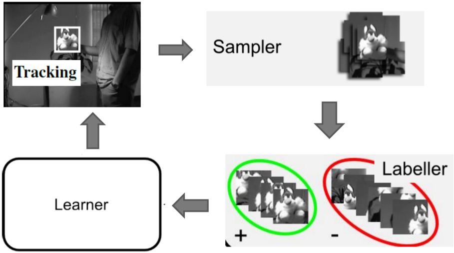
Online Single Object Tracking

Elena Burceanu
elena.burceanu@gmail.com

Tracking

- Tracking
  - specific classes: pedestrians, cars, etc (integrate prior knowledge)
  - generic (any objects, no special treatment)
- update (adaptive)
  - accommodate with the changes in obj appearance
  - keep the model learned so far
- some challenges
  - changes in appearance: lightning, (fast, complex) motion, occlusions
  - drifting: accumulating small errors (eg. bkg as train)
  - decide bbox based on detection map
  - labeler: artificial binarization step (similarity = bbox IoU)

Tracking by Detection - framework



- sampler and labeler
  - chooses patches to update on (near previous detection)
  - ex. label = threshold on the distance from the max activation
- learner (appearance model)
  - binary classifier (foreground vs background)
  - outputs the activation map for the target on each frame
  - trains with samples based on previous frame detection
- tracker
  - use the learner (detection) results to choose the next object location
  - choose the maximum activation zone

Tracking by Detection - Formal

- taxonomy
  - I_t - image at frame t
  - p_t - (predicted) target configuration in frame t (eg. bbox, + scale, + rotation)
  - y_t - transformations on current frame wrt prev frame (translation, scale, rotation)
  - $x_{t+1}^{y(p_t)}$ - features for patch in I_{t+1} transformed (y) around p_t
  - scoring function $g : \chi \mapsto \mathbb{R}$
- update
  - sample transformations (near current detection p_t): bboxes
  - extract features from bboxes and label them
  - update g
- propagation
  - detect near previous position and choose maximum activation
  - choose the transformation (y_t) that maximizes g score
  - $p_{t+1} = y_t(p_t)$

Trackers categories

- dictionary based trackers
  - sparse combinations of elements from dict
  - keep long and short term dict
  - dict for different aspects of the target
- ensemble based trackers
  - combine result of multiple weak classifiers
- segmentation based trackers
  - keep a segmentation model to better identify background in bbox
- Next in presentation:
  - structured learning (STRUCK)
  - circulant matrices trackers (KCF)
- others: oriented bbox

Structured Output Tracking with Kernels

Structured Output Tracking with Kernels, Sam Hare, Stuart Golodetz, Amir Saffari, Vibhav Vineet, Ming-Ming Cheng, Stephen L. Hicks and Philip H. S. Torr (PAMI 2015)

- online structured output SVM
  - allow the output space (structured) to express the needs of the tracking
  - remove the intermediate step of producing binary samples for the classifier update
  - the learner is directly connected to the tracker (predict the transformation between 2 frames)
- bugeting (limit the number of support vectors)

Structured output tracking

- generalize SVM for general output (not only for binary and multiclass classification and regression)
- the scoring function (g) has direct access to y (the transformation)
- SVM (arbitrary input, binary output):
  - $f(x|w) = sign(\langle w, \Phi(x) \rangle)$
  - $g(x, y|w) = y\langle w, \Phi(x) \rangle = \langle w, \Phi(x)y \rangle$
  - $\hat{y}_i = f(x_i|w) = argmax_{y \in \{-1,1\}} g(x_i, y|w)$
- structured output SVM: (arbitrary input and output)
  - $g(x, y|w) = \langle w, \Phi(x, y) \rangle$
  - $\hat{y}_i = f(x_i|w) = argmax_{y \in Y} g(x_i, y|w)$

