Scalable Kernel Correlation Filter with Sparse Feature Integration

Andrés Solís Montero Jochen Lang Robert Laganière
School of Electrical Engineering and Computer Science
University of Ottawa
asolismon@uottawa.ca {jlang,laganiere}@eecs.uottawa.ca

Abstract

Correlation filters for long-term visual object tracking have recently seen great interest. Although they present competitive performance results, there is still a need for improving their tracking capabilities. In this paper, we present a fast scalable solution based on the Kernalized Correlation Filter (KCF) framework. We introduce an adjustable Gaussian window function and a keypoint-based model for scale estimation to deal with the fixed size limitation in the Kernelized Correlation Filter. Furthermore, we integrate the fast HoG descriptors and Intel's Complex Conjugate Symmetric (CCS) packed format to boost the achievable frame rates. We test our solution using the Visual Tracker Benchmark and the VOT Challenge datasets. We evaluate our tracker in terms of precision and success rate, accuracy, robustness and speed. The empirical evaluations demonstrate clear improvements by the proposed tracker over the KCF algorithm while ranking among the top state-of-the-art trackers.

1. Introduction

Visual tracking is one of the fundamental research areas in computer vision. Tracking has several applications, such as video compression [6], augmented reality [23], traffic control [12], surveillance and security [21]. Although some settings allow for strong assumptions about the target [19, 10, 26] , most often tracking an object with no prior knowledge is desirable. Model-free tracking requires online learning and adaption of a representation for a given target. The tracker analyzes sequential video frames and after the initial detection of a target, outputs the successive positions of the object from frame to frame. Even though satisfactory progress has been made in model-free trackers, fast and robust model-free tracking is still a challenging problem due to geometric transformations, changes in illumination, fast motions, noise, occlusions and background clutter.

A considerable amount of work has been dedicated to monocular single-target visual tracking. Among the relevant work are model-free tracking solutions that use sparse object representations [14, 20, 24], histogram-based representations [2, 1, 11, 9] and combinations of them [5, 14, 13]. Performances measures have also been proposed and different benchmark datasets to evaluate visual trackers are available [28, 15, 4].

Recently, the tracking-by-detection paradigm has obtained excellent performance for their robust tracking of targets. [2, 9, 1, 20, 11, 13]. This tracking framework integrates an online learning component that acquires new information from each successful target detection. Recent correlation filter-based trackers [3, 1, 8, 11] replace the time-consuming convolution operation in the time domain by pointwise multiplication in the frequency domain. Henriques et al. [1, 11] extended correlation filter by exploiting the circulant structure of the tracking object to quickly incorporate information from all cyclically shifted samples into the Fourier analysis without iterating over all possible samples in the target's neighborhood in the time domain. Furthermore, Henriques et al. incorporated multiple feature channels instead of raw pixel into their correlation filter framework to improve the accuracy and robustness of the tracker [11]. However, the correlation filter tracker uses a fixed size template and hence the tracker is not able to handle scale changes of a target occurring during motion.

The main contributions of this paper can be summarized as follows. First, we propose a correlation filter that overcomes the fixed-size limitation of the KCF tracker and in addition is able to run faster than the original KCF tracker. We extend the KCF tracker with the capability of handling scale changes by introducing adjustable Gaussian window functions for better back- and foreground separation around the target leading to increased accuracy and robustness. We solve the scale estimation by combining the correlation filter with sparse keypoints to estimate location and scale. The window filtering does not alter the pipeline of the correlation filters presented in Henriques et al. [11], i.e., Kernelized Correlation Filter using a Gaussian or polynomial kernel, and Dual Correlation Filter (DCF) using a linear kernel. This allows including scale estimation to any variation

of the original correlation filters. Second, we achieve up to 2x speedup of the original solution while reducing the memory footprint by half by introducing CCS packed format and fast HoG descriptors. Third, we conducted experiments to compare the original KCF algorithm against our solution using the original paper dataset [11]. These experiments demonstrate the gain in performance in terms of success rate and precision rate. Finally, the empirical experimental evaluations are extended with the VOT challenge for a more accurate representation of the performance gain. Our proposed solution achieves performance gains in accuracy, robustness, and speed compared to state-of-the-art trackers.

2. Related Work

Numerous tracking-by-detection algorithms can be found in the current literature. Many of them have been evaluated with the Visual Tracker Benchmark [28] and in the VOT Challenges [15]. We restrict our review here to those trackers that are close to our work, including TLD [13], CMT [20], Struck [9], SCM [30], Alien [24], CSK [1], KCF [11] and SAMF [17].

Struck [9] is a tracking-by-detection algorithm, which relies on a kernelized structured output Support Vector Machine (SVM) in order to distinguish between tracked object and background. The binary classifier employs a Haar-like vector representation of the target to solve the classification problem. SCM [30] combines a binary classifier with a generative model to achieve high accuracy and robustness. TLD [13] combines forwards-backwards tracking of grid points with a sampling detection strategy using a boosted classifier to predict the target location and scale. Location and scale estimation are selected as the median value transformation between all pairs of successfully tracked points. CMT [20] uses a similar tracking strategy to TLD to estimate position, scale and also includes rotation. Instead of equally spaced points a keypoint-based matching strategy using BRISK [16] features and descriptor is implemented. Alien extends the keypoint-based strategy by creating two sets of keypoints, i.e, object and context. It adopts SIFT [18] features and descriptors to match keypoints and uses single value decomposition to estimate position, scale and orientation of the matches.

