum_{y\neq y_i} \alpha_i^y \right) \xi_i \end{aligned}$$

Since we are trying to maximise this with respect to α, we want $C - \sum_{y\neq y_i} \alpha_i^y$ to be ≥ 0 for every i, hence the constraints:

$$\forall i : \sum_{y\neq y_i} \alpha_i^y \leq C$$

We then try and maximise the rest of L with respect to α.

The derivation of the scoring function g is more straightforward:

$$\begin{aligned} g(t, y) &= \langle \mathbf{w}, \mathbf{\Phi}(t, y)\rangle \\ &= \left\langle \sum_{i,\bar{y}\neq y_i} \alpha_i^{\bar{y}} \delta\mathbf{\Phi}_i(\bar{y}), \mathbf{\Phi}(t, y) \right\rangle \\ &= \sum_{i,\bar{y}\neq y_i} \alpha_i^{\bar{y}} \langle \delta\mathbf{\Phi}_i(\bar{y}), \mathbf{\Phi}(t, y)\rangle \end{aligned}$$

Reparameterising the Dual SVM

We derive the reparameterisation of the dual SVM in several steps. The first sum in the maximisation can be straightforwardly rewritten as follows:

$$\begin{aligned} & \sum_{i,y\neq y_i} \Delta(y, y_i)\alpha_i^y \\ = \; & \sum_{i,y\neq y_i} \Delta(y, y_i)(-\beta_i^y) \\ = \; & -\sum_{i,y\neq y_i} \Delta(y, y_i)\beta_i^y \\ = \; & -\underbrace{\Delta(y_i, y_i)}_{=0}\beta_i^{y_i} - \sum_{i,y\neq y_i} \Delta(y, y_i)\beta_i^y \\ = \; & -\sum_{i,y} \Delta(y, y_i)\beta_i^y \end{aligned}$$

The second sum in the maximisation is more involved. We calculate as follows:

$$\begin{aligned} & \sum_{\substack{i,y\neq y_i \\ j,\bar{y}\neq y_j}} \alpha_i^y \alpha_j^{\bar{y}} \langle \delta\mathbf{\Phi}_i(y), \delta\mathbf{\Phi}_j(\bar{y})\rangle \\ = \; & \sum_{\substack{i,y\neq y_i \\ j,\bar{y}\neq y_j}} (-\beta_i^y)(-\beta_j^{\bar{y}}) \langle \delta\mathbf{\Phi}_i(y), \delta\mathbf{\Phi}_j(\bar{y})\rangle \\ = \; & \sum_{\substack{i,y\neq y_i \\ j,\bar{y}\neq y_j}} \left\langle \beta_i^y \delta\mathbf{\Phi}_i(y), \beta_j^{\bar{y}} \delta\mathbf{\Phi}_j(\bar{y}) \right\rangle \\ = \; & \sum_{i,j} \left\langle \sum_{y\neq y_i} \beta_i^y \delta\mathbf{\Phi}_i(y), \sum_{\bar{y}\neq y_j} \beta_j^{\bar{y}} \delta\mathbf{\Phi}_j(\bar{y}) \right\rangle \end{aligned}$$

The last equality in this follows from a property of the inner product:

$$\langle a_1, b_1\rangle + \langle a_2, b_2\rangle = \langle a_1 + a_2, b_1 + b_2\rangle$$

More generally:

$$\sum_{i,j} \langle a_i, b_j \rangle \quad = \quad \left\langle \sum_i a_i, \sum_j b_j \right\rangle$$

Now:

$$\begin{aligned}
& \sum_{y \neq y_i} \beta_i^y \delta \mathbf{\Phi}_i(y) \\
= & \sum_{y \neq y_i} \beta_i^y \left[\mathbf{\Phi}(t_i, y_i) - \mathbf{\Phi}(t_i, y) \right] \\
= & \mathbf{\Phi}(t_i, y_i) \sum_{y \neq y_i} \beta_i^y - \sum_{y \neq y_i} \beta_i^y \mathbf{\Phi}(t_i, y) \\
= & \mathbf{\Phi}(t_i, y_i) \sum_{y \neq y_i} (-\alpha_i^y) - \sum_{y \neq y_i} \beta_i^y \mathbf{\Phi}(t_i, y) \\
= & -\mathbf{\Phi}(t_i, y_i) \sum_{y \neq y_i} \alpha_i^y - \sum_{y \neq y_i} \beta_i^y \mathbf{\Phi}(t_i, y) \\
= & -\mathbf{\Phi}(t_i, y_i) \beta_i^{y_i} - \sum_{y \neq y_i} \beta_i^y \mathbf{\Phi}(t_i, y) \\
= & -\sum_y \beta_i^y \mathbf{\Phi}(t_i, y)
\end{aligned}$$

Plugging this back into the original equation, we get:

$$\begin{aligned}
& \sum_{i,j} \left\langle -\sum_y \beta_i^y \mathbf{\Phi}(t_i, y), -\sum_{\bar{y}} \beta_j^{\bar{y}} \mathbf{\Phi}(t_j, \bar{y}) \right\rangle \\
= & \sum_{i,j} \beta_i^y \beta_j^{\bar{y}} \left\langle \sum_y \mathbf{\Phi}(t_i, y), \sum_{\bar{y}} \mathbf{\Phi}(t_j, \bar{y}) \right\rangle \\
= & \sum_{i,y,j,\bar{y}} \beta_i^y \beta_j^{\bar{y}} \langle \mathbf{\Phi}(t_i, y), \mathbf{\Phi}(t_j, \bar{y}) \rangle
\end{aligned}$$

If desired, this can be rewritten in terms of the joint kernel map k:

$$\sum_{i,y,j,\bar{y}} \beta_i^y \beta_j^{\bar{y}} k(t_i, y, t_j, \bar{y})$$

From the first set of constraints, we get:

$$\begin{aligned}
& \forall i, \forall y \neq y_i : \alpha_i^y \geq 0 \\
\equiv \quad & \forall i, \forall y \neq y_i : (-\beta_i^y \geq 0) \\
\equiv \quad & \forall i, \forall y \neq y_i : \beta_i^y \leq 0
\end{aligned}$$

From the second set of constraints, we get:

$$\begin{aligned}
& \forall i : \sum_{y \neq y_i} \alpha_i^y \leq C \\
\equiv \quad & \forall i : \beta_i^{y_i} \leq C
\end{aligned}$$

Combining these, we get:

$$\forall i, \forall y : \beta_i^y \leq \delta(y, y_i) C$$

The remaining constraints in the reparameterised SVM come directly from the definition of β_i^y.