Structured SVM

- Primal SMV
  - $J(w) = \frac{1}{2}||w||^2 + C\sum_1^n \xi_i \ (min_w)$
  - s.t. $\forall i : \xi_i \geq 0$
  - s.t. $\forall i, \forall y \neq y_i : \langle w, \Phi(t_i, y_i) - \Phi(t_i, y)\rangle \geq \Delta(y_i, y) - \xi_i$
  - (Equivalent: $\xi_i \geq \Delta(y_i, y) - [g(x_i, y_i|w) - g(x_i, y|w)]$
  - $\Delta(y_i, y) = 1 - s_p^o(y_i, y)$ (s_p^o - the overlap function IoU)
  - ensure that $g(t_i, y_i)$ is grater than $g(t_i, y)$ by a margin given by the symmetric loss function $\Delta(y_i, y)$ (different from the SVM threshold binarization)
- Dual SVM Formulation (and β reparametrization)
  - $J(\beta) = -\sum_{i,y} \Delta(y, y_i)\beta_i^y - \frac{1}{2}\sum_{i,j,y,\tilde{y}} \beta_i^y \beta_j^{\tilde{y}} k(t_i, y, t_j, \tilde{y}) \ (max_\beta)$
  - s.t. $\forall i, \forall y : \beta_i^y \leq \delta(y, y_i)C$
  - s.t. $\sum_y \beta_i^y = 0$
  - scoring: $g(t, y) = \sum_{i,\tilde{y}} \beta_i^{\tilde{y}} k(t_i, \tilde{y}, t, y)$
- $(t_i, y_i) : y_i =$ the correct transformation of the object from p_{t_i} in p_{t_i+1}
- if $\beta_i^{\tilde{y}} \neq 0$, $(t_i, \tilde{y})$ - support vectors, t_i - support pattern
- $\beta_i^{y_i} > 0$; $\beta_i^{\tilde{y}} < 0, \tilde{y} \neq y_i$ (one positive, the rest are negative)

Update SVM

- online optimization: LaRank (based on Sequential Minimal Optimization + decompose in small sub-programs, solvable analytically)
- coordinate ascent in SMO (update only 2 parameters, keep the rest fixed)

- $\overline{\beta}_m^{y_+} = \beta_m^{y_+} + \lambda^u$
- $\overline{\beta}_m^{y_-} = \beta_m^{y_-} - \lambda^u$
- $\frac{\partial J(\overline{\beta})}{\partial \lambda^u} = 0$
- find λ^u (unconstrained) and truncate to keep constraints
- $\nabla_m^y = \frac{\partial J}{\partial \beta_m^y}$
- **Q:** how to choose y_+, y_-?

Algorithm 1 SMOStep

Require: m, y_+, y_-, $\mathcal{S}$, β, ∇, C

$k_{(++)} = k(t_m, y_+, t_m, y_+)$
$k_{(--)} = k(t_m, y_-, t_m, y_-)$
$k_{(+-)} = k(t_m, y_+, t_m, y_-)$
$\lambda^u = \frac{\nabla_m^{y_+} - \nabla_m^{y_-}}{k_{(++)} + k_{(--)} - 2k_{(+-)}}$
$\lambda = \max(0, \min(\lambda^u, C\delta(y_+, y_m) - \beta_m^{y_+}))$
Update coefficients
$\beta_m^{y_+} \leftarrow \beta_m^{y_+} + \lambda$
$\beta_m^{y_-} \leftarrow \beta_m^{y_-} - \lambda$
Update gradients
for $(t_i, y) \in \mathcal{S}$ **do**
  $k_{(+)} = k(t_i, y, t_m, y_+)$
  $k_{(-)} = k(t_i, y, t_m, y_-)$
  $\nabla_i^y \leftarrow \nabla_i^y + \lambda \left(k_{(-)} - k_{(+)}\right)$
end for

Update SVM

- online optimization: LaRank (based on Sequential Minimal Optimization + decompose in small sub-programs, solvable analytically)
- coordinate ascent in SMO (update only 2 parameters, keep the rest fixed)

- $\overline{\beta}_m^{y_+} = \beta_m^{y_+} + \lambda^u$
- $\overline{\beta}_m^{y_-} = \beta_m^{y_-} - \lambda^u$
- $\frac{\partial J(\overline{\beta})}{\partial \lambda^u} = 0$
- find λ^u (unconstrained) and truncate to keep constraints
- $\nabla_m^y = \frac{\partial J}{\partial \beta_m^y}$
- **Q:** how to choose y_+, y_-?

