

# Exploiting the Circulant Structure of Tracking-by-detection with Kernels

João F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista

Institute of Systems and Robotics, University of Coimbra  
{henriques,ruicaseiro,pedromartins,batista}@isr.uc.pt

**Abstract.** Recent years have seen greater interest in the use of discriminative classifiers in tracking systems, owing to their success in object detection. They are trained online with samples collected during tracking. Unfortunately, the potentially large number of samples becomes a computational burden, which directly conflicts with real-time requirements. On the other hand, limiting the samples may sacrifice performance. Interestingly, we observed that, as we add more and more samples, the problem acquires circulant structure. Using the well-established theory of Circulant matrices, we provide a link to Fourier analysis that opens up the possibility of extremely fast learning and detection with the Fast Fourier Transform. This can be done in the dual space of kernel machines as fast as with linear classifiers. We derive closed-form solutions for training and detection with several types of kernels, including the popular Gaussian and polynomial kernels. The resulting tracker achieves performance competitive with the state-of-the-art, can be implemented with only a few lines of code and runs at hundreds of frames-per-second. MATLAB code is provided in the paper (see Algorithm 1).



## 1 Introduction

Tracking is a fundamental problem in computer vision, with applications in video surveillance, human-machine interfaces and robot perception. Even though some settings allow for strong assumptions about the target [1, 2], sometimes it is desirable to track an object with little a-priori knowledge. Model-less tracking consists of learning and adapting a representation of the target online.

A very successful approach has been tracking-by-detection [3–7]. This stems directly from the development of powerful discriminative methods in machine learning, and their application to detection with offline training. Many of these algorithms can be adapted for online training, where each successful detection provides more information about the target.

Almost all of the proposed methods have one thing in common: a sparse sampling strategy [3, 5–7]. In each frame, several samples are collected in the target’s neighborhood, where typically each sample characterizes a subwindow the same size as the target (illustrated in Table 1). Clearly, there is a lot of redundancy, since most of the samples have a large amount of overlap. This

**Table 1:** Overview of the main differences between standard tracking-by-detection and the proposed approach. The speed is for a  $64 \times 64$  window region. See text for details.

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

underlying structure is usually ignored. Instead, most methods simply collect a small number of samples, because the cost of not doing so would be prohibitive.

The fact that the training data has so much redundancy means that we are probably not exploiting its structure efficiently. We propose a new theoretical framework to address this. We show that the process of taking subwindows of an image induces *circulant structure*. We then establish links to Fourier analysis that allows the use of the Fast Fourier Transform (FFT) to quickly incorporate information from all subwindows, without iterating over them.

These developments enable new learning algorithms that can be orders of magnitude faster than the standard approach. We also show that classification on non-linear feature spaces with the Kernel Trick can be done as efficiently as in the original image space.

### 1.1 Previous work

We will briefly discuss tracking-by-detection, but also other works that are relevant to our specific approach.

The literature on visual object tracking is extensive, and a full survey is outside the scope of this paper.<sup>1</sup> Like other works in tracking-by-detection, our contributions are focused on the appearance model, as opposed to the motion model and search strategy. Many use established learning algorithms such as Boosting [6, 3], Support Vector Machines (SVM) [5], or Random Forests [7], and adapt them to online training. Recent works have focused increasingly on problems specific to tracking, such as uncertainty in the training labels. Some notable examples use Semi-Supervised Learning [6] and Multiple Instance Learning [3] (MILTrack) to handle this. Going even further, Hare et al. [4] propose Struck, an online version of Structured Output SVM. This is closer to our work, since the framework allows sample selection over the possible subwindows (argmax step). However, in practice, the number of samples is still limited.

The idea of exploring subwindow redundancy has been noted before, but mostly in the context of detection, not training. Lampert et al. [10] use branch-and-bound optimization to find the maximum of a classifier's response without

<sup>1</sup> We refer the reader to 2 reviews: [8] is more in-depth, while [9, Sec. 3] is more recent.

necessarily evaluating it at all locations. Alexe et al. [11] propose a method that can efficiently find the most similar subwindows between two images, which is a related problem. Although they are useful and provide interesting insights, it may still be desirable to compute the responses at many locations, for example to allow more robust mode seeking or to evaluate the quality of the response [12]. An alternative is to use linear classification in a first stage, and then non-linear classification on promising locations [13, 14], but the results can be suboptimal.