All the trackers mentioned in the last paragraph achieve good results in the Visual Tracker Benchmark [28]. However, their computational cost is high which makes them less appealing than the correlation filter-based trackers. In addition, the correlation filter-based trackers CSK and KCF outperform all of the above mentioned trackers in the Visual Tracker Benchmark and VOT challenges while being faster.

CSK explores the structure of the circulant patch, which employs kernel correlation filter to achieve high performance. Based on CSK, KCF adds multichannel features to the correlation filter pipeline. It improves accuracy and robustness by adopting HoG features instead of the raw pixel values used in CSK.

SAMF is the closest work to our solution because of its goal. It adds scale to the KCF framework by sampling the original target with different scales and learning the model at each scale. Moreover, it combines HoG descriptor with colour-naming [27] technique to boost the overall performance. However, the inclusion of colour-naming to the HoG features and computing the kernelized correlation filter for all possible scale comes at a substantial computational cost.

Our proposed algorithm is based on the KCF algorithm. The main difference between our proposed solution and SAMF is that we learn the scale variations online instead of learning the model for all possible scales. Furthermore, instead of adding computation time, we are improving the high-speed performance of KCF while boosting its accuracy and robustness. The Gaussian window filtering allows us to adjust to scale changes while creating a better separation of the object and the background without affecting the KCF framework. Because of the separate Gaussian filtering, scale estimation can be added to any of the polynomial and linear correlation filters presented in Henriques et al. [11].In addition, our C++ implementation incorporates fast HoG descriptors using SSE instructions and CCS packed format to improve the performance while decreasing the memory footprint. For scale estimation, we use a similar approach as in TLD and combine keypoint tracking with detection. We implement the forward-backward Kanade-Lucas-Tomasi [29] keypoint tracking strategy initialized at the position estimated by KCF. However, we select good features to track instead of grid-points. Also, we include a different measure to estimate the scale. While TLD uses a median value to estimate scale we use a weighted arithmetic mean where points near the center of the tracker contribute more than those near the boundaries.

3. Scalable Kernelized Correlation Filter

In this section, we first review the KCF [11] algorithm (See Alg. 1). Second, we discuss the introduction of Gaussian window filtering to allow the correlation tracker react to scale changes. Furthermore, we introduce the scale estimation used in our approach and how to integrate it into the KCF pipeline (See Alg. 2). Finally, we describe our additions to increase the performance of the original approach.

3.1. KCF Algorithm

The key innovation of KCF is the use of the structure of circulant matrices to enhance the discriminative ability of the track-by-detection scheme. The algorithm proceedes as follows: Given the initial selection of a target (i.e., center position and size), a tracked region is created. The tracked region is increased from the target size to provide some con-

text. Features (either raw pixels or other feature channels) are extracted from the tracked region and each channel is weighted by a cosine window. A circulant matrix is used to learn all the possible shifts of the target from a base sample. The coefficient α_f encodes the training samples, consisting of all shifts of a base sample in the frequency domain. The fast learning equation is expressed as

$$\alpha_f = \frac{y_f}{k_f^{xx'} + \lambda} \tag{1}$$

where $y_f = Fy$ denotes the Discrete Fourier Transform(DFT) of y. The term $k_f^{xx'}$ denotes the DFT of $k^{xx'}$; the kernel correlation function between signals x and x'. The division represent a element-wise division and the scalar λ is a regularization term. The training label matrix y is a Gaussian function that smoothly decays from the value of one for the centered target to zero for other shifts.

For multiple channel features, the vector x concatenates the individual vectors of c channels, e.g., the 31 gradient orientation bins for the HoG descriptor [11] as $x = [x_1, x_2, ..., x_c]$. The Gaussian kernel correlation function $k^{xx'}$ is defined as

$$k^{xx'} = exp\left(-\frac{1}{\sigma^2}\left(\|x\|^2 + \|x'\|^2 - 2F^{-1}\left(\sum_c x_f^c \odot (x_f'^c)^*\right)\right)\right) \tag{2}$$

where $\odot$ represent element-wise multiplication and $(x_f^c)^*$ the complex-conjugate of x_f^c (see lines 9, and 5).

The displacement $\delta = \underset{loc}{\text{argmax}}(r(k_f^{z\tilde{x}}))$ between the current and the next patch z is the spatial index with maximum response in $r(k_f^{z\tilde{x}})$. The response is computed as the element-wise multiplication between the learnt alphas $\tilde{\alpha}_f$ and the correlation of z_f with the learnt model $\tilde{x}_f$. The detection response for each location is

$$r(k_f^{z\tilde{x}}) = F^{-1}(k_f^{z\tilde{x}} \odot \tilde{\alpha}_f) \tag{3}$$

The learnt model $\tilde{x}_f$ and alpha $\tilde{\alpha}_f$ are linear interpolations of x_f and α_f at each detection with the selection of an interpolation factor $factor \in [0, 1]$.