- highest gradient ($argmax_y \nabla_m^y$)

Algorithm 1 SMOSTEP

Require: m, y_+, y_-, $\mathcal{S}$, β, ∇, C

$k_{(++)} = k(t_m, y_+, t_m, y_+)$
$k_{(--)} = k(t_m, y_-, t_m, y_-)$
$k_{(+-)} = k(t_m, y_+, t_m, y_-)$
$\lambda^u = \frac{\nabla_m^{y_+} - \nabla_m^{y_-}}{k_{(++)} + k_{(--)} - 2k_{(+-)}}$
$\lambda = \max(0, \min(\lambda^u, C\delta(y_+, y_m) - \beta_m^{y_+}))$
Update coefficients
$\beta_m^{y_+} \leftarrow \beta_m^{y_+} + \lambda$
$\beta_m^{y_-} \leftarrow \beta_m^{y_-} - \lambda$
Update gradients
for $(t_i, y) \in \mathcal{S}$ **do**
  $k_{(+)} = k(t_i, y, t_m, y_+)$
  $k_{(-)} = k(t_i, y, t_m, y_-)$
  $\nabla_i^y \leftarrow \nabla_i^y + \lambda\left(k_{(-)} - k_{(+)}\right)$
end for

Update steps

- ProcessNew
  - add the entry for the true label (t_m, y_m) as a positive SV
  - search for the most important sample to become a negative SV
  - new example (t_m, y_m), init: $\beta_m^y = 0$
  - $y_+ = y_m$, $y_- = argmin_{y \in Y} \nabla_m^y$ (iterate over all transformations)
  - $SMO(m, y_+, y_-)$

Update steps

- ProcessNew
  - add the entry for the true label (t_m, y_m) as a positive SV
  - search for the most important sample to become a negative SV
  - new example (t_m, y_m), init: $\beta_m^y = 0$
  - $y_+ = y_m$, $y_- = argmin_{y \in Y} \nabla_m^y$ (iterate over all transformations)
  - $SMO(m, y_+, y_-)$
- ProcessOld
  - revisiting a frame and potentially add new negative SV (and adjust β)
  - random choose m (such that $\beta_m^{y_m} < C$; we want to be able to update β)
  - $y_+ = y_m$, $y_- = argmin_{y \in Y} \nabla_m^y$
  - $SMO(m, y_+, y_-)$

Update steps

- ProcessNew
  - add the entry for the true label (t_m, y_m) as a positive SV
  - search for the most important sample to become a negative SV
  - new example (t_m, y_m), init: $\beta_m^y = 0$
  - $y_+ = y_m$, $y_- = argmin_{y \in Y} \nabla_m^y$ (iterate over all transformations)
  - $SMO(m, y_+, y_-)$
- ProcessOld
  - revisiting a frame and potentially add new negative SV (and adjust β)
  - random choose m (such that $\beta_m^{y_m} < C$; we want to be able to update β)
  - $y_+ = y_m$, $y_- = argmin_{y \in Y} \nabla_m^y$
  - $SMO(m, y_+, y_-)$
- Optimize
  - random choose m
  - only modifies β of existing SV ($y_+ = y_m$, $y_- = argmin_{y \in Y_m} \nabla_m^y$)
  - $SMO(m, y_+, y_-)$

Tracking loop

- Bugeting
  - curse of kernelisation (storage space and eval time)
  - remove the support vector that results in the smallest change to the weight vector w (and update one more parameter s.t. $\sum_y \beta_i^y = 0$)
  - $\Delta w = -\beta_r^y \Phi(t_r, y) + \beta_r^y \Phi(t_r, y_r)$ **Q:** Solution?!