Also closely related are adaptive correlation filters, rooted on classical signal processing [15, 12]. Their response can be evaluated quickly at all subwindows using the Fast Fourier Transform (FFT). It's possible to perform training on the Fourier domain as well, minimizing the error of the filter's response at all subwindows of the training images. The crucial detail is that they never actually iterate over the subwindows. The Minimum Output Sum of Squared Error (MOSSE) filter [12] has been shown to be competitive with the methods outlined before, but at a fraction of the complexity, and runs at impressive speeds.

Because they can be interpreted as linear classifiers, there is the question of whether correlation filters can take advantage of the Kernel Trick to classify on richer non-linear feature spaces. Patnaik and Casasent [16] investigate this problem, and show that, given the Fourier representation of an image, many classical filters cannot be kernelized. Instead, they propose a kernelized filter that is trained with a single subwindow (called Kernel SDF). An ideal solution would implicitly train with all subwindows.

We believe that the method we propose achieves this goal. We are able to devise Kernel classifiers with the same characteristics as correlation filters, namely their ability to be trained and evaluated quickly with the FFT.

## 1.2 Contributions

The contributions of this paper are as follows:

1. A theoretical framework to study generic classifiers that are trained with all subwindows (of fixed size) of an image. We call this approach *dense sampling*.
2. Proof that the kernel matrix in this case has circulant structure, for unitarily invariant kernels (Theorem 1).
3. Closed-form, fast and exact solutions (all running in  $\mathcal{O}(n^2 \log n)$  for  $n \times n$  images) for:
  - (a) Kernel Regularized Least Squares with dense sampling (Section 2.4).
  - (b) Detection at all subwindows with generic Kernel classifiers (Section 2.5).
  - (c) Computation of a variety of kernels at all subwindows, including the popular Gaussian and polynomial kernels (Section 3).
4. Finally, we propose a tracker based on these ideas. We show it is competitive with state-of-the-art trackers, but has a simpler implementation and runs many times faster. Source code is provided.



**Fig. 1:** Example results for *coke* and *surfer* sequences, best viewed in color. High values in the response map are red/opaque, low values are blue/transparent. Notice the highly localized responses, except when the target is under occlusion.

## 2 Learning with dense sampling

The core component in tracking-by-detection is a classifier. Each frame, a set of samples is collected around the estimated position of the target; samples close to the target are labeled positive and the ones further away are labeled negative. Updating the classifier with these samples allows it to adapt over time. Due to computational constraints, only a handful of random samples are collected [3–7].

We propose a radically different approach. We intend to train a classifier with *all* samples: we call this *dense sampling*. Counter to intuition, this allows a more efficient training. The reason is that the kernel matrix in this case becomes highly structured, and we can exploit it to our advantage.

### 2.1 Regularized risk minimization

We start with a general formulation, mostly to introduce notation. Given a set of training patterns and labels  $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)$ , a classifier  $f(\mathbf{x})$  is trained by finding the parameters that minimize the regularized risk. A linear classifier has the form  $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$ , where  $\langle \cdot, \cdot \rangle$  is the dot product, and the minimization problem is

$$\min_{\mathbf{w}, b} \sum_{i=1}^m L(y_i, f(\mathbf{x}_i)) + \lambda \|\mathbf{w}\|^2, \quad (1)$$

where  $L(y, f(\mathbf{x}))$  is a loss function, and  $\lambda$  controls the amount of regularization<sup>2</sup>.

This framework includes the popular Support Vector Machine (SVM), which uses the hinge loss  $L(y, f(\mathbf{x})) = \max(0, 1 - yf(\mathbf{x}))$ . An alternative is Regularized Least Squares (RLS), also known as Ridge Regression, which uses the quadratic loss  $L(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$ . It has been shown that, in many practical problems, RLS offers equivalent classification performance to SVM [17].