Determining λ

The goal is to determine, for chosen values m, y_+ and y_-, a value λ by which to increase $\beta_m^{y_+}$ and decrease $\beta_m^{y_-}$. To do this, we consider the effect on the objective function for the dual SVM of modifying $\beta_m^{y_+}$ and $\beta_m^{y_-}$ by an arbitrary λ, and then maximise to find the optimal unconstrained λ, which we call λ^u. We then apply the constraints to find the actual λ. Define $\bar{\beta}_i^y$ to be the new value of β_i^y after the SMO step:

$$\bar{\beta}_i^y = \begin{cases} \beta_i^y + \lambda^u & \text{if } (i,y) = (m, y_+) \\ \beta_i^y - \lambda^u & \text{if } (i,y) = (m, y_-) \\ \beta_i^y & \text{otherwise} \end{cases}$$

Our objective is then:

$$\max_{\lambda^u} \left[-\sum_{i,y} \Delta(y, y_i) \bar{\beta}_i^y - \frac{1}{2} \sum_{i,y,j,\bar{y}} \bar{\beta}_i^y \bar{\beta}_j^{\bar{y}} \, k(t_i, y, t_j, \bar{y}) \right]$$

To solve for λ^u, we differentiate and set to 0 (note that we also multiply through by -1 for simplicity):

$$\frac{\partial}{\partial \lambda^u} \left[\sum_{i,y} \Delta(y, y_i) \bar{\beta}_i^y + \frac{1}{2} \sum_{i,y,j,\bar{y}} \bar{\beta}_i^y \bar{\beta}_j^{\bar{y}} \, k(t_i, y, t_j, \bar{y}) \right] = 0$$

It is simplest to consider the two terms separately. The first term can be decomposed as follows:

$$\begin{aligned}
& \sum_{i,y} \Delta(y, y_i) \bar{\beta}_i^y \\
= \quad & \Delta(y_+, y_m)(\beta_m^{y_+} + \lambda^u) \\
& + \Delta(y_-, y_m)(\beta_m^{y_-} - \lambda^u) \\
& + \sum_{\substack{i,y \\ (i,y) \neq (m,y_+) \\ (i,y) \neq (m,y_-)}} \Delta(y, y_i) \beta_i^y
\end{aligned}$$

If we differentiate this w.r.t. λ^u, we get:

$$\frac{\partial}{\partial \lambda^u} \left[\sum_{i,y} \Delta(y, y_i) \bar{\beta}_i^y \right] \quad = \quad \Delta(y_+, y_m) - \Delta(y_-, y_m)$$

The second term (temporarily ignoring the $\frac{1}{2}$) can similarly be decomposed:

$$\begin{aligned}
& \sum_{i,y,j,\bar{y}} \bar{\beta}_i^y \bar{\beta}_j^{\bar{y}} \, k(t_i, y, t_j, \bar{y}) \\
= \quad & (\beta_m^{y_+} + \lambda^u)^2 \, k(t_m, y_+, t_m, y_+) \\
& + (\beta_m^{y_-} - \lambda^u)^2 \, k(t_m, y_-, t_m, y_-) \\
& + 2(\beta_m^{y_+} + \lambda^u)(\beta_m^{y_-} - \lambda^u) \, k(t_m, y_+, t_m, y_-) \\
& + 2(\beta_m^{y_+} + \lambda^u) \sum_{\substack{i,y \\ (i,y) \neq (m,y_+) \\ (i,y) \neq (m,y_-)}} \beta_i^y \, k(t_i, y, t_m, y_+) \\
& + 2(\beta_m^{y_-} - \lambda^u) \sum_{\substack{i,y \\ (i,y) \neq (m,y_+) \\ (i,y) \neq (m,y_-)}} \beta_i^y \, k(t_i, y, t_m, y_-) \\
& + \sum_{\substack{i,y,j,\bar{y} \\ (i,y) \neq (m,y_+) \\ (i,y) \neq (m,y_-) \\ (j,\bar{y}) \neq (m,y_+) \\ (j,\bar{y}) \neq (m,y_-)}} \beta_i^y \beta_j^{\bar{y}} \, k(t_i, y, t_j, \bar{y})
\end{aligned}$$

If we differentiate this w.r.t. λ^u, we get:

$$\begin{aligned}
&\frac{\partial}{\partial \lambda^u}\left[\sum_{i,y,j,\bar{y}} \bar{\beta}_i^y \bar{\beta}_j^{\bar{y}}\, k(t_i, y, t_j, \bar{y})\right] \\
= \;& (2\lambda^u + 2\beta_m^{y_+})\, k(t_m, y_+, t_m, y_+) \\
&+ (2\lambda^u - 2\beta_m^{y_-})\, k(t_m, y_-, t_m, y_-) \\
&+ 2(\beta_m^{y_-} - \beta_m^{y_+} - 2\lambda^u)\, k(t_m, y_+, t_m, y_-) \\
&+ 2 \sum_{\substack{i,y \\ (i,y)\neq(m,y_+) \\ (i,y)\neq(m,y_-)}} \beta_i^y\, k(t_i, y, t_m, y_+) \\
&- 2 \sum_{\substack{i,y \\ (i,y)\neq(m,y_+) \\ (i,y)\neq(m,y_-)}} \beta_i^y\, k(t_i, y, t_m, y_-)
\end{aligned}$$

