Chapter 1

Fast structured tracker with improved motion model using robust Kalman filter

Ivan Bogun* and Eraldo Ribeiro†

Computer Vision Laboratory,
Department of Computer Sciences
Florida Institute of Technology,
Melbourne, FL 32901, U.S.A.
eribeiro@cs.fit.edu

In this chapter, we discuss the problem of tracking objects without having to learn a model of the object's appearance in advance (i.e., model-free tracking). Object tracking has many practical applications, which motivated continous developments in the field. A particularly successful tracking technique that has been recently proposed is the structured tracker called *Struck*. Struck uses support vector machines to learn the object's appearance as well as to predict its location from one video frame to the next. However, tracking remains a challenging problem due to change of lighting conditions throughout the video, change of scale, rotation of the object of the interest, and occlusions. Here, we present a tracker that is based on Struck. Structured learning is a suitable framework for tracking as it calculates a discriminative function mapping the appearance information from one video frame onto the bounding box containing the tracking object in another video frame. We discuss a number of concepts that help keep limit computational complexity throughout tracking: budget for support vectors and fast computation of the intersection kernel. We then show how the use of the Robust Kalman filter can help smooth tracking results, and consequently make the tracker robust to false-negative detections. We show how the Robust Kalman Filter helps detect short-time occlusions by classifying them as outliers. We discuss the tracking evaluation using different evaluation protocols, and present experimental results that show that the structured tracker with the robust filter improves upon original Struck tracker regardless of the features or kernels used.

1. Introduction

Model-free tracking is one of the earliest problem in computer vision. Problem settings are simple: given a bounding box in the first frame, find the position of the tracked object in consecutive frames. Tracking applications include surveillance, video analysis, and traffic monitoring. Nevertheless, tracking remains a hard problem as the object's appearance vary considerably due to changes of illumination, rotation, and occlusion. In addition to

*E-mail: ibogun2010@my.fit.edu
†Corresponding author. E-mail: eribeiro@cs.fit.edu

these variations in appearance, the computational cost of tracking must be low as real-time performance is often desired.

Recent benchmarks[1,2] showed that state-of-the-art trackers *adapt* to the changing appearance of the tracked object. This adaptability is needed because as objects deform and rotate, matching them to a fixed-appearance template inevitably become impossible. In contrast with fixed-template trackers, adaptive-template trackers have much larger generalization power. The tracker proposed by Zhong et al.[3] combines a holistic template with a local representation of the object in each frame. The template remains constant while the local representation is updated in each video frame. The tracker by Kalal et al.[4] adds a new appearance patch to the object model whether the label from a nearest-neighbor classifier is different from that of temporal and spatial experts. The tracker known as Struck, proposed by Hare et al.,[5] processes each new prediction as a new support vector in an online structural SVM framework. The weight of each such support vector is calculated as contribution to the objective function. Other characteristic of state-of-the art trackers is their robustness to partial or complete occlusions.

In,[3] occlusion handling is done by thresholding the reconstruction error between the object and a sparse combination of local object appearances of the model. Here, these local appearances are called *patches*. The contribution of each patch to the object reconstruction is controlled by a coefficient (i.e., the patch's sparse coefficient). If the patch is deemed occluded, its sparse coefficient is set to zero, which gives the tracker the ability to handle partial occlusions. The tracker in[4] uses a different strategy for detecting occlusions. Here, the tracked object is assumed to be occluded if two conditions are simultaneously satisfied: (a) the forward-backward optical flow error in the bounding box is high, and (b) the nearest-neighbor classifier returns a value below than a predefined threshold. A key drawback of the method is that thresholding cannot be made robust because it is impossible to distinguish between the appearance of a novel (unseen) object and occlusions.

Instead of using thresholds, we propose to combine the tracker by,[5] which achieved the highest discriminative power in recent benchmarks,[1,2] with the Robust Kalman filter.[6] This combination increases tracker's robustness to false positives and short-time occlusions. We achieve that by comparing the results of the detector and filter in every frame. Based on the agreement between the detector and the robust Kalman filter, we either update the tracker or look for the object in an extended search region, allowing the tracker to recover from the occlusion or incorrect detection. We test our tracker on various benchmarks,[1,2] and compare it to the original Struck tracker[5] and other state-of-the-art trackers.[3,4,7]

2. Related work

Recent benchmarks[1,2] showed that adapting to new object appearances is a key feature of state-of-the-art trackers. From a machine-learning perspective, such an adaptation is known as online learning, where the data comes in a stream. Previously, online variants of multiple-instance learning,[8] structured SVM,[5] sparsity-collaborative model[3] were used for tracking. The structured tracker,[5] known as Struck, deserves a special mention as it

achieved state-of-the-art results on main benchmarks[1,2] and is one of ingredients of our work. The tracker is based on the structured SVM, introduced by.[9] It is designed as a max-margin classifier, which is learnt in an online manner. To overcome the curse of kernelization only a few support vectors (i.e., a budget) is kept at any given time.

Another ingredient in our work is the robust Kalman filter.[6] The robust Kalman filter was used for tracking by Nguyen and Smeulders.[10] However, in their work, the filter is used to temporally smooth a feature template, which is used to locate the object. It was also shown that the robust Kalman filter is robust to noise and severe occlusions.

In the method described in this chapter, we combine the robust Kalman filter with a state-of-the-art tracking-by-detection tracker[5] to increase resilience towards partial occlusions and false positives. In our approach, the location of the bounding boxes is smoothed with the robust Kalman filter, under a constant-velocity model, and the result is compared to the best detection produced by the tracker. If the detector makes a mistake, it is likely that the filter prediction will disagree with the detector's. In such scenario, the tracker has an increased chance of making a mistake, thus we propose not to update the tracker whether it disagrees with the filter, and if that is the case, the tracker should search for the best location in the next frame in the neighborhood of tracker's and filter's predicted locations.