A detailed description of KCF algorithm pipeline in pseudocode follows. For the more detailed formulation, please refer to [11].

Algorithm 1 : KCF.

Variables with subscript f are in the frequency domain. Circled operators represent element-wise operations (i.e., $\odot$ and $\oslash$).
- w_sz: size of the tracked region, (W×H).
- pos: center location of the tracker in spatial domain.
- $patch$: region of img centered at pos with size w_sz, (W×H×C).
- $features(x)$: extracted features (e.g., HoG), (m×n×c).
- cos_window: cosine window weights each feature channel, (m×n×1).

1: **for each** img **in** $sequence$:
2: **if not** first image:
3: $patch \leftarrow$ region(img, pos, w_sz)
4: $z_f \leftarrow F$(features($patch$) $\odot$ cos_window)
5: $k_f^{z\tilde{x}} \leftarrow F$(correlation($z_f$, $\tilde{x}_f$)) ▷ Eq. (2)
6: $pos \leftarrow pos +$ argmax$_{loc}(r(k_f^{z\tilde{x}}))$ ▷ Eq. (3)
7: $patch \leftarrow$ region(img, pos, w_sz)
8: $x_f \leftarrow F$(features($patch$) $\odot$ cos_window)
9: $k_f^{xx} \leftarrow F$(correlation(x_f, x_f)) ▷ Eq. (2)
10: $\alpha_f \leftarrow y_f \oslash (k_f^{xx} + \lambda)$ ▷ Eq. (1)
11: **if** first image: $f \leftarrow 1$ **else** $f \leftarrow factor$
12: $\tilde{\alpha}_f \leftarrow f \times \alpha_f +$ (1 - f) $\times \tilde{\alpha}_f$
13: $\tilde{x}_f \leftarrow f \times x_f +$ (1 - f) $\times \tilde{x}_f$

3.2. Adjustable Window Filtering

In image processing, the process of multiplying an image with a smoothly ending function that gradually reduces its values near the boundaries is called "windowing". The multiplication creates a windowed view of the input signal where the overlap is the signal of interest and reducing to zero the rest. The purpose of windowing is usual to isolate the signal of interest while reducing the frequency leakage in the DFT calculations. Separating the signal from the background is of vital importance for detection. Frequency leakage takes place when the frequency spectrum of a measured input has other frequencies than those of the original signal. Adjustable windows are functions that are capable of reducing the frequency leakage while controlling the bandwidth. They allow controlling how much information of the original signal we want to analyze. Gaussian windows have been widely used for window filtering because of their simple formulation and their benefits in reducing leakage and adjusting to different signal size [25, 22].

In KCF, the target size is strongly linked to the tracked region (i.e., the search area) because of the cosine window. Target scale changes will alter the signal to process and affect the learning process. If the target gets smaller, the cosine window will merge the target signal with the background. If the target gets bigger it discards information of the target and only processes a subset of it. Another issue linked to this behaviour is that we cannot change the tracked region (e.g., we would like to increase/decrease the search area) given the target size.

To overcome this limitation in KCF we replaced their cosine window with a Gaussian window G (See Eq. 4) to allow for target changes of scale and a better separation from the background (See Fig. 3.2). The Gaussian window allows us to control the bandwidth of the distribution while the cosine function is fixed to the region size. Furthermore, the Fourier transform of a Gaussian is also a Gaussian which ensure the separation between foreground and the background while reducing the frequency leakage. Scaling a cosine window has no such property.

$$G(m, n, \sigma_w, \sigma_h) = g(m, \sigma_w) * g(n, \sigma_h)' \quad (4)$$

The function $g(N, \sigma)$ returns a vector of size N computed as follow

$$g(N, \sigma) = \exp\left(-\frac{1}{2}\left(\frac{i}{\sigma(N-1)}\right)^2\right), \quad 0 \leq i \leq N. \quad (5)$$

The resulting two-dimensional Gaussian window is a matrix $gauss$ that has size $m \times n$. Note that the $W \times H \times C$ tracked region, is filtered in feature space, $(m \times n \times c)$. For example, when using raw pixels as features their values match, i.e, $m = W, n = H$, and $c = C$. In the case of HoG features with a cell size of 4×4 and 9 orientation bins, their correspondent values are $m = W/4, n = H/4$, and $c = 31$.

The bandwidth σ of the Gaussian function $g(N, \sigma)$ is computed independently for the horizontal and vertical orientations. The values of σ are selected as the ratio between the feature dimensions and the target dimensions for the horizontal and vertical orientations (i.e., $\sigma_w = \frac{m}{w}$ and $\sigma_h = \frac{n}{h}$). The replaced cosine window from KCF is computed as $cos\_window = h(m) * h(n)'$; $h(x)$ is the Hann windowing function

$$h(N) = \frac{1}{2}\left(1 - cos\left(2\pi\frac{i}{N}\right)\right), \quad 0 \leq i \leq N. \quad (6)$$