Tracking loop

- Bugeting
  - curse of kernelisation (storage space and eval time)
  - remove the support vector that results in the smallest change to the weight vector w (and update one more parameter s.t. $\sum_y \beta_i^y = 0$)
  - $\Delta w = -\beta_r^y \Phi(t_r, y) + \beta_r^y \Phi(t_r, y_r)$ **Q:** Solution?! minimize $||\Delta w||^2$
- ProcessNew, ProcessOld: might add SVs
- $n_O = n_R = 10$
- for all SVs $(t_i, y) \in S$, they store actualized: β_i^y, ∇_i^y
- if $\beta_i^y = 0$, remove from S
- sample y from a grid (not all 2D transformations Y)

Algorithm 2 Struck tracking loop.

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

Propagate the estimated object configuration
$y_t = f(t)$
$\mathbf{p}_{t+1} = y_t(\mathbf{p}_t)$
Update the SVM
ProcessNew(t, y_t)
MaintainBudget$()$
for $i = 1$ to n_R **do**
  ProcessOld$()$
  MaintainBudget$()$
  for $j = 1$ to n_O **do**
    Optimize$()$
  end for
end for
return $\mathbf{p}_{t+1}, \mathcal{S}_{t+1}$

Practical Considerations

- Search spaces
  - y: 2D translation, + scale; r = 30 px around previous point
  - scale only with 5 % difference from previous frame
  - 81 transformations (5x16 grid in 60 px)
- Kernels
  - linear $k(t, y, \overline{t}, \overline{y}) = \langle x_{t+1}^{y(p_t)}, x_{\overline{t}+1}^{\overline{y}(p_{\overline{t}})} \rangle$
  - gaussian $k(t, y, \overline{t}, \overline{y}) = \exp(-\sigma || x_{t+1}^{y(p_t)} - x_{\overline{t}+1}^{\overline{y}(p_{\overline{t}})} ||_2^2)$
  - intersection $\frac{1}{2} \sum_{j=1}^{D} min(x_{t+1}^{y(p_t)}[j], x_{\overline{t}+1}^{\overline{y}(p_{\overline{t}})}[j])$, D - feature vector size
- Features
  - raw 16x16 gray scale: 256D
  - Haar (6 types, 2 scales, 4x4 grid): 192D
  - histogram (4 levels pyramid, LxL size per level, 16 features): 480D
- kernel[i] + features[i] = best (**Q:** Why?)
- Multiple Kernel Learning (average more kernels for 1 result) outperforms KCF (but very slow)

Benchmark

- Wu dataset, otb50
- evaluate on categories: fast motion, occlusions, scale changes, etc
- metrics
  - precision = location error (% of frames whose predicted bboxes were within a threshold of gt bbox) (20 px)
  - success = overlap (% of frames whose IoU (predicted, gt bbox) $>$ threshold (AuC)
- Robustness
  - perturb the initialization in time and space
  - OnePassEval: first frame gt bbox
  - TemporalRobustnessEval: other starting frame
  - SpatialRobustnessEval: shifts + scales on initial bbox

Results

Tracker	Variant	Features	Kernel	Budget	Success TRE	Success SRE	Precision TRE	Precision SRE
Struck	fkbRL100	Raw	Linear	100	0.471	0.400	0.651	0.569
Struck	fkbRL25	Raw	Linear	25	0.446	0.377	0.611	0.529
Struck	fkbHG100	Haar	Gaussian	100	*0.504*	0.434	0.706	0.628
Struck	fkbHG25	Haar	Gaussian	25	0.479	0.406	0.665	0.579
ThunderStruck	fkbRL100	Raw	Linear	100	0.459	0.384	0.633	0.546
ThunderStruck	fkbRL25	Raw	Linear	25	0.367	0.308	0.494	0.421
ThunderStruck	fkbHG100	Haar	Gaussian	100	0.490	0.417	*0.681*	*0.602*
ThunderStruck	fkbHG25	Haar	Gaussian	25	0.410	0.350	0.562	0.479
Baseline	–	Haar	Gaussian	100	0.473	0.401	0.656	0.567
ASLA	–	–	–	–	0.485	*0.421*	0.620	0.577
SCM	–	–	–	–	0.513	0.420	0.652	0.575
TLD	–	–	–	–	0.448	0.402	0.624	0.573
KCF	–	–	–	–	**0.556**	**0.463**	**0.774**	**0.683**