3. Structured learning

3.1. *Tracking as structured learning*

Structured learning is the problem of learning dependencies between arbitrary input and output spaces. Tracking is the problem of *localizing* the object in the image. One way to parametrize spaces in tracking is as follows: let $\mathcal{X} \in \mathcal{R}^{n \times m}$ be the matrix of pixels, while $\mathcal{Y} \in \mathcal{R}^4$ be the space of bounding boxes defined on the image X which are parametrized by the center of the bounding box and it's dimensions, $y = [c_x, c_y, w, h] \in \mathcal{Y}$. Since the goal is to find location of the object we need to learn the function in the form :

$$f : \mathcal{X} \rightarrow \mathcal{Y}. \tag{1.1}$$

Following,[11] we assume that function f depends on a parameter vector $\mathbf{w}$ and is in the form:

$$f(\mathbf{x}; \mathbf{w}) = \arg\max_{\mathbf{y} \in \mathcal{Y}} F(\mathbf{x}, \mathbf{y}; \mathbf{w}), \tag{1.2}$$

where $F : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{R}$ is a *discriminative function* that measures compatibility between pairs $(\mathbf{x}, \mathbf{y})$. Function F is assumed to be linear in combined-feature representation in terms of the parameter vector $\mathbf{w}$:

$$F(\mathbf{x}, \mathbf{y}; \mathbf{w}) = \langle \mathbf{w}, \mathbf{\Phi}(\mathbf{x}, \mathbf{y}) \rangle. \tag{1.3}$$

In tracking, function $\mathbf{\Phi}$ is defined as a feature vector calculated over a bounding box extracted from the full image, i.e., let $\mathbf{y} = [c_x, c_y, w, h]$ be a bounding box and function

g be a function that calculates a feature vector from the image (e.g., similar to calculating Haar features), then

$$\mathbf{\Phi}(\mathbf{x}, \mathbf{y}) = g(\mathbf{x}|_{\mathbf{y}}). \tag{1.4}$$

Here, $\mathbf{x}|_{\mathbf{y}}$ denotes the operation of cropping bounding box $\mathbf{y}$ from image $\mathbf{x}$. Note that g can be any function that maps a matrix of pixels onto a feature vector. Next, we will show how function F could be learned for tracking.

3.2. *Structured SVM*

It was shown in[11] how *support vector machines* could be generalized to learn functions in arbitrary, *structured* spaces. Let $(\mathbf{x}_i, \mathbf{y}_i)_{i=1}^n$ be a set of input-output pairs, then the structured SVM solves the following convex optimization problem:

$$\begin{aligned} \min_{w} \quad & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i \\ \text{s.t} \quad & \forall i : \xi_i \geq 0 \\ & \forall i, \forall \mathbf{y} \neq \mathbf{y}_i : \langle \mathbf{w}, \delta\mathbf{\Phi}_i(\mathbf{y})\rangle \geq \Delta(\mathbf{y}_i, \mathbf{y}) - \xi_i, \end{aligned} \tag{1.5}$$

where $\delta\mathbf{\Phi}_i(\mathbf{y}) = \mathbf{\Phi}(\mathbf{x}_i, \mathbf{y}_i) - \mathbf{\Phi}(\mathbf{x}_i, \mathbf{y})$. Function $\Delta(\cdot, \cdot)$ is called a *loss* function. For tracking, the loss function is defined by the one minus intersection-over-union metric, i.e.:

$$\Delta(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \frac{|\mathbf{y} \cap \hat{\mathbf{y}}|}{|\mathbf{y} \cup \hat{\mathbf{y}}|}. \tag{1.6}$$

The loss function is designed to penalize bounding boxes that have low overlap with the ground truth given their size.

Often, the optimization problem in Eq. 1.5 is easier to solve in its dual form. The conversion to the dual can be done by setting up a Lagrangian. The dual, in new variables, has the form:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i, \mathbf{y} \neq \mathbf{y_i}} \Delta(\mathbf{y}, \mathbf{y}_i)\alpha_i^{\mathbf{y}} - \frac{1}{2}\sum_{\substack{i, \mathbf{y} \neq \mathbf{y_i} \\ j, \bar{\mathbf{y}} \neq \mathbf{y_i}}} \alpha_i^{\mathbf{y}}\alpha_j^{\bar{\mathbf{y}}}\langle \delta\mathbf{\Phi}_i(\mathbf{y}), \delta\mathbf{\Phi}_j(\bar{\mathbf{y}})\rangle \\ \text{s.t} \quad & \forall i, \forall \mathbf{y} \neq \mathbf{y}_i : \alpha_i^{\mathbf{y}} \geq 0 \\ & \forall i : \sum_{\mathbf{y} \neq \mathbf{y}_i} \alpha_i^{\mathbf{y}} \leq C. \end{aligned} \tag{1.7}$$

As suggested by,[12] the problem in Eq.1.7 can be simplified with the following parametrization:

$$\beta_i^{\mathbf{y}} = \begin{cases} -\alpha_i^{\mathbf{y}}, & \text{if } \mathbf{y} \neq \bar{\mathbf{y}} \\ \sum_{\bar{\mathbf{y}} = \mathbf{y}_i} \alpha_i^{\bar{\mathbf{y}}} & \text{otherwise.} \end{cases} \tag{1.8}$$