We now reintroduce the factor of $\frac{1}{2}$ and add this to the result of differentiating the first term:

$$\begin{aligned}
&\Delta(y_+, y_m) - \Delta(y_-, y_m) \\
&+ (\lambda^u + \beta_m^{y_+})\, k(t_m, y_+, t_m, y_+) \\
&+ (\lambda^u - \beta_m^{y_-})\, k(t_m, y_-, t_m, y_-) \\
&+ (\beta_m^{y_-} - \beta_m^{y_+} - 2\lambda^u)\, k(t_m, y_+, t_m, y_-) \\
&+ \sum_{\substack{i,y \\ (i,y)\neq(m,y_+) \\ (i,y)\neq(m,y_-)}} \beta_i^y\, k(t_i, y, t_m, y_+) \\
&- \sum_{\substack{i,y \\ (i,y)\neq(m,y_+) \\ (i,y)\neq(m,y_-)}} \beta_i^y\, k(t_i, y, t_m, y_-)
\end{aligned}$$

If we now define

$$k_{(++)} = k(t_m, y_+, t_m, y_+),$$

and make analogous definitions of $k_{(--)}$ and $k_{(+-)}$, then this can be simplified to give:

$$\begin{aligned}
&\Delta(y_+, y_m) - \Delta(y_-, y_m) \\
&+ \sum_{i,y} \beta_i^y\, k(t_i, y, t_m, y_+) \\
&- \sum_{i,y} \beta_i^y\, k(t_i, y, t_m, y_-) \\
&+ \lambda^u \left[k_{(++)} + k_{(--)} - 2k_{(+-)}\right]
\end{aligned}$$

Using the definition given for ∇_i^y in the main paper, this can be written as:

$$\nabla_m^{y_-} - \nabla_m^{y_+} + \lambda^u \left[k_{(++)} + k_{(--)} - 2k_{(+-)}\right]$$

Setting this to 0, we find that

$$\lambda^u = \frac{\nabla_m^{y_+} - \nabla_m^{y_-}}{k_{(++)} + k_{(--)} - 2k_{(+-)}},$$

as required. We must then apply the constraints to obtain λ. The first constraint is that $\lambda \geq 0$: we are trying to increase $\beta_m^{y_+}$ and decrease $\beta_m^{y_-}$. The second constraint is that $\lambda \leq C\delta(y_+, y_m) - \beta_m^{y_+}$: this comes from the SVM constraint that $\beta_m^{y_+} \leq \delta(y_+, y_m)C$, since we want the new value of $\beta_m^{y_+}$ (after adding λ) to still satisfy this constraint. Our equation for λ thus becomes

$$\lambda = \max(0, \min(\lambda^u, C\delta(y_+, y_m) - \beta_m^{y_+})),$$

as stated in the main paper.

Updating the Gradients

At the end of an SMO step that modifies the coefficients $\beta_m^{y_+}$ and $\beta_m^{y_-}$, the gradient value ∇_i^y that we store for each support vector (t_i, y) must be appropriately updated. Let $\bar{\nabla}_i^y$ be the new value of ∇_i^y, then:

$$\begin{aligned}
\bar{\nabla}_i^y &= -\Delta(y, y_i) - \sum_{j,\bar{y}} \bar{\beta}_j^{\bar{y}}\, k(t_i, y, t_j, \bar{y}) \\
&= -\Delta(y, y_i) \\
&\quad - (\beta_m^{y_+} + \lambda)\, k(t_i, y, t_m, y_+) \\
&\quad - (\beta_m^{y_i} - \lambda)\, k(t_i, y, t_m, y_-) \\
&\quad - \sum_{\substack{j,\bar{y} \\ (j,\bar{y})\neq(m,y_+) \\ (j,\bar{y})\neq(m,y_-)}} \beta_j^{\bar{y}}\, k(t_i, y, t_j, \bar{y})
\end{aligned}$$

We can then calculate that:

$$\bar{\nabla}_i^y - \nabla_i^y = \lambda\left[k(t_i, y, t_m, y_-) - k(t_i, y, t_m, y_+)\right]$$

CUDA Implementation Details

Computing the SVM's Weight Vector

The SVM's weight vector, $\mathbf{w}$, can be expressed as follows:

$$\mathbf{w} = \sum_{i,\bar{y}} \beta_i^{\bar{y}} \mathbf{\Phi}(t_i, \bar{y})$$

To be able to compute $\mathbf{w}$, we need to know $\mathbf{\Phi}$, which will be the case if and only if we are using a linear kernel for our SVM. Assuming this is the case, we can compute $\mathbf{w}$ efficiently in CUDA by using a single thread block containing a number of threads equal to the size of our feature vectors (see Figure 1). Each thread then computes one element of $\mathbf{w}$, i.e. thread j computes:

$$\mathbf{w}[j] = \sum_{i,\bar{y}} \beta_i^{\bar{y}} \mathbf{\Phi}(t_i, \bar{y})[j] \tag{1}$$

Since all of the threads are in the same thread block, we can load the indices and β 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.

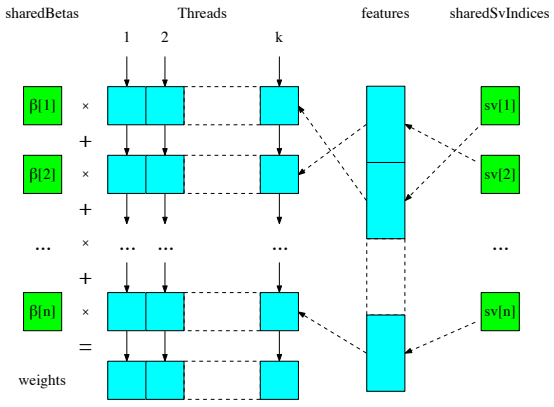


Fig. 1: To compute the weight vector $\mathbf{w}$ for an SVM with a linear kernel efficiently using CUDA, we use a single thread block and compute a single element of $\mathbf{w}$ on each thread. The indices and β coefficients of the support vectors are loaded into shared memory at the start of the computation 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).