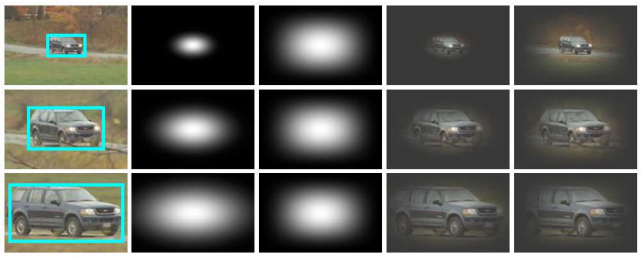

Figure 1. Gaussian and cosine window filtering raw pixel value example. The tracked region has a size of (175×113). First column: same regions with targets at three different scales (i.e., small (56×32), medium (110×61), large (164×83). Second column: Gaussian windows according the target size (i.e., small $(\sigma_w = .32, \sigma_h = .28)$, medium $(\sigma_w = .63, \sigma_h = .54)$, large $(\sigma_w = .94, \sigma_h = .73)$. Third column: cosine window for all the three examples (i.e, size (175×113)). Fourth and fifth columns: images filtered with Gaussian and cosine windows respectively. Figure shows how the fixed cosine window fails to represent the target compared to the Gaussian windows. The cosine window includes background for small targets and discards information for big targets.

3.3. Scale Estimation

With the adjustable window function the algorithm is capable of adjusting to changes in scale, we need to define how to estimate the new dimension of the target at each frame. The change of scale will be used to update the target size, which in turn, will update the σ values for the Gaussian window, the regression labels and the learnt model of the correlation filter. For the estimation, we implemented a keypoint-based strategy where the most interesting points inside the target area are extracted $K^{p1} = K_1^{p1}, K_2^{p1}, ..., K_N^{p1}$ and tracked to the next region (i.e., $K^{p2} = K_1^{p2}, K_2^{p2}, ..., K_N^{p2}$). Pairs of tracked points are used to estimate the change of scale (i.e., $pairs = (K_1^{p1}, K_1^{p2}), (K_2^{p1}, K_2^{p2}), ..., (K_T^{p1}, K_T^{p2})$, with $T \leq N$). As in the TLD tracker, we use a forwards-backwards optical flow strategy where we keep the tracked points with high confidence value [13]. The differences between TLD and our approach are that we extract relevant keypoints instead of grid points and estimate the scale using a different metric. TLD uses the median value of all scale ratios between matched points to estimate the scale variation and pairs of points contribute equally to the final estimation. When TLD selects the grid-points inside the target area, they assume that the points belongs to the object. During the tracking process this assumption is generally not correct. The rectangular representation of the target might not follow exactly the geometry of the target. To account for this fact, we assign different weights to the keypoints. We assume that points near the center of the target will be more likely to be part of the object (i.e., they should have greater weight in the scale estimation) than those near the boundaries (i.e., weight lower).

An extracted keypoint K_i^{p1} and its tracked position K_i^{p2} will have an associated weight w_i corresponding to the response of the extracted keypoint K_i^{p1} in a Gaussian window centered at the target area. The horizontal and vertical bandwidths of the Gaussian function are computed as in the previous section; relative to the target size. The variation of scale is computed as the weighted arithmetic mean of the ratio between all possible pairs of extracted and tracked points between next patch $p2$ and the current one $p1$.

$$scale(K^{p1}, K^{p2}) = \frac{\sum_i^T \sum_j^T w_i w_j * \frac{\left\|K_i^{p2} - K_j^{p2}\right\|^2}{\left\|K_i^{p1} - K_j^{p1}\right\|^2}}{\sum_i^T \sum_j^T w_i w_j}. \quad (7)$$

where i and j are the indices of successfully tracked points T. The weights w_i and w_j are the responses of the extracted points.

The proposed sKCF algorithm can be seen in Alg. 2. Note that the scale estimation step is independent and does not affect the correlation filter. This allow us to select a different kernel correlation function (i.e., polynomial or linear) or a different scale estimation strategy without affecting the sKCF pipeline.

Algorithm 2 : sKCF.

Changes to the KCF pipeline are showed in different color.
- w_sz: size of the tracked region, (W×H).
- t_sz: size of the target, (w×h).
- $features(x)$: extracted features (e.g., HoG), (m×n×c).

for each img **in** $sequence$:
  if not first image:
    $p2 \leftarrow$ region(img, pos, w_sz)
    $z_f \leftarrow F$(features($p2$) $\odot$ cos_window)
    $k_f^{z\tilde{x}} \leftarrow F$(correlation($z_f$, $\tilde{x}_f$)) ▷ Eq. (2)
    $pos \leftarrow pos$ + $\underset{loc}{\text{argmax}}(r(k_f^{z\tilde{x}}))$ ▷ Eq. (3)
    $t\_sz \leftarrow t\_sz$ * scale(K^{p1}, K^{p2}) ▷ Eq. (7)
  $p1 \leftarrow$ region(img, pos, w_sz)
  $gauss \leftarrow$ G(m, n, t_sz, w_sz) ▷ Eq. (4, 5)
  $x_f \leftarrow F$(features($p1$) $\odot$ $gauss$)
  $k_f^{xx} \leftarrow F$(correlation(x_f, x_f)) ▷ Eq. (2)
  $\alpha_f \leftarrow y_f \oslash (k_f^{xx} + \lambda)$ ▷ Eq. (1)
  if first image: $f \leftarrow 1$ **else** $f \leftarrow factor$
  $\tilde{\alpha}_f \leftarrow f \times \alpha_f$ + (1 - f) × $\tilde{\alpha}_f$
  $\tilde{x}_f \leftarrow f \times x_f$ + (1 - f) × $\tilde{x}_f$