Using new variables, the dual problem can be rewritten as:

$$\max_{\beta} \quad \sum_{i,\mathbf{y}} \Delta(\mathbf{y}, \mathbf{y}_i)\beta_i^{\mathbf{y}} - \frac{1}{2} \sum_{i,\mathbf{y},j,\bar{\mathbf{y}}} \beta_i^{\mathbf{y}} \beta_j^{\bar{\mathbf{y}}} \langle \mathbf{\Phi}(\mathbf{x}_i, \mathbf{y}), \mathbf{\Phi}(\mathbf{x}_j, \bar{\mathbf{y}}) \rangle \tag{1.9}$$

$$\text{s.t} \quad \forall i, \forall \mathbf{y} : \beta_i^{\mathbf{y}} \leq \delta(\mathbf{y}, \mathbf{y}_i) C \tag{1.10}$$

$$\forall i : \sum_{\mathbf{y}} \beta_i^{\mathbf{y}} = 0, \tag{1.11}$$

where $\delta(\mathbf{y}, \bar{\mathbf{y}})$ is 1 if $\mathbf{y} = \bar{\mathbf{y}}$ or 0 otherwise. In new variables discriminative function has the form:

$$F(\mathbf{x}, \mathbf{y}; \beta) = \sum_{i,\bar{\mathbf{y}}} \beta_i^{\bar{\mathbf{y}}} \langle \mathbf{\Phi}(\mathbf{x_i}, \bar{\mathbf{y}}), \mathbf{\Phi}(\mathbf{x}, \mathbf{y}) \rangle. \tag{1.12}$$

Because discriminative function depends only on pairs $(\mathbf{x}_i, \mathbf{y})$ for which $\beta_i^{\mathbf{y}}$ is not zero, they are called *support vectors*. An important corollary from Eqs. 1.10 is that, for every $\mathbf{x}_i$, there can be at most one positive coefficient $\beta_i^{\mathbf{y}} > 0$; call such vectors *positive* support vectors. In tracking, this means that each frame can have at most one true location of the object. Support vectors whose coefficients satisfy $\beta_i^{\mathbf{y}} < 0$ are called *negative*. Another important property is that the dual and the discriminative function only depend on the dot product of the feature mapping $\mathbf{\Phi}$ and, as a result, the kernel trick can be applied. If we set

$$K(\mathbf{x}, \mathbf{y}, \bar{\mathbf{x}}, \bar{\mathbf{y}}) = \langle \mathbf{\Phi}(\mathbf{x}, \mathbf{y}), \mathbf{\Phi}(\bar{\mathbf{x}}, \bar{\mathbf{y}})) \rangle, \tag{1.13}$$

we can use different higher dimensional embeddings by replacing the kernel function.

3.3. *Online algorithm for structured SVM*

In tracking training instances are not known in advance but are given in the stream of data, thus solving optimization problem 1.9,1.10,1.11 directly is not feasible. Instead, an SMO-step algorithm[13,14] based on gradient descent is used which requires only a single pass over the data. To do so find the gradient of the objective function in 1.9:

$$g_i(\mathbf{y}) = -\Delta(\mathbf{y}, \mathbf{y}_i) - \sum_{j,\bar{\mathbf{y}}} \beta_j^{\bar{\mathbf{y}}} K(\mathbf{x}_j, \bar{\mathbf{y}}, \mathbf{x}_i, \mathbf{y}) \tag{1.14}$$

Let $S = \{(\mathbf{x}_i, \mathbf{y}) | \beta_i^{\mathbf{y}} \neq 0\}$ be a set of all support vectors. For a pair of coefficients, $\beta_i^{\mathbf{y}+}$ and $\beta_i^{\mathbf{y}-}$, SMO-step performs one dimensional maximization and changes each by adding and subtracting λ respectively. Since the change is the same for both inequality 1.11 remains satisfied. To ensure that inequality 1.10 is satisfied as well update is performed only if $\lambda > 0$ and if the amount is not more than $C\delta(\mathbf{y}_+, \mathbf{y}_-) - \beta_i^{\mathbf{y}+}$. Finally, to maintain a valid set of gradients set S has to be updated with the new values of $\beta_i^{\mathbf{y}-}, \beta_i^{\mathbf{y}+}$. A pseudo-code for the SMO-step procedure is given in Alg. 1.1

Algorithm 1.1: SMO-step

input: $i, \mathbf{y}_+, \mathbf{y}_-$

$k_{00} = \langle \mathbf{\Phi}(\mathbf{x}_i, \mathbf{y}_+), \mathbf{\Phi}(\mathbf{x}_i, \mathbf{y}_+) \rangle$;

$k_{11} = \langle \mathbf{\Phi}(\mathbf{x}_i, \mathbf{y}_-), \mathbf{\Phi}(\mathbf{x}_i, \mathbf{y}_-) \rangle$;

$k_{01} = \langle \mathbf{\Phi}(\mathbf{x}_i, \mathbf{y}_+), \mathbf{\Phi}(\mathbf{x}_i, \mathbf{y}_-) \rangle$;

$\lambda^u = \frac{g_i(\mathbf{y}_+) - g_i(\mathbf{y}_-)}{k_{00} + k_{11} - 2k_{01}}$;

$\lambda = \max(0, \min(\lambda^u, C\delta(\mathbf{y}_+, \mathbf{y}_-) - \beta_i^{\mathbf{y}_+}))$;

$\beta_i^{\mathbf{y}_+} \leftarrow \beta_i^{\mathbf{y}_+} + \lambda$;

$\beta_i^{\mathbf{y}_-} \leftarrow \beta_i^{\mathbf{y}_-} - \lambda$;

for $(\mathbf{x}_j, \mathbf{y}) \in S$ **do**

  $k_0 = K(\mathbf{x}_j, \mathbf{y}, \mathbf{x}_i, \mathbf{y}_+)$;

  $k_1 = K(\mathbf{x}_j, \mathbf{y}, \mathbf{x}_i, \mathbf{y}_-)$;

  $g_j(\mathbf{y}) \leftarrow g_j(\mathbf{y}) - \lambda(k_0 - k_1)$;

Due to hard computational constraints of the tracking not all data instances are passed to the SMO-step. To ensure that with the minimum calls to SMO-step structured SVM would have the highest discriminative ability, three update steps are considered: ProcessNew Alg. 1.2, ProcessOld Alg. 1.3 and Optimize Alg. 1.4. ProcessNew corresponds to the case where new positive support vector is added to the set S. In this step we also search for the most negative support vector to use it in SMO-step. In ProcessOld procedure we randomly choose a frame to update support vectors. Condition $\beta_i^{\mathbf{y}} < C\delta(\mathbf{y}, \mathbf{y}_i)$ ensures that only existing support vectors are considered. Optimize works similarly to ProcessOld.