Gaussian Kernel Evaluation

Since computing $\mathbf{w}$ is reliant on knowing the feature mapping $\mathbf{\Phi}$, when using a non-linear kernel we cannot use $\mathbf{w}$ and must instead evaluate the SVM directly in terms of its support vectors. For a Gaussian kernel, our scoring function takes the form

$$g(t, y) = \sum_{i,\bar{y}} \beta_i^{\bar{y}} \exp(-\sigma \|\mathbf{x}_{t_i+1}^{\bar{y}(\mathbf{p}_{t_i})} - \mathbf{x}_{t+1}^{y(\mathbf{p}_t)}\|^2). \quad (2)$$

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 $(\mathbf{x}_{t_i+1}^{\bar{y}(\mathbf{p}_{t_i})}[j] - \mathbf{x}_{t+1}^{y(\mathbf{p}_t)}[j])^2$, i.e. the square of the difference between the j'th elements of the two feature vectors. These results are summed using a reduction [1] at the end of each pass and the sum is used to calculate the contribution made by that pass's support vector.

Further Experiments

Multi-scale tracking

To investigate the effects of extending our tracker to deal with scale, we tested a number of multi-scale variants of ThunderStruck on the Wu *et al.* benchmark. The variants we tested varied in the extent to which the scale was allowed to vary between consecutive frames; we also investigated varying the features/kernel used.

The results are shown in Table 1. In comparison with ThunderStruck fkbHG100, they demonstrate that the effects of incorporating scale are somewhat mixed. For sequences exhibiting significant scale variation, a number of the multi-scale variants (e.g. ThunderStruck sHG95_105_1) achieve modest improvements over ThunderStruck fkbHG100 on all but the SRE precision tests; however, over the entire benchmark, there is a gradual fall-off in tracking performance as the extent to which the scale is allowed to vary increases. Moreover, incorporating scale has a significant impact on the speed of the tracker (see §6.5 of the main paper), so there is a trade-off between tracking performance on some sequences and computation time. For these reasons, we believe that the single-scale variants of our tracker currently remain superior to the multi-scale variants.

Effects of structured learning

In addition to the overall results we presented in the main paper (see §6.4), we include here some success/precision plots comparing Struck fkbHG100 with the baseline classification SVM (see Figure 2), and present detailed results on individual subsets of the Wu benchmark (see Table 2).

Our results demonstrate that our structured learning framework outperforms the baseline on every subset of the Wu benchmark. However, the improvements achieved vary somewhat between different subsets. For example, for sequences involving fast motion, some of the results achieved by the baseline are almost as good as those achieved by Struck. This makes some sense, because if the object is moving quickly then some of the benefits of weighting samples based on the extent to which they overlap the central bounding box will be lost. However, in the vast majority of cases, using structured learning significantly improves the results.

In Figure 3, we show some key moments from sequences in which structured learning achieves dramatically better tracking performance than the baseline. In the `basketball_1` sequence, the baseline tracker loses the target player in frame 25 due to an occlusion, whereas the structured learner manages to avoid losing track. Although the baseline tracker does pick up the player again in frame 95 as he moves through its bounding box, this is entirely due to chance and can be seen as artificially inflating the tracking performance of the baseline; moreover, the baseline quickly loses the player again after this, while the structured learner does a much better job of keeping track. It eventually loses track of the player towards the end of the tracking sequence (around frame 645) due to incorrectly latching on to another player on the same team. In the `crossing_1` sequence, the structured learner successfully tracks the person crossing the road to the end of the sequence; by contrast, the baseline classifier performs well until frame 48 and then incorrectly starts tracking the back of a passing car. In the `doll_1` sequence, the baseline temporarily becomes poorly localised around frame 330 due to out-of-plane rotation, and then loses track more seriously around frame 402, despite the fact that nothing particularly significant is happening in the sequence at this point. By contrast, the structured learner keeps track of the doll until

frame 1121, at which point it becomes confused by a poster on the wall behind it. In the `lemming_1` sequence, both trackers perform relatively well until around frame 699, at which point the baseline tracker fails due to the significant motion blur in that part of the sequence. By contrast, the structured tracker survives the motion blur, despite finding it quite challenging (as evidenced by the relatively poor localisation of the bounding box during that part of the sequence).

In Figure 4, we show some key moments from sequences in which the baseline performs better than the structured tracker. In the `david3_1` sequence, both trackers initially lose the target in frame 84 as he passes behind an occluding tree, and then serendipitously reacquire him in frame 192 as he returns in the opposite direction; however, the structured learner then confuses him with the background car and loses track again in frame 199, whereas the baseline classifier continues to successfully track him until the end of the sequence in frame 252. In the `deer_1` sequence, both trackers lose the deer around frame 35 due to the fast motion present in the scene, and then track the background water in different ways. As a result, the baseline classifier subsequently reacquires the deer by chance in frame 45, whereas the structured learner fails to do so. As a result, even though the structured learner briefly reacquires the object again by chance in frame 55, it loses track once again by frame 60 because it spent longer learning the background water than the baseline did. The superior tracking performance of the baseline in this case can thus be traced back to the different ways in which the two trackers tracked the water after they had lost track for the first time: such things essentially boil down to chance. In the `shaking_1` sequence, both trackers perform relatively well until frame 102, at which point the singer's face becomes occluded by his hat. When this happens, the baseline classification SVM shifts to his left shoulder and Struck shifts to his right shoulder. When the singer then raises his head again in frame 104, the baseline classification SVM recovers and returns to tracking his face, whereas Struck continues tracking his right shoulder. We hypothesise that this may be due to Struck's greater ability to adapt to a changing object, in this case with unfortunate consequences. This hypothesis is somewhat borne out by the fact that Struck adapts back to track the face again at around frame 177.

Videos of all the results we show here can be found at https://goo.gl/cJ1Dg7.