3.4. Improving Algorithm Run-time

The overall complexity of the correlation filter is determined by the DFT/IDFT calls. As the algorithm only requires element-wise operations for the fast learning and the detection, the computational cost is $O(n \log n)$ where n is the number of pixels in the tracked region. Other parts of the code that can affect the performance of the algorithm are the scale estimation and the feature extraction, i.e., the use of raw pixels values demands no extra computation in the pipeline, but stronger and robust representations such as HoG descriptors can affect overall performance. Therefore, we include the following modifications to the original KCF algorithm to improve its run-time.

When performing the spectral analysis of an image, the original data is usually padded to get a bigger image that can be transformed much faster then the original. Images whose size is a power of two are the fastest to process. Nevertheless, data whose size can be represented as $S = 2^p \times 3^q \times 5^r$, for some integer values of p, q, r are also processed quite efficiently. Unfortunately, selecting the optimal input size for DFT/IDFT might be not desirable because the exponential growth of 2^p. For example, the spectral analysis of an image with size (300×300) could be efficiently computed (i.e., $300 = 5^2 \times 3 \times 2^2$) while using the optimal power of two selection, will end up computing the spectrum in an image of almost double its original size (i.e., 512×512). We utilize OpenCV $getOptimalDFTSize$ to compute the minimum number S that is greater or equal to the original dimensions. We are able to take advantage of this fact because of our Gaussian window function where the filtered data is not affected by the size of the tracked region unlike the original KCF. The bandwidth of the Gaussian distribution is controlled by the target size. This small modification can not be adapted directly to the KCF pipeline because of the cosine window used in KCF; increasing the search area will fail to correctly filter the target data.

Intel's CCS packed format exploits the symmetrical properties of the Fourier spectrum to efficiently encode the full spectrum of the frequency analysis. The complete spectrum is usually encoded into two matrices of floating precision with the same dimensions of the input data (i.e., for the real and imaginary part). CCS format interlaces the first half of the real and the imaginary spectrum data into one matrix of floating precision. This can be achieved because of its symmetrical representation. Introducing the CCS format into the DFT/IDFT calls effectively reduces the computation time and memory footprint by half. OpenCV only provides CCS data manipulation for their DFT/IDFT calls. We extended the OpenCV functionality to implement the element-wise division of the learning Formula 1 using the CCS packed format.

For the feature extraction, we introduce the fast HoG descriptor by Felzenszwalb et al. [7] in their work on discriminatively trained deformable part models. The implementation uses SIMD (SSE) to perform same operations on multiple data points simultaneously, exploiting data level parallelism for the HoG descriptor. This implementation gives nearly identical results to the original HoG descriptors while speeding up the algorithm 4x [7]. The algorithm is capable of computing a (640×480) image in less than $24ms$ runtime on a Core i5 (2415M). The original algorithm was implemented for the Matlab environment and process data in column-major data format. OpenCV uses a row-major data format so we adapted the algorithm to be used in the correct data format.

Finally, in scale estimation the LucasKanade [29] optical flow for a sparse set of features dominantes run-time. To accelerate this process we use initial estimations by introducing the location results from the correlation filter to the tracked points. Also, we only need to compute the optical flow in the tracked region instead of the complete image frame. We drop the pyramidal approach under the assumption of small displacements. The overall complexity is then expressed as $O(kn)$, where k is the number of extracted features and n the size of the processed images. The number of keypoints k is considerable lower than n which in turn is bound by the tracked region (i.e., search area).

4. Experiments

We conduct the evaluation of the proposed algorithm in three experiments. First, we compare the speed and accuracy performance using the 50 videos from the Visual

Tracker Benchmark [28] as in the KCF paper [11]. Second, we evaluate and rank the trackers using the VOT Challenge [15]. Finally, we compare our tracker against state-of-the-art trackers to show the effectiveness of sKCF. We report the detailed evaluation on the VOT 2015 dataset.

4.1. Experimental Setup and Methodology

We developed three C++ implementations to compare against the original KCF algorithm (which is available as a Matlab implementation only). First, we implemented the KCF algorithm using Gaussian correlation plus integrating Gaussian windowing and fast HoG descriptors. This implementation uses the full Fourier spectrum for the DFT/IDFT calls (i.e., no CCS). We call it KCF$_c$. Second, we integrated CSS packed format to the previous implementation (i.e., KCF$_{ccs}$). Finally, we integrated scale estimation to the previous variation (i.e., sKCF). All experiments are conducted on an Intel i5 (2415M) at 2.3GHz with 8GB of memory. We compared the three implementations to the original Matlab KCF implementation using HoG descriptors. We use the same parameter configurations for all implementations as described in [11]: interpolation $factor = .02$, Gaussian kernel correlation $\sigma = .5$, regulation term $\lambda = 1e-4$, a HoG cell size of 4×4 and 9 orientations bins.