Algorithm 1.2: ProcessNew

input: $\mathbf{x}_i, \mathbf{y}_i$

$\mathbf{y}_+ = \mathbf{y}_i$;

$\mathbf{y}_- = \arg\min_{\mathbf{y} \in \mathcal{Y}} g_i(\mathbf{y})$;

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

Algorithm 1.3: ProcessOld

input:

Randomly sample $\mathbf{x}_i$;

$\mathbf{y}_+ = \arg\max_{\mathbf{y} \in \mathcal{Y}} g_i(\mathbf{y})$ subject to $\beta_i^{\mathbf{y}} < C\delta(\mathbf{y}, \mathbf{y}_i)$;

$\mathbf{y}_- = \arg\min_{\mathbf{y} \in \mathcal{Y}} g_i(\mathbf{y})$;

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

Algorithm 1.4: Optimize

input:

1 Randomly sample $\mathbf{x}_i$;

2 Set $\mathcal{Y}_i = \{\mathbf{y} \in \mathcal{Y} | (\mathbf{x}_i, \mathbf{y}) \in S)\}$;

3 $\mathbf{y}_+ = \arg\max_{\mathbf{y} \in \mathcal{Y}_i} g_i(\mathbf{y})$ subject to $\beta_i^{\mathbf{y}} < C\delta(\mathbf{y}, \mathbf{y}_i)$;

4 $\mathbf{y}_- = \arg\min_{\mathbf{y} \in \mathcal{Y}_i} g_i(\mathbf{y})$;

5 SMO-step($i, \mathbf{y}_+, \mathbf{y}_-$)

3.4. *Budget*

The cost of evaluating discriminative function F is proportional to the number of support vectors, which will increase with each frame. To bound this complexity an upper bound on how many support vectors is kept at any given time is used, known as *budget*. To keep the number of support vectors fixed the following strategy is used: calculate contribution of each support vector to the primal objective function and if new support vector has to be added remove the one whose contribution is the lowest.[5,15] After removal few coefficients have to be adjusted so that inequalities 1.10,1.11 remain satisfied. Formally, let $(\mathbf{x}_r, \mathbf{y})$ be support vector then it's removal will change weight vector by

$$\bar{\mathbf{w}} = \mathbf{w} - \beta_r^{\mathbf{y}} \mathbf{\Phi}(\mathbf{x}_r, \mathbf{y}) + \beta_r^{\mathbf{y}} \mathbf{\Phi}(\mathbf{x}_r, \mathbf{y}_r). \tag{1.15}$$

The contribution to the primal objective is then

$$||\Delta \mathbf{w}||^2 = (\beta_r^{\mathbf{y}})^2 \{\langle \mathbf{\Phi}(\mathbf{x}_r, \mathbf{y}), \mathbf{\Phi}(\mathbf{x}_r, \mathbf{y}) \rangle + \langle \mathbf{\Phi}(\mathbf{x}_r, \mathbf{y}_r), \mathbf{\Phi}(\mathbf{x}_r, \mathbf{y}_r) \rangle - 2\langle \mathbf{\Phi}(\mathbf{x}_r, \mathbf{y}), \mathbf{\Phi}(\mathbf{x}_r, \mathbf{y}_r) \rangle \tag{1.16}$$

Whenever the budget is exceeded, i.e. $|S| > B$ where B is the budget the support vector containing the lowest contribution to the objective is deleted.

Complete tracking algorithm is summarized int the Alg. 1.5. On line 1 the algorithm finds the best bounding box containing the object. After that tracker is updated by applying SMO-step with different arguments. Parameters n_R, n_O control the complexity of the tracker, the higher the values the more accurate the tracker can get while getting slower. We set $n_R = 10, n_O = 10$.

Algorithm 1.5: Struck

input: S_{t-1}

$\mathbf{y}_t = \arg\max_{\mathbf{y} \in \mathcal{Y}} F(\mathbf{x}_t, \mathbf{y})$;
$(i, \mathbf{y}_+, \mathbf{y}_-) \leftarrow$ ProcessNew$(\mathbf{x}_t, \mathbf{y}_t)$;
SMOStep$(i, \mathbf{y}_+, \mathbf{y}_-)$;
BudgetMaintaince() ;
for $j = 1$ *to* n_R **do**
    $(i, \mathbf{y}_+, \mathbf{y}_-) \leftarrow$ ProcessOld() ;
    SMOStep$(i, \mathbf{y}_+, \mathbf{y}_-)$;
    BudgetMaintaince() ;
    for $i = 1$ *to* n_O **do**
        $(i, \mathbf{y}_+, \mathbf{y}_-) \leftarrow$ Optimize() ;
        SMOStep$(i, \mathbf{y}_+, \mathbf{y}_-)$;
        BudgetMaintaince() ;

4. The Robust Kalman filter

Drifting is a common problem for many trackers. It occurs when a tracker makes a mistake either due to false negative detection or occlusion. Trackers utilizing tracking-by-detection paradigm are especially susceptible to drifting because due to complexity considerations they sample bounding boxes only in the neighborhood of the current object location. As the result, one bad detection can lead to the loss of a track. We observed that often during false positives tracker chooses locations which are not consistent with the motion model of the object. For example, a car driving one direction cannot suddenly move the opposite one. In our method we use motion consistency to detect false positives detections or occlusions. The method consists of using detector which builds objet's model in online manner and the Kalman filter to spatially smooth results of the detector. Unfortunately, classical Kalman filter is unable to perform well if tracker's detection was incorrect as it cannot deal with outliers. Instead, we use the robust Kalman filter which is designed to be resilient to outliers.