Effects of changing the search radii

In order to examine the effects of using different search radii r_L and r_P for learning and propagation, we compared a number of variants of our ThunderStruck tracker (using various settings of r_L and r_P) with the default fkbHG100 variant by running them on the entire Wu benchmark. The results are shown in Table 3. For all variants, we set the unrelated parameters to be the same as for fkbHG100 to ensure a fair comparison (i.e. we used Haar features, a Gaussian kernel and a support vector budget of 100, and set n_R and n_O, respectively the numbers of reprocessing and optimisation steps used for LaRank, to 10).

Our results demonstrate that changing the search radii can have a significant impact on the results, and that it is generally advantageous to set the learning radius r_L to be larger than the propagation radius r_P. Over the entire benchmark, we found that a learning radius of 60 and a propagation radius of 30 (i.e. the settings we use for our default fkbHG100 variant) achieved the best tracking performance. However, for certain types of tracking sequence (notably those in which the target object moves out of the viewing window at some point), we found that better results were obtained when using a larger propagation radius than normal. This is not entirely surprising, since when the target object moves only slightly out of the viewing window, a larger search radius around the target will overlap more with the viewing window than a smaller one and offer more opportunities for correctly relocalising the target as it moves back into view.

Effects of changing the number of LaRank steps

In order to examine the effects of using different values for n_R and n_O (respectively the numbers of reprocessing and optimisation steps used for LaRank), we compared a number of variants of our ThunderStruck tracker (using various settings of n_R and n_O) with the default fkbHG100 variant by running them on the entire Wu benchmark. The results are shown in Table 4. For all variants, we set the unrelated parameters to be the same as those used for fkbHG100 to ensure a fair comparison (i.e. we used Haar features, a Gaussian kernel and a support vector budget of 100, and set the learning search radius r_L to 60 pixels and the propagation search radius r_P to 30 pixels).

Our results demonstrate that by changing the values of n_R and n_O, we can achieve small improvements in tracking performance over fkbHG100. Interestingly, such improvements are possible even when we decrease the values of n_R and n_O, which is significant because using fewer LaRank steps also increases the speed of the tracker. However, in comparison with the difference that changing other parameters (e.g. the search radii) can make, the effect on the tracking performance of changing the number of LaRank steps is in general fairly modest.

Effects of changing the number of Haar features

For our default fkbHG100 variant of Struck, the number of Haar features used (192) is kept deliberately small for speed reasons, and we would naturally expect this to have a negative impact on tracking performance. In order to quantify the actual impact this has, we compared Struck fkbHG100 with a variant that uses a much larger number of Haar features (see Table 5). As for our original Haar features (see §4.7.3 of the main paper), we used 6 types of Haar feature arranged at 2 scales on a grid, but this time we used a 7×7 grid rather than a 4×4 one, giving us new feature vectors of length $7 \times 7 \times 2 \times 6 = 588$.

As we would expect, our results demonstrate that by using a larger number of Haar features, we can significantly improve the performance of our tracker. The increase in performance was especially pronounced for the SRE (i.e. spatial robustness) tests, for which we achieve results that are comparable with the state-of-the-art KCF tracker and only marginally inferior to those of the best-performing MKL variant of our tracker, Struck mklHGHI. This suggests that larger feature vectors on their own can play a significant role in improving the spatial robustness of our tracker, and that they perhaps make a larger difference to the results than multi-kernel learning. However, it was noticeable that for the TRE (i.e. temporal robustness) tests, the improvements in tracking performance we obtained were much more modest.

Comparing Struck mklHGHI with KCF

In the main paper, we show headline figures comparing the overall results of our mklHGHI variant of Struck (which combines a Gaussian kernel with an intersection kernel on Haar and histogram features) with KCF on the Wu *et al.* [2] benchmark, and demonstrate that on all but the TRE success tests, Struck mklHGHI outperforms KCF. However, further insight into the areas in which each of the two trackers performs well can be gleaned by looking at their results on individual subsets of the benchmark (see Table 6). We make the following observations:

1) For sequences exhibiting deformations of the target object or out-of-plane rotations, Struck mklHGHI generally outperforms KCF. We believe this is because the filter learnt by KCF is an average based on the entire tracking sequence up to the current frame, which makes it progressively harder for it to adapt to significant changes in the target object as the number of frames seen increases. By contrast, because Struck maintains explicit support vectors, it is better able to account for different appearances of the object as it deforms or rotates.
2) For sequences containing significant background clutter or motion blur, KCF outperforms Struck mklHGHI. We believe this is because KCF performs dense sampling, giving it a wider range of object candidates from which to choose at each frame.
3) For sequences in which the target object moves out of the viewing window (marked 'Out of View' in Table 6), KCF outperforms Struck mklHGHI by a significant margin. We believe that this is primarily a result of the current implementation of Struck insisting that its target bounding box remain entirely within the viewing window at all times; by contrast, KCF allows its bounding box to leave the viewing window and is thus able to achieve better results.

Overall, the performances achieved by Struck mklHGHI and KCF are fairly comparable on many of the subsets of the Wu benchmark. However, we believe that the differences identified above are significant, and could help to inform the design of new trackers that combine the best aspects of the two approaches.

Examining the Support Vectors

It is interesting to examine the support vectors that are learnt by Struck, as it provides further insight into how our method works. In Figure 5, we show some examples of the support vector set $\mathcal{S}$ after running the tracker on various benchmark sequences. An interesting property that 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, and thereby expresses the foreground 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 the drift during tracking that could occur if old information was discarded.

References

[1] M. Harris, "Optimizing Parallel Reduction in CUDA," *NVIDIA Developer Technology*, vol. 2, p. 45, 2007.
[2] Y. Wu, J. Lim, and M.-H. Yang, "Online Object Tracking: A Benchmark," in *CVPR*, 2013, pp. 2411–2418.