It is well known that the Kernel Trick [18] can improve performance further, by allowing classification on a rich high-dimensional feature space. The inputs are mapped to the feature space using  $\varphi(\mathbf{x})$ , defined by the kernel  $\kappa(\mathbf{x}, \mathbf{x}') = \langle \varphi(\mathbf{x}), \varphi(\mathbf{x}') \rangle$ . The Representer Theorem [18, p. 89] then states that a solution can be expanded as a linear combination of the inputs:  $\mathbf{w} = \sum_i \alpha_i \varphi(\mathbf{x}_i)$ .

<sup>2</sup> The bias term  $b$  is not important in practice, when finding the maximum response.

Then, RLS with Kernels (KRLS) has the simple closed form solution [17]

$$\boldsymbol{\alpha} = (K + \lambda I)^{-1} \mathbf{y}, \quad (2)$$

where  $K$  is the kernel matrix with elements  $K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$ ,  $I$  is the identity matrix, and the vector  $\mathbf{y}$  has elements  $y_i$ . The solution  $\mathbf{w}$  is implicitly represented by the vector  $\boldsymbol{\alpha}$ , whose elements are the coefficients  $\alpha_i$ . We will show that the matrix inversion in Eq. 2 can be avoided entirely for our purposes.

## 2.2 Circulant matrices

The main observation that will allow efficient learning is that, under suitable conditions, the kernel matrix becomes *circulant*. An  $n \times n$  circulant matrix  $C(\mathbf{u})$  is obtained from the  $n \times 1$  vector  $\mathbf{u}$  by concatenating all possible cyclic shifts of  $\mathbf{u}$ :

$$C(\mathbf{u}) = \begin{bmatrix} u_0 & u_1 & u_2 & \cdots & u_{n-1} \\ u_{n-1} & u_0 & u_1 & \cdots & u_{n-2} \\ u_{n-2} & u_{n-1} & u_0 & \cdots & u_{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ u_1 & u_2 & u_3 & \cdots & u_0 \end{bmatrix}. \quad (3)$$

The first row is vector  $\mathbf{u}$ , the second row is  $\mathbf{u}$  shifted one element to the right (the last element wraps around), and so on.

The motivation behind circulant matrices is that they encode the convolution of vectors, which is conceptually close to what we do when evaluating a classifier at many different subwindows. Since the product  $C(\mathbf{u})\mathbf{v}$  represents convolution of vectors  $\mathbf{u}$  and  $\mathbf{v}$  [19], it can be computed in the Fourier domain, using

$$C(\mathbf{u})\mathbf{v} = \mathcal{F}^{-1}(\mathcal{F}^*(\mathbf{u}) \odot \mathcal{F}(\mathbf{v})), \quad (4)$$

where  $\odot$  is the element-wise product, while  $\mathcal{F}$  and  $\mathcal{F}^{-1}$  denote the Fourier transform and its inverse, respectively, and  $*$  is the complex-conjugate.

The properties of circulant matrices make them particularly amenable to manipulation, since their sums, products and inverses are also circulant [19]. We never have to explicitly compute and store a circulant matrix  $C(\mathbf{u})$ , because it is defined by  $\mathbf{u}$ . These operations often involve the Fourier Transform of  $\mathbf{u}$ .

There are a couple of different definitions of  $C(\mathbf{u})$  that we will find useful [19]. One is that the row  $i$  of  $C(\mathbf{u})$  is given by  $P^i \mathbf{u}$ , where  $P$  is the permutation matrix that cyclically shifts  $\mathbf{u}$  by one element. The matrix power in  $P^i$  applies the permutation  $i$  times, resulting in  $i$  cyclic shifts.

Alternatively, the elements of  $C(\mathbf{u})$  can be defined as  $c_{ij} = u_{(j-i) \bmod n}$ . That is, a matrix is circulant if its elements only depend on  $(j-i) \bmod n$ , where mod is the modulus operation (remainder of division by  $n$ ). To make some derivations easier, all indexes are zero-based.