We follow the Robust Kalman filter as presented in.[6] Here, the gain step is replaced by a hubernization of the step. Formally, let $\mathbf{y}_t$ be ground truth unobserved state at time t, then it is related to the state at previous time by the equation:

$$\mathbf{y}_t = F\mathbf{y}_{t-1} + v_t, \tag{1.17}$$

where F is a transition matrix, and $v_t \sim N(0, Q)$ is a Gaussian noise. Since $\mathbf{y}_t$ is unobserved, we only observe their noisy estimates, denoted by $\hat{\mathbf{y}}_t$, called *observed* state. The noisy estimates are related to the real ones by a linear transformation:

$$\hat{\mathbf{y}}_t = H\mathbf{y}_t + \epsilon_t, \tag{1.18}$$

where H is a matrix that transforms latent variables into observed ones. The noise $\epsilon_t \sim N(0, R)$ is also assumed to be Gaussian. Let $\mathbf{y}_{t|s}$ be the minimum least-square

error estimate of the location at frame t given locations in each of the frames $1, ..., s$, while $\Sigma_{t|s}$ is a covariance of the error estimate $\mathbf{y}_{t|s}$, then Kalman filter prediction step is given by:

$$\mathbf{y}_{t|t-1} = F\mathbf{y}_{t-1|t-1} \tag{1.19}$$

$$\Sigma_{t|t-1} = F\Sigma_{t-1|t-1}F^T + Q. \tag{1.20}$$

Correction step is used to update current prediction by taking into account new measurement:

$$\mathbf{y}_{t|t} = \mathbf{y}_{t|t-1} + h(\underbrace{K\Delta\hat{\mathbf{y}}_t}_{\text{correction}}) \tag{1.21}$$

$$\Delta\hat{\mathbf{y}}_t = \hat{\mathbf{y}}_t - H\mathbf{y}_{t|t-1} \tag{1.22}$$

$$K = \Sigma_{t|t-1}H\Delta_t^{-1} \tag{1.23}$$

$$\Sigma_{t|t} = (\mathbb{I} - KH)\Sigma_{t|t-1} \tag{1.24}$$

$$\Delta_t = H\Sigma_{t|t-1}H^T + R \tag{1.25}$$

Hubernization is achieved by applying the function $h_b(x) = x \cdot \min\{1, \frac{b}{||x||}\}$ to the correction term. Due to hubernization, corrections with high norm are multiplied with a value lower than 1, which weights down their contribution to the result. Robust constant, b, defines a threshold when the filter decreases correction if correction is the far from the model prediction. Note that when $b \to \infty$ the filter approaches the classical Kalman filter.

4.1. *Robust Kalman filter for tracking*

In every frame, we use the detector to locate the best position of the object. Then, we apply the filter and check if its result has significant overlap (more than 50%) with the detector's. If it has not, it is very likely that the object's location changed unexpectedly according to the motion model of the filter. In this case it is very likely that detector either: (i) made a mistake during detection (e.g., false negative) or (ii) the object was partially (of completely) occluded. Regardless of the reason, the current detection is likely to be incorrect and should not be used to update the tracker, thus for frames where overlap is small tracker is not updated.

To recover from the situation where the filter and the tracker disagree in the next frame, we calculate a discriminative function from bounding boxes sampled in the neighborhood of the filter's best prediction and detector's. Here we use the fact that filter is able to filter detection as an outlier and not change it's position by much. As a result if detector made a mistake it will be able to recover since the object will remain in the neighborhood of the filter's prediction. During partial or short-termed occlusions, it is possible that the detection can make few consecutive mistakes or is unreliable for use. Detection is considered unreliable if the overlap between best detection of the filter and the detector is lower then 50%. While this is happening we set the value of the robust constant is to half of the value

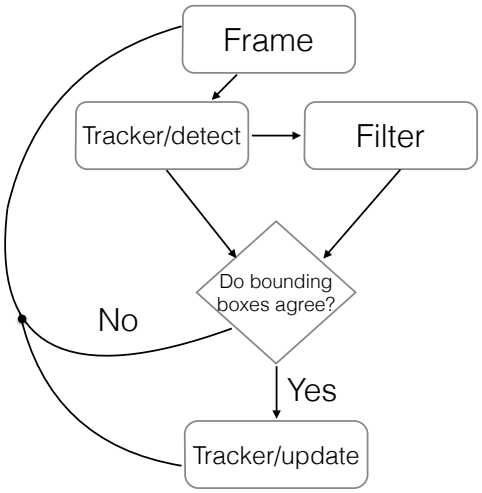

Fig. 1. Struck tracker with Robust Kalman filter.

originally set by the tracker. Once overlap is larger than 50% again the robust constant is set back to it's original value. While such a strategy is insufficient to address long-term occlusions, however, it performs well if occlusions are short-term. A diagram of our method is shown in Fig. 1.

5. Implementation

5.1. *Bounding box sampling*

Recall that structured SVM learns a function which maps bounding box to the real number value which determines it's compatibility with support vectors. To find a bounding box containing the object a set of bounding boxes has to be generated for it to choose from. As in,[5] we sample bounding boxes for tacker's search and update. Tracker's search is a step when given novel frame the tracker estimates the location of the object. During update the tracker uses the best bounding box it found from the search step and samples bounding boxes to add new positive and negative support vectors and update their gradients. Let $b = \{c_x, c_y, w_{\text{prev}}, h_{\text{prev}}\}$ be a bounding box parametrized so that c_x, c_y is the location of the center and w, h are dimensions. Given the bounding box of the object in the previous frame bounding boxes for search and update were sampled from the set in the form :

$$\begin{aligned} B(m,n,R,w,h) = \{b = (c_x + \rho\cos\phi, c_y + \rho\sin\phi, w, h) | \\ \phi \in [0, \tfrac{2\pi}{m}, ..., 2\pi], \\ \rho \in [0, \tfrac{R}{n}, ..., R]\}. \end{aligned} \tag{1.26}$$

Similarly bounding boxes are sampled for the search in the neighborhood of the filter if detector and the filter disagree.