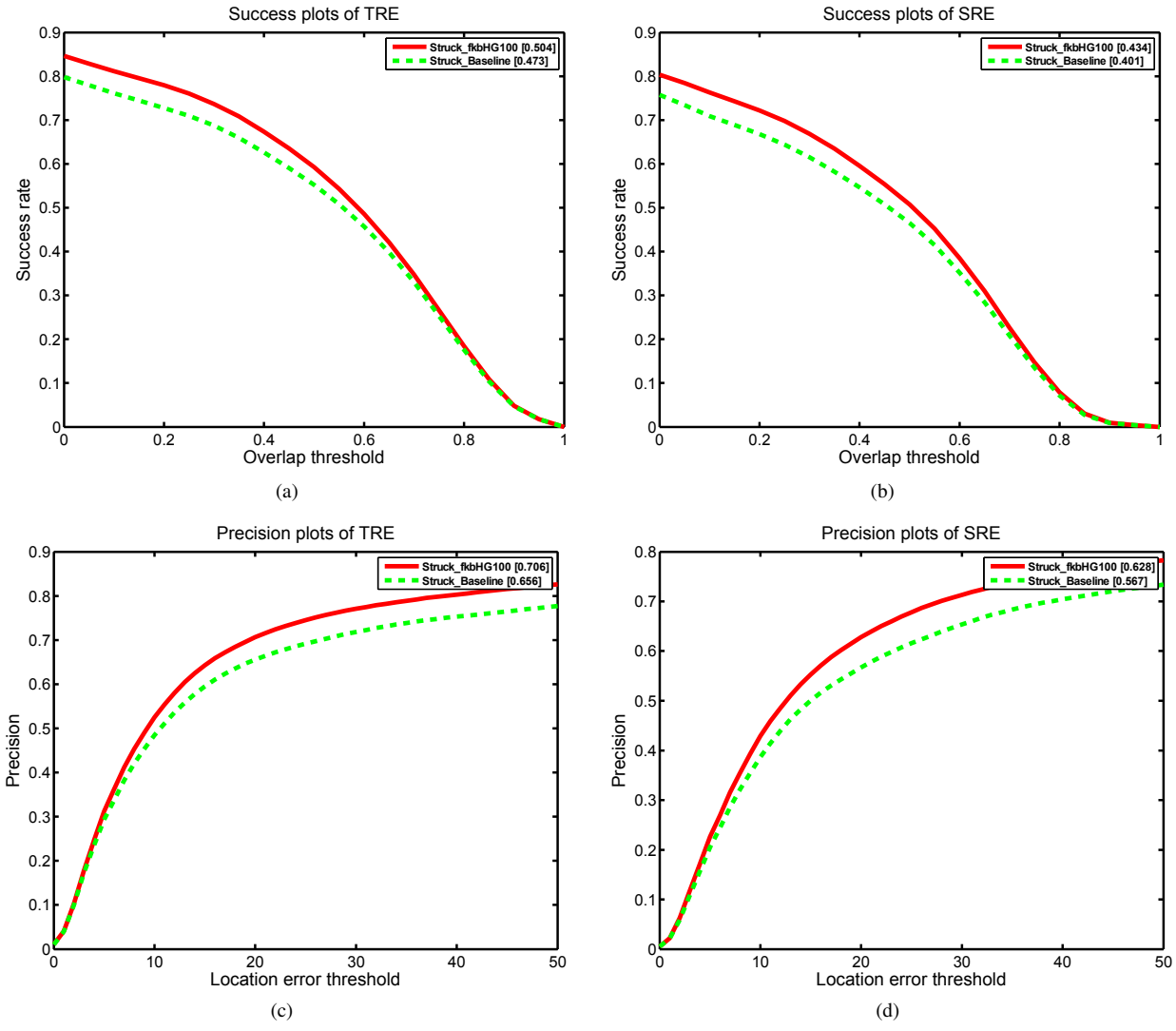


Fig. 2: Comparing the overall results of a baseline classification SVM with the fkbHG100 variant of Struck on the Wu benchmark (see §6.4 in the main paper).

Tracker	Variant	Features/Kernel	Scale Change		Success		Precision		Success (Scale)		Precision (Scale)	
			Min	Max	TRE	SRE	TRE	SRE	TRE	SRE	TRE	SRE
ThunderStruck	sHG95_105_1	Haar/Gaussian	0.95	1.05	0.484	0.406	0.674	0.582	0.451	*0.388*	**0.645**	0.574
ThunderStruck	sHG97_103_1	Haar/Gaussian	0.97	1.03	*0.486*	0.418	*0.676*	*0.597*	0.441	**0.395**	*0.641*	0.582
ThunderStruck	sHG99_101_1	Haar/Gaussian	0.99	1.01	0.487	**0.420**	0.679	**0.603**	0.434	0.382	0.644	*0.580*
ThunderStruck	sRL95_105_1	Raw/Linear	0.95	1.05	0.447	0.383	0.635	0.560	*0.450*	0.384	0.633	0.578
ThunderStruck	sRL97_103_1	Raw/Linear	0.97	1.03	0.466	0.398	0.641	0.575	**0.456**	0.391	0.628	*0.580*
ThunderStruck	sRL99_101_1	Raw/Linear	0.99	1.01	0.472	0.411	0.647	0.581	0.429	0.383	0.614	0.568
ThunderStruck	fkbHG100	Haar/Gaussian	1.00	1.00	**0.490**	*0.417*	**0.681**	0.602	0.424	0.382	0.630	**0.591**

TABLE 1: The tracking performance of multi-scale ThunderStruck variants on the Wu *et al.* [2] benchmark using various feature/kernel combinations and choices of the minimum and maximum scaling factors allowed between consecutive frames (see §4.7.1 in the main paper). The step size between the different scaling factors allowed in each case was always 0.01. For all variants, we used a budget of 100, and search radii of 30 pixels for propagation and 60 pixels for learning, and set n_R and n_O, respectively the numbers of reprocessing and optimisation steps used for LaRank, to 10.