TABLE 1: The tracking performance of single-scale, single-kernel Struck and ThunderStruck variants on the Wu *et al.* [21] benchmark using various feature/kernel/budget combinations. We used search radii of 30 pixels for propagation and 60 pixels for learning, and set n_R and n_O, the numbers of reprocessing and optimisation steps used for LaRank, to 10.

- why KCF is better?
  - computational efficiency (HOG vs Haar)
  - dense sampling (vs grid) - they've invalidated this assumption with tests
- structured learning
  - compare with a SVM with binary learner (overlap $<$ 0.5 for negatives and one positive)

Multi-kernel Results

Tracker	Variant	Features/Kernels	Feature Count	Success TRE	Success SRE	Precision TRE	Precision SRE
Struck	mklHGRL	Haar/Gaussian + Raw/Linear	448	0.476	0.401	0.660	0.575
Struck	mklHGHI	Haar/Gaussian + Histogram/Intersection	672	0.545	**0.469**	**0.785**	**0.707**
Struck	mklHIRL	Histogram/Intersection + Raw/Linear	736	0.494	0.418	0.690	0.606
Struck	mklHGHIRL	Haar/Gaussian + Histogram/Intersection + Raw/Linear	928	0.495	0.422	0.692	0.610
Struck	fkbHG100	Haar/Gaussian	192	0.504	0.434	0.706	0.628
Struck	fkbHI100	Histogram/Intersection	480	*0.517*	*0.455*	*0.734*	*0.679*
Struck	fkbRL100	Raw/Linear	256	0.471	0.400	0.651	0.569
KCF	–	–	–	**0.556**	0.463	0.774	0.683

TABLE 2: Comparing the tracking performance of some single-kernel variants of Struck with variants that use multiple kernel learning (MKL). For all variants, we use a learning search radius r_L of 60 pixels, a propagation search radius r_P of 30 pixels 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. We show the results of the KCF tracker for comparison purposes.

- multi-kernel (mklHGHI) - very slow, not on CUDA, outperforms KCF

Qualitative Results

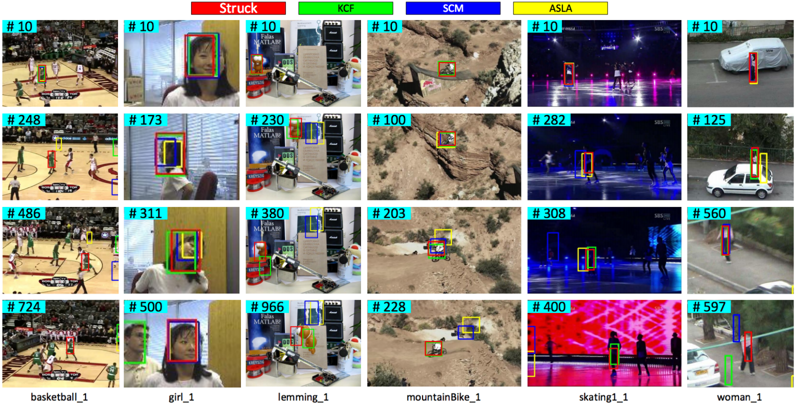


Fig. 6: Example frames from benchmark sequences, comparing the results of Struck (variant mklHGHI) with KCF [22], SCM [49] and ASLA [51]. Videos of these results can be found at https://goo.gl/cJ1Dg7.