5.2. *The Robust Kalman filter*

Our implementation uses a Kalman filter with constant-velocity model, thus unobserved state at frame t is given by $\mathbf{y}_t = [x_t^{tl}, y_t^{tl}, x_t^{br}, y_t^{br}, v_t^x, v_t^y]$, where $[x_t^{tl}, y_t^{tl}]$ and $[x_t^{br}, y_t^{br}]$ are pixel locations of the top left and bottom right corners of the bounding box and $[v_t^x, v_t^y]$ are velocities. Locations of two corners of the bounding box are observed, $\hat{\mathbf{y}}_t = [x_t^{tl}, y_t^{tl}, x_t^{br}, y_t^{br}]$. Constant velocity model assumes that location of the corners in time changes only because of velocity, i.e.,

$$x_t^{tl} = x_{t-1}^{tl} + v_{t-1}^x \tag{1.27}$$
$$y_t^{tl} = y_{t-1}^{tl} + v_{t-1}^y \tag{1.28}$$
$$x_t^{br} = x_{t-1}^{br} + v_{t-1}^x \tag{1.29}$$
$$y_t^{br} = y_{t-1}^{br} + v_{t-1}^y, \tag{1.30}$$

while velocity remains constant from frame to frame:

$$v_t^x = v_{t-1}^x \tag{1.31}$$
$$v_t^y = v_{t-1}^y. \tag{1.32}$$

In matrix notation matrices F and H are defined as follows:

$$F = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}, \tag{1.33}$$

$$H = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}. \tag{1.34}$$

Robust Kalman filter's covariance matrices were parametrized as follows:

$$Q = q \times \begin{pmatrix} \frac{1}{4} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{4} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{4} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 1 & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 1 \end{pmatrix}, \tag{1.35}$$

$$R = r \times E(4), \tag{1.36}$$

$$P = p \times E(6). \tag{1.37}$$

where $E(m)$ is an identity matrix of size $m \times m$ and p, q, r are parameters.

5.3. *Features*

Tracking framework presented here could be used with different set of features. To make it work a function extracting a feature vector from the bounding box has to be specified. Here is a list of features we consider here: resized raw image patches, histograms, Haar features, HoG features.

5.3.1. *Raw image patches*

Raw images patches are extracted by resizing a given bounding box to a fixed size (e.g., 64×64). Consequently the matrix is reshaped as a feature vector.

5.3.2. *Histograms*

Histograms are calculated by splitting a bounding box into squares of the same size in both directions. Let l be a number of squares, then there are $l \times l$ squares in total. For each such square a histogram is calculated and a result is concatenated into a feature vector.

If we vary parameter l, we can extract features on multiple levels and concatenate the result (e.g., $l = 1, 2, 3$). For n bins per histogram such feature has a size of $\sum_{l=1}^{L} l^2 n$. We used $n = 8$ and $L = 4$.

5.3.3. *Haar features*

Haar features are closely related to calculating derivatives. The benefit of Haar features is low computational cost as they could be calculated in constant time per bounding box by using integral images.

5.3.4. *HoG features*

Histograms of Oriented gradients are powerful features which achieved top results for different recognition tasks like pedestrian detection[16] and object recognition.[17] Features are computed by calculating orientation histograms in a set of image parts called cells.

5.4. *Fast Intersection Kernel*

By replacing $\langle \mathbf{\Phi}(\mathbf{x}, \mathbf{y}), \mathbf{\Phi}(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \rangle \rightarrow K(\mathbf{x}, \mathbf{y}, \bar{\mathbf{x}}, \bar{\mathbf{y}})$ we could use kernel or a combination of kernels to map features into higher dimensional space. One of kernel functions is an intersection kernel which, for vectors $\mathbf{z}, \bar{\mathbf{z}} \in \mathbf{R}^n$ is defined as:

$$K_{\text{int}}(\mathbf{z}, \bar{\mathbf{z}}) = \sum_{i=1}^{n} \min(z_i, \bar{z}_i) \tag{1.38}$$

To calculate discriminative function we need to compute the following:

$$F(\mathbf{x},\mathbf{y};\beta) = \sum_{(\mathbf{x},\mathbf{y})\in S} \beta^{\mathbf{y}}_{\mathbf{x}} \sum_{i=1}^{n} K(\mathbf{x},\mathbf{y},\bar{\mathbf{x}},\bar{\mathbf{y}}) = \tag{1.39}$$

$$= \sum_{(\mathbf{x},\mathbf{y})\in S} \beta^{\mathbf{y}}_{\mathbf{x}} \sum_{i=1}^{n} K_{\text{int}}(g(\mathbf{x}|_{\mathbf{y}}), g(\bar{\mathbf{x}}|_{\bar{\mathbf{y}}})) = \tag{1.40}$$

$$= \sum_{(\mathbf{x},\mathbf{y})\in S} \beta^{\mathbf{y}}_{\mathbf{x}} \sum_{i=1} \min(z_i(\mathbf{x},\mathbf{y}), \bar{z}_i) \tag{1.41}$$

Naive implementation of intersection kernel requires $O(n)$ operations per pair $\mathbf{x}, \bar{\mathbf{x}}$ while to compute discriminative function would require $O(|S|n)$. It can be reduced to $O(|S|\log n)$ if some preprocessing is performed.[18] Note that Eq. 1.41 could be rewritten as a sum of independent functions:

$$F(\mathbf{x},\mathbf{y};\beta) = \sum_{i=1}^{n} [\sum_{(\mathbf{x},\mathbf{y})\in S} \beta^{\mathbf{y}}_{\mathbf{x}} \min(z_i(\mathbf{x},\mathbf{y}), \bar{z}_i)] = \tag{1.42}$$