Subset	Success (TRE)		Success (SRE)		Precision (TRE)		Precision (SRE)	
	Baseline	Struck	Baseline	Struck	Baseline	Struck	Baseline	Struck
Background Clutter	0.442	**0.489**	0.378	**0.406**	0.584	**0.652**	0.505	**0.549**
Deformation	0.473	**0.522**	0.376	**0.408**	0.633	**0.701**	0.528	**0.570**
Fast Motion	0.394	**0.403**	0.382	**0.407**	0.510	**0.512**	0.482	**0.517**
Illumination Variation	0.454	**0.489**	0.363	**0.411**	0.603	**0.654**	0.491	**0.572**
In-Plane Rotation	0.424	**0.451**	0.369	**0.404**	0.581	**0.630**	0.510	**0.577**
Low Resolution	0.345	**0.361**	0.281	**0.299**	0.441	**0.471**	0.296	**0.354**
Motion Blur	0.381	**0.407**	0.366	**0.392**	0.482	**0.520**	0.453	**0.491**
Occlusion	0.443	**0.477**	0.376	**0.418**	0.602	**0.657**	0.522	**0.588**
Out-of-Plane Rotation	0.440	**0.474**	0.375	**0.413**	0.620	**0.672**	0.543	**0.603**
Out of View	0.395	**0.403**	0.361	**0.411**	0.443	**0.454**	0.385	**0.454**
Scale Variation	0.417	**0.447**	0.358	**0.400**	0.622	**0.668**	0.543	**0.623**
Overall	0.473	**0.504**	0.401	**0.434**	0.656	**0.706**	0.567	**0.628**

TABLE 2: Comparing the detailed results of a baseline classification SVM with the fkbHG100 variant of Struck on the Wu benchmark (see §6.4 in the main paper). The results demonstrate that the structured learner outperforms the baseline classification SVM on every attribute-based subset of the benchmark.

Tracker	Variant	r_L	r_P	Success		Precision		Success (Out of View)		Precision (Out of View)	
				TRE	SRE	TRE	SRE	TRE	SRE	TRE	SRE
ThunderStruck	lp30_30	30	30	0.477	0.411	0.661	0.592	0.332	0.343	0.326	0.348
ThunderStruck	lp30_60	30	60	0.414	0.356	0.557	0.497	0.306	0.317	0.305	0.336
ThunderStruck	lp60_60	60	60	*0.474*	0.405	*0.650*	0.567	**0.411**	*0.393*	**0.455**	0.435
ThunderStruck	lp60_90	60	90	0.428	0.359	0.586	0.504	0.362	0.358	0.397	0.405
ThunderStruck	lp90_60	90	60	0.470	*0.408*	0.645	*0.569*	*0.393*	**0.412**	0.432	**0.438**
ThunderStruck	lp90_90	90	90	0.446	0.381	0.610	0.528	0.398	0.395	0.432	*0.433*
ThunderStruck	fkbHG100	60	30	**0.490**	**0.417**	**0.681**	**0.602**	0.380	0.369	*0.410*	0.400

TABLE 3: Comparing the tracking performance of the fkbHG100 variant of ThunderStruck on the Wu *et al.* [2] benchmark with variants that change the settings of the learning search radius r_L and the propagation search radius r_P. For all variants, we use Haar features, a Gaussian kernel and a support vector budget of 100, and set n_R and n_O, respectively the numbers of reprocessing and optimisation steps used for LaRank, to 10.

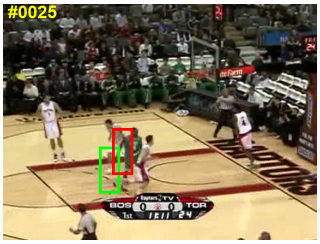
Tracker	Variant	n_R	n_O	Success		Precision	
				TRE	SRE	TRE	SRE
ThunderStruck	ro5_5	5	5	**0.497**	0.420	**0.692**	*0.606*
ThunderStruck	ro5_10	5	10	*0.492*	**0.423**	*0.684*	**0.613**
ThunderStruck	ro10_5	10	5	0.494	**0.423**	0.686	0.607
ThunderStruck	ro15_15	15	15	0.490	0.420	0.681	0.607
ThunderStruck	fkbHG100	10	10	0.490	*0.417*	0.681	0.602

TABLE 4: Comparing the tracking performance of the fkbHG100 variant of ThunderStruck on the Wu *et al.* [2] benchmark with variants that change the settings of n_R and n_O, respectively the numbers of reprocessing and optimisation steps used for LaRank. For all variants, we use Haar features, a Gaussian kernel and a support vector budget of 100, and set the learning search radius r_L to 60 pixels and the propagation search radius r_P to 30 pixels.

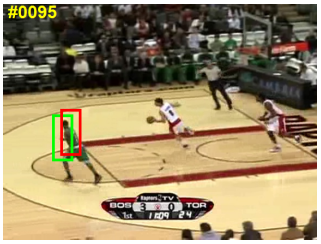
Tracker	Variant	Feature Count	Success		Precision	
			TRE	SRE	TRE	SRE
Struck	fcH588	588	*0.528*	*0.461*	*0.749*	0.690
Struck	fkbHG100	192	0.504	0.434	0.706	0.628
KCF	–	–	**0.556**	0.463	0.774	*0.683*
Struck	fkbHI100	480	0.517	0.455	0.734	0.679
Struck	mklHGHI	672	0.545	**0.469**	**0.785**	**0.707**

TABLE 5: Comparing the tracking performance of the fkbHG100 variant of ThunderStruck on the Wu *et al.* [2] benchmark with a variant that uses a much larger number of Haar features. For this variant, we use a Gaussian kernel and a support vector budget of 100, and set the learning search radius r_L to 60 pixels and the propagation search radius r_P to 30 pixels. The results of KCF and some relevant variants of Struck are also shown, for comparison purposes.