Conclusions

Tracker	Variant	Average FPS
Struck	fkbRL100	20.9
Struck	fkbHG100	20.8
Struck	mklHGHI	2.4
ThunderStruck	fkbRL100	**146.3**
ThunderStruck	fkbHG100	*93.9*
ThunderStruck	ro5_5	125.1
ThunderStruck	sHG95_105_1	19.9

TABLE 3: Comparing the average speed (in frames per second) of a number of variants of Struck and ThunderStruck over the entire Wu benchmark. For details of the parameters used and the tracking performance obtained for each variant, see the corresponding experiments sections.

- fewer steps in LaRank, more scales (11)
- Conclusions
  - structured output prediction
  - budget maintenance
  - cuda implementations
  - (not anymore) state of the art
  - best: large feature vectors + multi-kernel tracking

KCF

High-Speed Tracking with Kernelized Correlation Filters, Joo F. Henriques, Rui Caseiro, Pedro Martins, Jorge Batista (PAMI 2015)

- in the context of the discriminative classifier (target vs surrounding)
- augment dataset with translated + scaled patches (redundancies: overlap)
- use a circulant data matrix, which is diagonal in Discrete Fourier Transform space
- very fast: from $O(D^3)$ to $O(Dlog(D))$

Circulant matrices and Fourier

$$X = C(\mathbf{x}) = \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_n \\ x_n & x_1 & x_2 & \cdots & x_{n-1} \\ x_{n-1} & x_n & x_1 & \cdots & x_{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_2 & x_3 & x_4 & \cdots & x_1 \end{bmatrix}$$

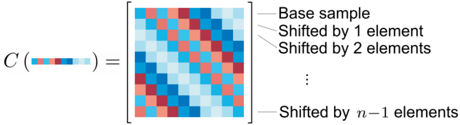


Figure 3: Illustration of a circulant matrix. The rows are cyclic shifts of a vector image, or its translations in 1D. The same properties carry over to circulant matrices containing 2D images.

- $Px = [x_n, x_1, x_2, \ldots, x_{n-1}]^T$
- $\{P^u x | u = \overline{0, n-1}\}$ - set of all shifts
- all circulant matrices are made diagonal by the Discrete Fourier Transform (DFT)
- $X = F diag(\hat{x}) F^H$ (circular matrix eigen decomposition)
- $\hat{x} = DFT(x) = \sqrt{n} F x$, F is constant

Cyclic shifts

$$P = \begin{bmatrix} 0 & 0 & 0 & \dots & 1 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix}$$

Figure 2: Examples of vertical cyclic shifts of a base sample. Our Fourier domain formulation allows us to train a tracker with *all* possible cyclic shifts of a base sample, both vertical and horizontal, without iterating them explicitly. Artifacts from the wrapped-around edges can be seen (top of the left-most image), but are mitigated by the cosine window and padding.

		Storage	Bottleneck	Speed
Random Sampling (p random subwindows)		Features from p subwindows	Learning algorithm (Struct. SVM [4], Boost [3, 6]...)	10 - 25 FPS
Dense Sampling (all subwindows, proposed method)		Features from one image	Fast Fourier Transform	320 FPS

Train in DFT (Linear Ridge Regression)

- classical
  - $J(w) = \sum_i (w^H x_i - y_i)^2 + \lambda ||w||_2^2$
  - $J(w) = ||X^H W - Y||_2^2 + \lambda ||w||_2^2$
  - $min_w J \rightarrow \frac{\partial J}{\partial w} = 0$
  - $w = (X^H X + \lambda I_D)^{-1} X^H y$, $O(D^3 + D^2 N)$ complexity
- interesting
  - matrix inversion lemma:
    $(P^{-1} + B^T R^{-1} B)^{-1} B^T R^{-1} = PB^T (BPB^T + R)^{-1}$
  - $R = I_N$, $B = X$, $P = \lambda^{-1} I_D$
  - $w = X^H (XX^H + \lambda I_N)^{-1} y$, $O(N^3 + N^2 D)$ complexity
- X - circulant
- $w = (F * diag(\hat{x} \bigodot \hat{x}^*) F^H + \lambda I)^{-1} F diag(\hat{x}^*) F^H y$
- $\hat{w} = \frac{\hat{x}^* \bigodot \hat{y}}{\hat{x} \bigodot \hat{x}^* + \lambda}$