For the Visual Tracker Benchmark [28] dataset, two evaluation criteria are used (i.e., precision and success). Precision is the percentage of frames in a sequence the tracker output is under a certain location error value. The location error is defined as the distance between the center of tracked results and the ground truth. The overall precision is defined as the mean precision for all location errors; the higher the more accurate the result. Success is the percentage of frames the tracker output is under a certain overlap ratio. The overlap ratio is defined as $VOC = \frac{Area(B_T \cap B_G)}{Area(B_T \cup B_G)}$, where B_T is the tracked bounding box, and B_G is the ground truth bounding box. The overlap ratio values go from 0 to 1. The overall success is defined as the mean success for all overlap ratio values.

In the VOT Challenge, two evaluation criteria are used (i.e., accuracy and robustness). Accuracy is measured as the VOR score. The robustness indicates the number of failures to track an object in a sequence. A failure is determined when the VOR score goes to zero.

4.2. Experiment 1: Visual Tracker Benchmark

We evaluated the performance of our three implementations (i.e., KCF$_c$, KCF$_{ccs}$, and sKCF) against KCF using the Visual Tracker Benchmark [28] as in the KCF paper [11]. We computed the overall precision and success rates for the 51 video dataset presented by Wu et al. [28].

The introduction of the Gaussian window in KCF$_c$ and KCF$_{ccs}$ resulted in increased precision and success rates over the KCF tracker (see Figure 4.2). KCF$_c$ and KCF$_{ccs}$ displayed similar performance validating the idea that the CCS packed format does not affect the accuracy while speeding the computations. The sKCF tracker produces better results than all the trackers by introducing the scale estimation into the pipeline. Furthermore, sKCF displayed an average speed-up of 2.2x for all the sequences compared to the KCF implementation. For this experiment, we included the performances of the top three ranked trackers in Wu et al. [28] (i.e., SCM, Struck, and TLD) for completeness.

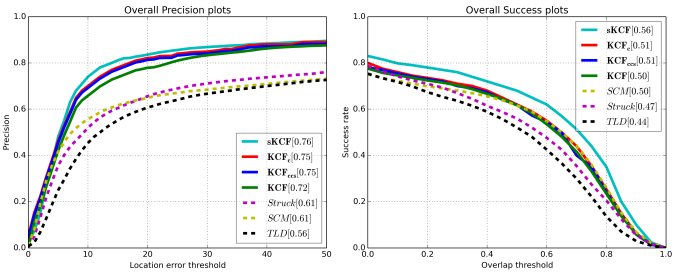

Figure 2. Precision and success plots for our three implementations (i.e., sKCF, KCF$_c$, KCF$_{ccs}$), KCF, and the top three ranked trackers in the Visual Tracker Benchmark [28] dataset.

4.3. Experiment 2: VOT Challenge

We continue to evaluate the trackers in terms of accuracy and robustness. For this experiment we used the VOT 2015 and VOT TIR 2015 datasets (i.e., 60 and 20 video sequences respectively).

Along with sKCF, KCF$_c$, KCF$_{ccs}$ and KCF, we included the NCC (Fast Normalized Cross-Correlation) implementation provided in the VOT packages for evaluation purposes. Our three implementations produce superior results than KCF and NCC trackers in the overall ranking output for both challenges. In terms of speed, NCC was the faster tracker (i.e., higher frame per seconds) followed by KCF$_{ccs}$ and sKCF. The proposed sKCF algorithm showed the higher overall accuracy values (i.e., mean accuracy value from all the sequences) and a lower failure rate for the VOT 2015 dataset. In the VOT TIR dataset, NCC displayed the best accuracy values from all the implementations but with higher failure rate which translated into the lowest overall rank from all the implementations. KCF$_c$ and KCF$_{ccs}$ had the same positioning above the KCF tracker but at different speeds.

Table 1 shows the results obtained from the five trackers in the VOT and TIR 2015 datasets. The tables display the overall average accuracy and average number of failures in all the sequences, with their accuracy, robustness and overall ranks. The tables also display the average tracker speed given in frames per seconds (i.e., fps).

Table 1. VOT and VOT TIR 2015 Results

VOT 2015	Overall		Rank			
	Acc.	Fail.	Acc.	Rob.	Overall	fps
sKCF	**0.50**	**2.49**	**2.22**	**2.60**	**2.41**	64.5
KCF_c	0.49	2.58	3.19	2.64	2.92	49.8
KCF_{ccs}	0.49	2.58	3.19	2.64	2.92	71.2
KCF	0.47	2.61	3.29	2.68	2.99	24.4
NCC	0.48	11.34	3.18	4.43	3.81	**78.5**
VOT TIR 2015						
sKCF	0.58	**5.28**	2.92	**2.50**	**2.71**	215.0
KCF_c	0.57	5.40	3.32	2.52	2.92	180.8
KCF_{ccs}	0.57	5.40	3.32	2.52	2.92	216.3
KCF	0.56	5.66	3.40	2.65	3.02	94.8
NCC	**0.65**	9.52	**1.98**	4.80	3.39	**262.3**

Finally, we evaluate our proposed sKCF tracker on the VOT 2014 dataset (i.e., 25 sequences). Table 2 summarizes the top three ranked trackers. Our algorithm is the fastest of the top trackers with competitive high accuracy and low failure rates. We achieved similar results to the two variations of the original KCF that incorporate scale support (i.e., SAMF and KCF*). The KCF* tracker improves the original KCF by adding a multi-scale support, sub-cell peak estimation and replacing the model update scheme. We couldn't find details of this implementation nor publication with details of the improvements. The KCF* entry on the VOT 2014 submission only refers to the original KCF paper [11]. The last entry in Table 2 shows the KCF Matlab's code [11] results on the VOT 2014 challenge.