(a) Tracking results after the first few frames

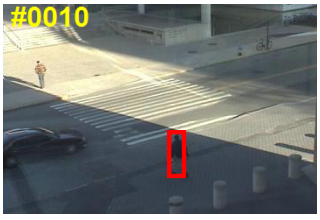


(b) The baseline loses track after an occlusion

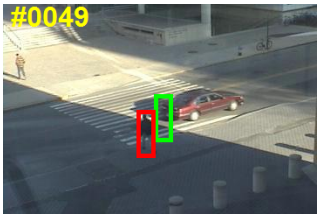


(c) The baseline reacquires track through good fortune

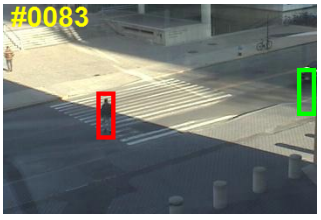
(d) Struck gets confused by a player on the same team



(e) Tracking results after the first few frames



(f) The baseline gets confused by a car



(g) The baseline tracks the car to the edge of the frame

(h) Struck tracks the real target to the end of the sequence

(i) Tracking results after the first few frames

(j) The baseline is poorly-localised due to out-of-plane rotation

(k) The baseline loses track

(l) Struck gets confused by the poster

(m) Tracking results after the first few frames

(n) The baseline loses track due to motion blur

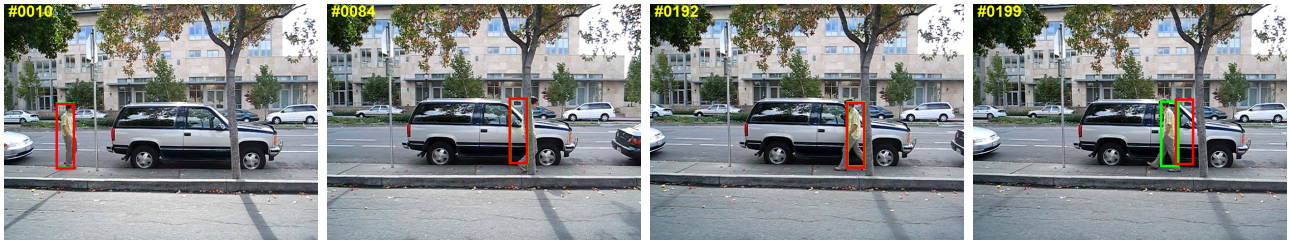
(o) Struck loses track due to out-of-plane rotation

(p) Struck reacquires track through good fortune

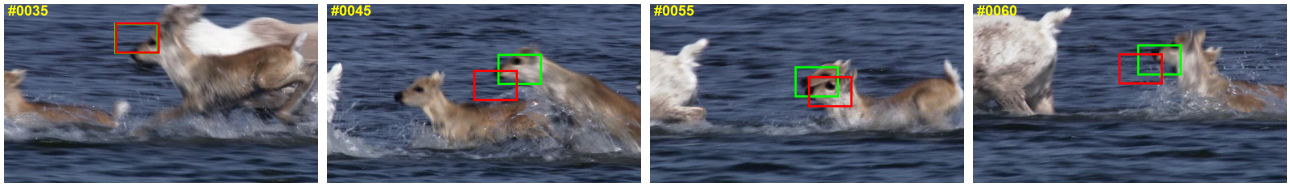
Fig. 3: Key moments from sequences for which Struck fkbHG100 (red) significantly outperforms a baseline classification SVM (green). First row: `basketball_1`. Second row: `crossing_1`. Third row: `doll_1`. Fourth row: `lemming_1`.

Subset	Success (TRE)		Success (SRE)		Precision (TRE)		Precision (SRE)	
	KCF	Struck mklHGHI	KCF	Struck mklHGHI	KCF	Struck mklHGHI	KCF	Struck mklHGHI
Background Clutter	**0.564**	0.526	**0.484**	0.453	**0.776**	0.730	**0.694**	0.653
Deformation	0.570	**0.576**	0.469	**0.473**	0.757	**0.796**	0.677	**0.686**
Fast Motion	**0.455**	0.440	0.414	**0.425**	**0.579**	0.568	0.545	**0.549**
Illumination Variation	0.527	**0.537**	0.442	**0.444**	0.729	**0.748**	0.652	**0.654**
In-Plane Rotation	**0.519**	0.508	**0.450**	0.443	0.728	**0.733**	**0.667**	0.665
Low Resolution	0.383	**0.387**	0.290	**0.292**	**0.502**	0.501	**0.377**	0.330
Motion Blur	**0.492**	0.454	**0.424**	0.412	**0.627**	0.589	**0.567**	0.529
Occlusion	**0.546**	0.530	0.446	**0.467**	**0.758**	0.749	0.663	**0.681**
Out-of-Plane Rotation	**0.530**	0.523	0.445	**0.449**	0.749	**0.766**	0.666	**0.686**
Out of View	**0.538**	0.443	**0.455**	0.427	**0.643**	0.513	**0.534**	0.488
Scale Variation	**0.488**	0.474	0.401	**0.405**	0.728	**0.733**	0.631	**0.659**
Overall	**0.556**	0.545	0.463	**0.469**	0.774	**0.785**	0.683	**0.707**

TABLE 6: Comparing the detailed results of Struck mklHGHI (a multi-kernel variant of our tracker that combines a Gaussian kernel with an intersection kernel on Haar and histogram features) with KCF on the Wu benchmark.



(a) Tracking results after the first few frames (b) Both trackers lose track due to the occluding tree (c) Both trackers reacquire track through good fortune (d) Struck gets confused by the background car



(e) Both trackers lose track due to fast motion (f) The baseline reacquires track through good fortune (g) Struck reacquires track through good fortune (h) Struck loses track again due to having learnt the background

(i) Tracking results after the first few frames (j) Both trackers become poorly-localised due to occlusion (k) The baseline starts to resume tracking the face, whereas Struck now tracks the right shoulder (l) Struck resumes tracking the face again

Fig. 4: Key moments from sequences for which a baseline classification SVM (green) outperforms Struck fkbHG100 (red). First row: `david3_1`. Second row: `deer_1`. Third row: `shaking_1`.

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

Fig. 5: Visualisation of the support vector set $\mathcal{S}$ after running Struck on various benchmark sequences with a budget of 64 support vectors (chosen for illustrative purposes). 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.