$$= \sum_{i=1}^{n} h(\bar{z}_i) \tag{1.43}$$

where we defined scalar function $h_i(s) = \sum_{(\mathbf{x},\mathbf{y})\in S} \beta^{\mathbf{y}}_{\mathbf{x}} \min(z_i(\mathbf{x},\mathbf{y}), \bar{z}_i)$. Let $\{\tilde{z}_i^j\}_{j=1}^{|S|}$ be sorted version of the sequence $\{z_i = g(\mathbf{x}|_{\mathbf{y}})_i\}_{(\mathbf{x},\mathbf{y})\in S}$ with corresponding coefficients $\{\tilde{\beta}^j\}_{j=1}^{|S|}$. Fix s, let r be the largest index so that $\tilde{z}_i^r \leq s$. Then we have

$$h_i(s) = \sum_{j=1}^{|S|} \tilde{\beta}^j \min(s, \tilde{z}_i^j) = \tag{1.44}$$

$$= \sum_{1\leq j\leq r} \tilde{\beta}^j \tilde{z}_i^j + s \sum_{r<j\leq|S|} \tilde{\beta}^j \tag{1.45}$$

$$= A_i(r) + sB_i(r) \tag{1.46}$$

where we defined $A_i(r) = \sum_{1\leq j\leq r} \tilde{\beta}^j \tilde{z}_i^j$ and $B_i(r) = s\sum_{r<j\leq|S|} \tilde{\beta}^j$. So if we have $A_i(r), B_i(r)$ precomputed we could find the index r using binary search which would allow to calculate kernel in total in $O(|S|\log n)$ operations. This is especially useful speed-up for tracking as the more bounding boxes structured tracker can evaluate the better are the tracking results.

6. Tracking evaluation

Due to specifics of the tracking it's evaluation is a hard problem in itself. Variety of the metrics and evaluation protocols were proposed[1,2,19,20] but the consensus is far from reached. Most of the metrics are highly correlated[2] and are borrowed from the problem of object detection. The problem with such metrics is when comparing trackers which drift in the

tracking process. In such scenario the frame number where the drifting occurred will be a deciding factor as the tracker which drifting later is very likely to have higher evaluation metrics.

6.1. *Metrics*

Two of the most common metrics used are *precision* and *success*. Success is a metric which measure the area of the overlap divided by the area of the union of the tracker's and ground truth bounding box. Let y, y_{gt} be tracker's and ground truth bounding box respectively then success, also known as union-over-overlap, is defined as

$$\Delta(y, y_{gt}) = \frac{|y \cap y_{gt}|}{|y \cup y_{gt}|} \tag{1.47}$$

Note that union-over-overlap is exactly the loss function used in the structured SVM section 3. Success rate is defined as the fraction of frames where overlap-over-union is less or equal than the threshold:

$$P(\gamma) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_{\Delta(y^i, y^i_{gt}) \leq \gamma} \tag{1.48}$$

where N is the total number of frames and $\mathbb{1}$ is an indicator function. For a single quantitative measurement an area under the curve in Eq. 1.48 is used.

Precision is another metric which measures the proximity of centers of the bounding boxes. Similarly as precision it is defined in terms of success rate, i.e. fraction of frames whose precision is within the threshold:

$$S(\gamma) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_{||c(y^i) - c(y^i_{gt})|| \leq \gamma} \tag{1.49}$$

where we used $c(y)$ as a function which calculates the center from the bounding box. It is easy to see that these two metrics are highly correlated as the tracker which has high success rate is likely to have high precision rate as well. An important metric which is different from precision and success is a *failure rate* which is the number of times the tracker failed during the tracking. The failure is occurs when the overlap between the tracker and the ground truth becomes zero. To calculate a failure rate evaluation protocol has to be ale to re-initialize the tracker after it failed as most trackers once failed will never recover by themselves.

6.2. *Evaluation protocols*

A standard evaluation of the tracker on the dataset, usually called *one-pass evaluation*, is simply to run it initializing it on the first frame in each video and to calculate precision and success rates. Such evaluation can be extended to calculate spatial and temporal robustness of the tracker.[1] In spatial robustness experiment (SRE), the initial bounding box perturbed with the noise: the center, each of the corners is shifted and the scale is changed. The

amount of shift is equal to 10% if the bounding box size. In temporal robustness experiment (TRE) the tracker is evaluated using different frames in the video and run in both directions: backward and forward. After a set of experiments (OPE, SRE, TRE) metrics like precision and success rates are calculated. A common critique of such approaches is that once the tracker fails the metric do not realistically depict tracker's accuracy as the earlier the tracker failed the more likely for it to have low success and precision rates. To account for this an evaluation protocol with re-initialization was proposed.[20] The protocol uses success and failure rate as the main two metrics but once the tracker fails it is immediately re-initialized. Re-initialization allows to get more information of the tracker per run. It, however, has a drawback.

Because the tracker is re-initialized the moment the overlap is zero it penalizes equally the trackers which have a potential to recover themselves compared to the ones which cannot. The tracker presented here is capable of recovering due to the robust kalman filter and is thus at disadvantage.