Table 2. VOT 2014 Results

	Overall		Rank			
	Acc.	Fail.	Acc.	Rob.	Overall	fps
DSST	0.65	**16.90**	5.44	**12.17**	**8.81**	5.8
SAMF	0.65	19.23	5.23	12.94	9.09	1.6
KCF*	**0.66**	19.79	**5.16**	13.55	9.36	24.2
sKCF	0.61	18.44	7.68	13.14	10.41	**65.4**
KCF	0.56	27.14	13.14	18.02	15.58	20.3

5. Conclusion

We presented a very effective tracker based on the Kernel Correlation Filter. We introduced a fast scale scheme to improve on the KCF limitation of fixed template size. Moreover, the Gaussian window creates a better separation of the target and the background improving accuracy. Fast HoG descriptors, variable template size selection, and the CCS packed format improved the overall run-time. The modifications to the KCF pipeline allows the use of different kernel correlations as in the original paper [11]. The scale estimation is independent of the framework and a different approach could be adopted with our approach. The empirical evaluations on the Visual Tracker Benchmark and VOT datasets demonstrate the validity of our algorithm. Our proposed solution ranked among the top state-of-the-art trackers.

6. Acknowledgment

This research was partly funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) and Thales, Canada.

References

[1] J. ao F. Henriques, R. Caseiro, P. Martins, and J. Batista. Exploiting the circulant structure of tracking-by-detection with kernels. In *Proceedings of the European Conference on Computer Vision - Volume Part IV*, ECCV'12, pages 702–715, Berlin, Heidelberg, 2012. Springer-Verlag.

[2] B. Babenko, M.-H. Yang, and S. Belongie. Visual tracking with online multiple instance learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, August 2011.

[3] V. N. Boddeti, T. Kanade, and B. V. K. V. Kumar. Correlation filters for object alignment. In *Proceedings of CVPR*, pages 2291–2298. IEEE, 2013.

[4] L. Cehovin, A. Leonardis, and M. Kristan. Visual object tracking performance measures revisited. *CoRR*, abs/1502.05803, 2015.

[5] T. B. Dinh, N. Vo, and G. G. Medioni. Context tracker: Exploring supporters and distracters in unconstrained environments. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 1177–1184, 2011.

[6] L. Dong, I. Zoghlami, and S. Schwartz. Object tracking in compressed video with confidence measures. In *IEEE International Conference on Multimedia and Expo*, pages 753–756, July 2006.

[7] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

[8] H. K. Galoogahi, T. Sim, and S. Lucey. Multi-channel correlation filters. In *IEEE International Conference on Computer Vision, ICCV 2013, Sydney, Australia, December 1-8, 2013*, pages 3072–3079, 2013.

[9] S. Hare, A. Saffari, and P. H. S. Torr. Struck: Structured output tracking with kernels. In *IEEE International Conference on Computer Vision*, pages 263–270, Nov 2011.

[10] J. F. Henriques, R. Caseiro, and J. Batista. Globally optimal solution to multi-object tracking with merged measurements. In D. N. Metaxas, L. Quan, A. Sanfeliu, and L. J. V. Gool, editors, *ICCV*, pages 2470–2477. IEEE, 2011.

[11] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 2015.