Train in DFT (Kernelized Ridge Regression) I

- starting from $w = X^H(XX^H + \lambda I_N)^{-1}y$
- $XX^T = K$
- $\alpha = (K + \lambda I_N)^{-1}y$
- $w = X^T\alpha = \sum_i \alpha_i x_i$
- $f(x) = w^T x = \sum_i \alpha_i x_i^T x = \sum_i \alpha_i k(x_i, x) = (K^x)^T\alpha$
- Kernelized Ridge Regression: $\alpha = (K + \lambda I)^{-1}y$

Train in DFT (Kernelized Ridge Regression) II

- $\alpha = (K + \lambda I)^{-1} y$
- Theorem: Kernel matrix K of a circular matrix C(x) is circulant if $\forall$ unitary matrix M ($MM^T = I$), $k(x, x') = k(Mx, Mx')$
  - $k(x_i, x_j) = k(P^i x, P^j x) = k(MP^i x, MP^j x)$
  - $M = P^{-i} \rightarrow k_{i,j} = k(x, P^{(j-i) mod n} x) \rightarrow K$ is circulant
  - k^{xx} - first row of the circular K (generating vector)
- K is circulant $\rightarrow \hat{\alpha} = \frac{\hat{y}}{\hat{k}^{xx} + \lambda}$

Detection in DFT

- Detection
  - more general definition: $k_i^{xx'} = k(x', P^{i-1}x)$ - kernel correlation
  - $K^z = C(k^{xz})$, k^{xz} - kernel correlation between image x and patch z; circulant in base vectors permutations (x, z)
  - $f(z) = (K^z)^T * \alpha$
  - $\hat{f}(z) = \hat{k}^{xz} * \hat{\alpha}$, a vector containing the output for all cyclic shifts of z
- Kernel correlation
  - dot product (polynomial kernel)
    - $k_i^{xx'} = k(x', P^{i-1}x) = g(x'^T P^{i-1}x) \rightarrow k^{xx'} = k(C(x)x')$
    - $k^{xx'} = g(F^{-1}(\hat{x}^* \bigodot \hat{x}'))$
  - RBF and Gaussian kernel
    - $k_i^{xx'} = k(x', P^{i-1}x) = h(||x'^T - P^{i-1}x||^2)$
    - $k_i^{xx'} = k(x', P^{i-1}x) = h(||x'^T||^2 + ||P^{i-1}x||^2 - 2x'^T P^{i-1}x)$
    - $k^{xx'} = h(||x'^T||^2 + ||x||^2 - 2F^{-1}(\hat{x}^* \bigodot \hat{x}'))$
    - $k^{xx'} = \exp(-\frac{1}{\sigma^2}(||x'^T||^2 + ||x||^2 - 2F^{-1}(\hat{x}^* \bigodot \hat{x}')))$
- Multiple Channels:
  $k^{xx'} = \exp(-\frac{1}{\sigma^2}(||x'^T||^2 + ||x||^2 - 2F^{-1}(\sum_c \hat{x}_c^* \bigodot \hat{x}_c')))$

Algorithm

Inputs
- x: training image patch, $m \times n \times c$
- y: regression target, Gaussian-shaped, $m \times n$
- z: test image patch, $m \times n \times c$