6.3. *Results*

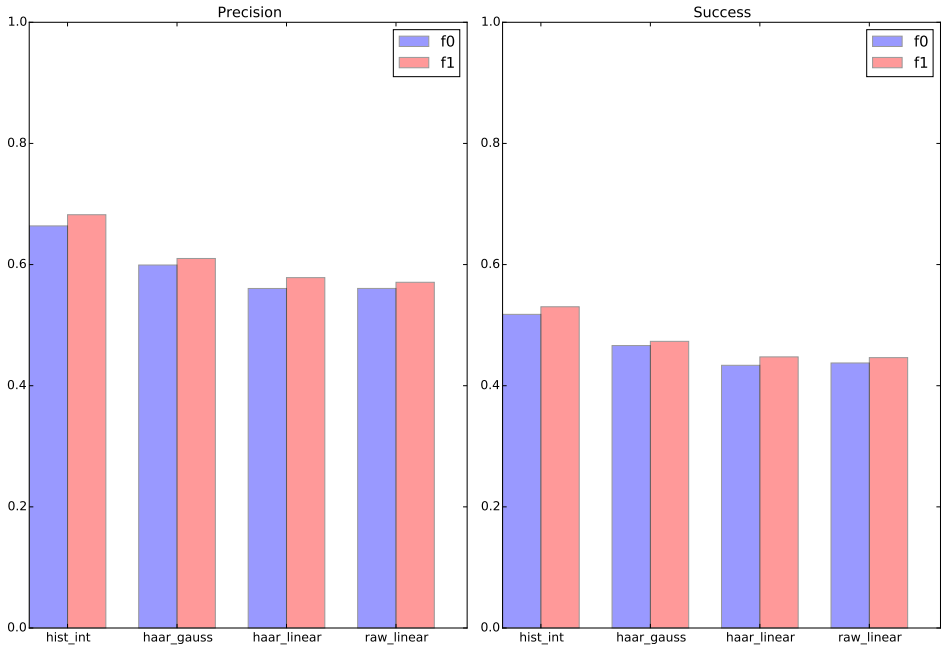

Fig. 2. Evaluation of the tracker with and without filter with different set of feature-kernel pairs on the dataset.[1]

To show the difference the Robust Kalaman filter makes we run evaluation on the[1] using all experiments (OPE, SRE, TRE). The dataset contains 51 different videos of variable length. For TRE experiments we initialized the tracker uniformly 20 times across the video.

In total there were 20 runs for TRE and 12 for SRE which were run on each of 51 videos. Results in Figure 2 was calculated by averaging across all runs.

Results in Figure 2 suggest that the filter improves the tracking independently of the features or kernels used.

7. Discussion and Conclusion

In this chapter we described a method for single-object visual tracking. The method uses tracking-by-detection paradigm and is based on structured SVM. To cope with computational requirements of the tracking we present a faster way to calculate intersection kernel which allows to decrease computational complexity from $O(|S|n)$ to $O(|S|\log n)$ where S is a set of support vectors and n is a dimension of the feature vector. To make the tracker resilient to false negatives we extend it with the Robust Kalman filter. The filter is used to detect and recover from incorrect detections and short time occlusions. Our results suggest that the filter improves tracking results independently of the features used in each set of experiments (OPE, SRE, TRE).

There are a number of possible extensions of this work. Once include extending the framework beyond bounding boxes to *rotated bounding boxes* which in addition to the location of center, dimension also have an angle. The extension could be done by replacing loss function, feature extraction and the sampling function with equivalents defined on the rotation bounding boxes. Another direction is to intelligently filter bounding boxes from the sampling step which are unlikely to contain an object.

The source code with the method is available online.[21] It contains the code necessary to run it on two benchmarks with evaluation protocols described in Sec. 6.2.

References

1. Y. Wu, J. Lim, and M.-H. Yang. Online object tracking: A benchmark. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pp. 2411–2418. IEEE, (2013).
2. A. Smeulders, D. Chu, R. Cucchiara, and S. Calderara, Visual tracking: An experimental survey, *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
3. W. Zhong, H. Lu, and M.-H. Yang. Robust object tracking via sparsity-based collaborative model. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 1838–1845. IEEE, (2012).
4. Z. Kalal, K. Mikolajczyk, and J. Matas, Tracking-learning-detection, *Pattern Analysis and Machine Intelligence, IEEE Transactions on*. **34**(7), 1409–1422, (2012).
5. S. Hare, A. Saffari, and P. H. Torr. Struck: Structured output tracking with kernels. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 263–270. IEEE, (2011).
6. P. Ruckdeschel, Optimally robust kalman filtering. (2010). URL `http://d-nb.info/102738966X/34`.
7. J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. Exploiting the circulant structure of tracking-by-detection with kernels. In *Computer Vision–ECCV 2012*, pp. 702–715. Springer, (2012).
8. B. Babenko, M.-H. Yang, and S. Belongie. Visual tracking with online multiple instance learn-

ing. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 983–990. IEEE, (2009).
9. I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the twenty-first international conference on Machine learning*, p. 104. ACM, (2004).
10. H. T. Nguyen and A. W. Smeulders, Fast occluded object tracking by a robust appearance filter, *Pattern Analysis and Machine Intelligence, IEEE Transactions on*. **26**(8), 1099–1104, (2004).
11. I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. In *Journal of Machine Learning Research*, pp. 1453–1484, (2005).
12. A. Bordes, L. Bottou, P. Gallinari, and J. Weston. Solving multiclass support vector machines with larank. In *Proceedings of the 24th international conference on Machine learning*, pp. 89–96. ACM, (2007).
13. J. Platt et al., Fast training of support vector machines using sequential minimal optimization, *Advances in kernel methodssupport vector learning*. **3**, (1999).
14. A. Bordes, N. Usunier, and L. Bottou. Sequence labelling svms trained in one pass. In *Machine Learning and Knowledge Discovery in Databases*, pp. 146–161. Springer, (2008).
15. Z. Wang, K. Crammer, and S. Vucetic. Multi-class pegasos on a budget. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 1143–1150, (2010).
16. N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 886–893. IEEE, (2005).
17. P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8. IEEE, (2008).
18. S. Maji, A. C. Berg, and J. Malik, Efficient classification for additive kernel svms, *Pattern Analysis and Machine Intelligence, IEEE Transactions on*. **35**(1), 66–77, (2013).
19. L. Čehovin, A. Leonardis, and M. Kristan, Visual object tracking performance measures revisited, *arXiv preprint arXiv:1502.05803*. (2015).
20. M. Kristan, J. Matas, A. Leonardis, T. Vojir, R. Pflugfelder, G. Fernandez, G. Nebehay, F. Porikli, and L. Cehovin, A novel performance evaluation methodology for single-target trackers, *arXiv preprint arXiv:1503.01313*. (2015).
21. I. Bogun, Antrack. (2015). doi: 10.5281/zenodo.18171. URL `http://dx.doi.org/10.5281/zenodo.18171`.