### 2.3 The kernel matrix with dense sampling

We introduce the concept of dense sampling. For a matter of clarity, we start with one-dimensional images with a single feature (ie., the pixel value). This allows more intuitive proofs with simpler notation. However, they are readily transferable to the case of 2D images with multiple channels, such as RGB images, and dense SIFT or HOG descriptors. Appendix A.3 presents more details.

Given a single image  $\mathbf{x}$ , expressed as a  $n \times 1$  vector, the samples are defined as

$$\mathbf{x}_i = P^i \mathbf{x}, \quad \forall i = 0, \dots, n-1 \quad (5)$$

with  $P$  the permutation matrix that cyclically shifts vectors by one element, as defined earlier. Intuitively, the samples are all possible translated versions of  $\mathbf{x}$  (except at the boundaries, discussed in Section 4.1). We will now prove that the resulting kernel matrix is circulant, and show under what conditions.

**Theorem 1.** *The matrix  $K$  with elements  $K_{ij} = \kappa(P^i \mathbf{x}, P^j \mathbf{x})$  is circulant if  $\kappa$  is a unitarily invariant kernel.*

*Proof.* A kernel  $\kappa$  is unitarily invariant if  $\kappa(\mathbf{x}, \mathbf{x}') = \kappa(U\mathbf{x}, U\mathbf{x}')$  for any unitary matrix  $U$ . Since permutation matrices are unitary,  $K_{ij} = \kappa(P^i \mathbf{x}, P^j \mathbf{x}) = \kappa(P^{-i} P^i P^i \mathbf{x}, P^{-i} P^j \mathbf{x}) = \kappa(\mathbf{x}, P^{j-i} \mathbf{x})$ . Because  $K_{ij}$  depends only on  $(j-i) \bmod n$ ,  $K$  is circulant.

**Corollary 1.**  *$K$  as defined above is circulant for dot-product and radial basis function kernels. Particular examples are the polynomial and Gaussian kernels.*

This is an important property that allows the creation of efficient learning algorithms. We will now focus on applying this knowledge to KRLS.

### 2.4 Efficient Kernel Regularized Least Squares solution

Theorem 1 is readily applicable to KRLS. We will define vector  $\mathbf{k}$  with elements

$$k_i = \kappa(\mathbf{x}, P^i \mathbf{x}), \quad \forall i = 0, \dots, n-1 \quad (6)$$

which compactly represents the kernel matrix  $K = C(\mathbf{k})$ . Notice that  $\mathbf{k}$  is only  $n \times 1$ , while the full  $K$  would be  $n \times n$ .

Some operations on matrices of the form  $C(\mathbf{u})$ , like multiplication and inversion, can be done element-wise on the vectors  $\mathbf{u}$ , if they are transformed to the Fourier domain [19].

By applying these properties to Eq. 2 and Eq. 6, we obtain the KRLS solution:

$$\boldsymbol{\alpha} = \mathcal{F}^{-1} \left( \frac{\mathcal{F}(\mathbf{y})}{\mathcal{F}(\mathbf{k}) + \lambda} \right), \quad (7)$$

where the division is performed element-wise. A detailed proof is in Appendix A.1.

Note that the vector  $\boldsymbol{\alpha}$  contains all the  $\alpha_i$  coefficients. This closed-form solution is very efficient: it uses only Fast Fourier Transform (FFT) and element-wise operations. We'll see in Sec. 3 that  $\mathbf{k}$  can also be computed quickly with the FFT.

For  $n \times n$  images, the proposed algorithm has a complexity of only  $\mathcal{O}(n^2 \log n)$ , while a naive KRLS implementation would take  $\mathcal{O}(n^4)$  operations. This is done without reducing the number of samples, which would sacrifice performance.