Output
- responses: detection score for each location, $m \times n$

```
function alphaf = train(x, y, sigma, lambda)
  k = kernel_correlation(x, x, sigma);
  alphaf = fft2(y) ./ (fft2(k) + lambda);
end

function responses = detect(alphaf, x, z, sigma)
  k = kernel_correlation(z, x, sigma);
  responses = real(ifft2(alphaf .* fft2(k)));
end

function k = kernel_correlation(x1, x2, sigma)
  c = ifft2(sum(conj(fft2(x1)) .* fft2(x2), 3));
  d = x1(:)'*x1(:) + x2(:)'*x2(:) - 2 * c;
  k = exp(-1 / sigma^2 * abs(d) / numel(d));
end
```

- train a new model at the new position
- linearly interpolate the obtained values of α and x with the ones from the previous frame

Results

	Algorithm	Feature	Mean precision (20 px)	Mean FPS
Proposed	KCF	HOG	**73.2%**	172
	DCF		**72.8%**	**292**
	KCF	Raw pixels	56.0%	154
	DCF		45.1%	278
Other algorithms	Struck [7]		65.6%	20
	TLD [4]		60.8%	28
	MOSSE [9]		43.1%	**615**
	MIL [5]		47.5%	38
	ORIA [14]		45.7%	9
	CT [3]		40.6%	64

Table 1: Summary of experimental results on the 50 videos dataset. The reported quantities are averaged over all videos. Reported speeds include feature computation (e.g. HOG).

Results by category

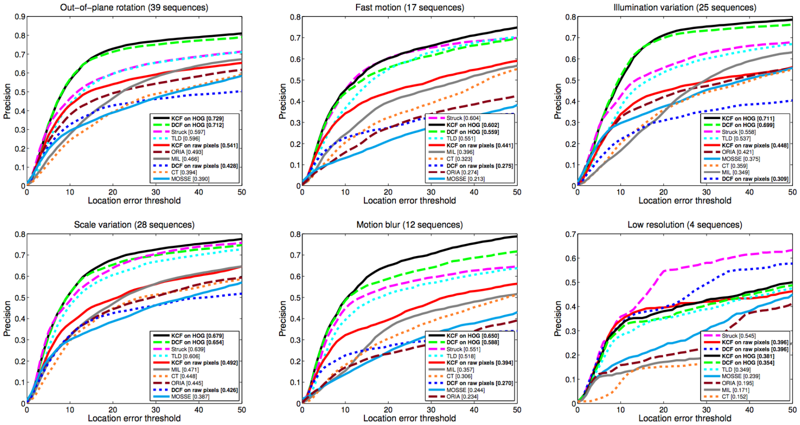
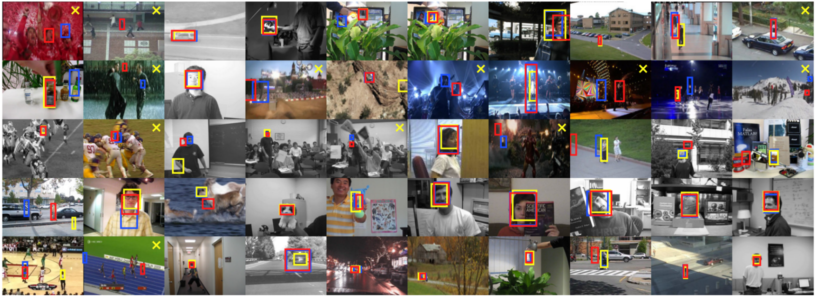


Figure 7: Precision plots for 6 attributes of the dataset. Best viewed in color. In-plane rotation was left out due to space constraints. Its results are virtually identical to those for out-of-plane rotation (above), since they share almost the same set of sequences.

Qualitative Results



Kernelized Correlation Filter (proposed) TLD Struck

Figure 1: Qualitative results for the proposed Kernelized Correlation Filter (KCF), compared with the top-performing Struck and TLD. Best viewed on a high-resolution screen. The chosen kernel is Gaussian, on HOG features. These snapshots were taken at the midpoints of the 50 videos of a recent benchmark [11]. Missing trackers are denoted by an "x". KCF outperforms both Struck and TLD, despite its minimal implementation and running at 172 FPS (see Algorithm 1, and Table 1).

Other Questions?