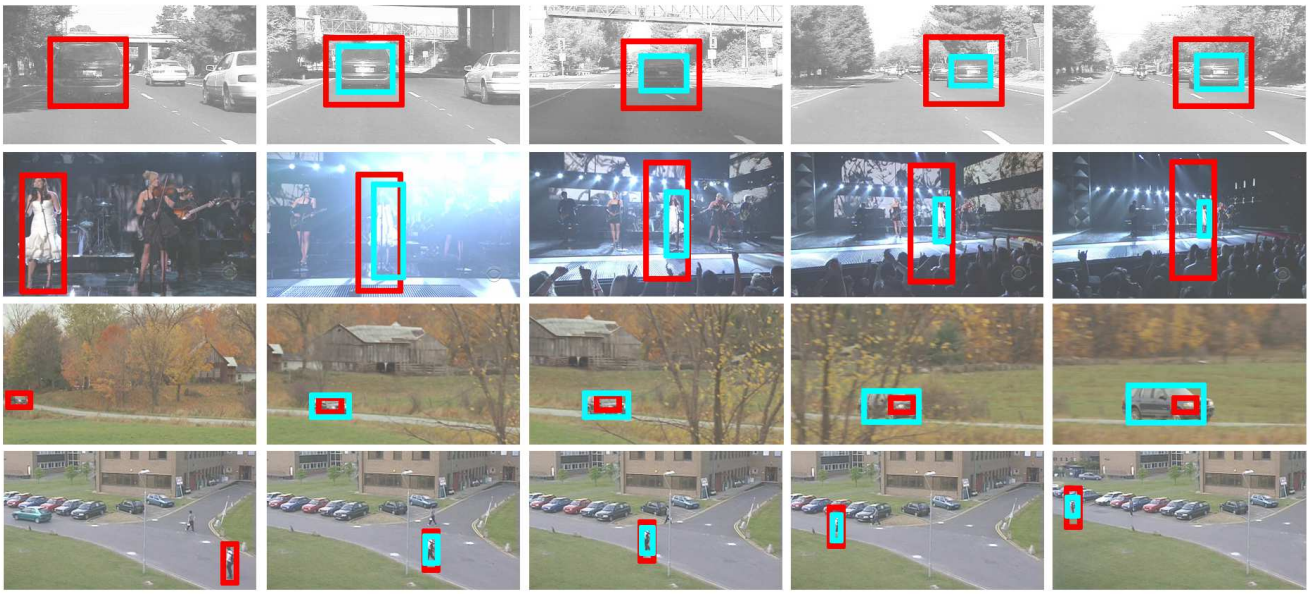

Figure 3. Some sequence examples with scale changes and the responses from KCF (i.e., red boxes) and sKCF (i.e., light blue boxes). The first column represents the initial selection and the size of the tracker at t he first frame in the sequence.

[12] I. Hwang, H. Balakrishnan, K. Roy, and C. Tomlin. Multiple-target tracking and identity management in clutter for air traffic control. In *Proceedings of the AACC American Control Conference*, 2004.

[13] Z. Kalal, K. Mikolajczyk, and J. Matas. P-n learning: Bootstrapping binary classifiers by structural constraints. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 49–56, June 2010.

[14] Z. Kalal, K. Mikolajczyk, and J. Matas. Tracking-learning-detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1409–1422, 2012.

[15] M. Kristan, J. Matas, A. Leonardis, T. Vojir, R. P. Pflugfelder, G. Fernández, G. Nebehay, F. Porikli, and L. Cehovin. A novel performance evaluation methodology for single-target trackers. *CoRR*, abs/1503.01313, 2015.

[16] S. Leutenegger, M. Chli, and R. Y. Siegwart. Brisk: Binary robust invariant scalable keypoints. *IEEE International Conference on Computer Vision*, 0:2548–2555, 2011.

[17] Y. Li and J. Zhu. A scale adaptive kernel correlation filter tracker with feature integration. In *European Conference on Computer Vision*, volume 8926 of *Lecture Notes in Computer Science*, pages 254–265. Springer Publishing, 2015.

[18] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov. 2004.

[19] A. S. Montero, J. Lang, and R. Laganière. A general framework for fast 3d object detection and localization using an uncalibrated camera. In *2015 IEEE Winter Conference on Applications of Computer Vision, WACV 2014, Waikoloa, HI, USA, January 5-9, 2015*, pages 884–891, 2015.

[20] G. Nebehay and R. Pflugfelder. Consensus-based matching and tracking of keypoints for object tracking. In *Winter Conference on Applications of Computer Vision*. IEEE, Mar. 2014.

[21] J. Omar and M. Shah. Tracking and object classification for automated surveillance. In *Proceedings of the European Conference on Computer Vision-Part IV*, pages 343–357, London, UK, UK, 2002. Springer-Verlag.

[22] A. V. Oppenheim, R. W. Schafer, and J. R. Buck. *Discrete-time Signal Processing (2Nd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.

[23] Y. Park, V. Lepetit, E. Cvlab, and W. Woo. Multiple 3d object tracking for augmented reality. In *Proceedings International Symposium on Mixed and Augmented Reality*, pages 117–120, 2008.

[24] F. Pernici and A. D. Bimbo. Object tracking by oversampling local features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 99(PrePrints):1, 2013.

[25] K. M. M. Prabhu. *Window Functions and their Applications in Signal Processing*. CRC Press, 2013.

[26] A. Roshan Zamir, A. Dehghan, and M. Shah. Gmcp-tracker: Global multi-object tracking using generalized minimum clique graphs. In *Proceedings of the European Conference on Computer Vision*, 2012.

[27] J. van de Weijer, C. Schmid, J. Verbeek, and D. Larlus. Learning color names for real-world applications. *Transactions Image Processing*, 18(7):1512–1523, July 2009.

[28] Y. Wu, J. Lim, and M.-H. Yang. Online object tracking: A benchmark. *IEEE International Conference on Computer Vision and Pattern Recognition*, 0:2411–2418, 2013.

[29] J. yves Bouguet. Pyramidal implementation of the lucas kanade feature tracker. *Intel Corporation, Microprocessor Research Labs*, 2000.

[30] W. Zhong. Robust object tracking via sparsity-based collaborative model. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1838–1845, Washington, DC, USA, 2012. IEEE Computer Society.