## 2.5 Fast detection

The general formula for computing the classifier response for a single input  $\mathbf{z}$  is

$$y' = \sum_i \alpha_i \kappa(\mathbf{x}_i, \mathbf{z}). \quad (8)$$

This formula is typically evaluated at all subwindows, in a sliding-window manner. However, we can exploit the circulant structure to compute all the responses simultaneously and efficiently. Using the properties discussed earlier, the vector with the responses at *all* positions is given by

$$\hat{\mathbf{y}} = \mathcal{F}^{-1}(\mathcal{F}(\bar{\mathbf{k}}) \odot \mathcal{F}(\boldsymbol{\alpha})), \quad (9)$$

where  $\bar{\mathbf{k}}$  is the vector with elements  $\bar{k}_i = \kappa(\mathbf{z}, P^i \mathbf{x})$ . We provide an extended proof in Appendix A.2. Just like the formula for KRLS training, the complexity is bound by the FFT operations and is only  $\mathcal{O}(n^2 \log n)$  for 2D images.

## 3 Fast computation of non-linear kernels

The proposed training procedure is fast, but the question of how to evaluate non-linear kernels quickly for all subwindows (ie., compute  $\mathbf{k}$  and  $\bar{\mathbf{k}}$ ) still remains. As of this writing, this is a topic of active research [10, 11, 16].

Linear kernels are usually preferred in time-critical problems such as tracking, because the weights vector  $\mathbf{w}$  can be computed explicitly. Non-linear kernels require iterating over all samples (or support vectors). The work that comes closest to the goal of efficiently computing non-linear kernels at all locations is by Patnaik [20]. Unfortunately, it requires inputs that have unit norm, and the normalization may discard important information.

In this work, we propose closed-form solutions to compute a variety of kernels at all image locations, in an efficient manner that fully exploits the problem structure. The formulas are exact, and simple to compute.

### 3.1 Dot-product kernels

Dot-product kernels have the form  $\kappa(\mathbf{x}, \mathbf{x}') = g(\langle \mathbf{x}, \mathbf{x}' \rangle)$ , for some function  $g$ . In this case, the compact representation  $\mathbf{k}$  of the kernel matrix (Eq. 6) will be denoted by  $\mathbf{k}^{\text{dp}}$ . Each element of  $\mathbf{k}^{\text{dp}}$  is given by

$$k_i^{\text{dp}} = \kappa(\mathbf{x}, P^i \mathbf{x}') = g(\mathbf{x}^T P^i \mathbf{x}'). \quad (10)$$

With slight abuse of notation, we will say that  $g$  can also be applied element-wise to an input vector, so  $\mathbf{k}^{\text{dp}}$  can be written as  $\mathbf{k}^{\text{dp}} = g(C(\mathbf{x}') \mathbf{x})$ .

Using the convolution property from Eq. 4, we obtain the solution

$$\mathbf{k}^{\text{dp}} = g(\mathcal{F}^{-1}(\mathcal{F}(\mathbf{x}) \odot \mathcal{F}^*(\mathbf{x}'))). \quad (11)$$

Eq. 11 means that a dot-product kernel can be quickly evaluated at all image locations, using only a few FFT and element-wise operations. In particular, for a polynomial kernel,

$$\mathbf{k}^{\text{poly}} = (\mathcal{F}^{-1}(\mathcal{F}(\mathbf{x}) \odot \mathcal{F}^*(\mathbf{x}')) + c)^d. \quad (12)$$

### 3.2 Radial Basis Function kernels

RBF kernels have the form  $\kappa(\mathbf{x}, \mathbf{x}') = h(\|\mathbf{x} - \mathbf{x}'\|^2)$ , for some function  $h$ . The corresponding  $\mathbf{k}$  from Eq. 6 will be denoted by  $\mathbf{k}^{\text{rbf}}$ .

$$k_i^{\text{rbf}} = \kappa(\mathbf{x}, P^i \mathbf{x}') = h(\|\mathbf{x} - P^i \mathbf{x}'\|^2) \quad (13)$$

We can expand the norm, obtaining

$$k_i^{\text{rbf}} = h(\|\mathbf{x}\|^2 + \|\mathbf{x}'\|^2 - 2\mathbf{x}^T P^i \mathbf{x}'). \quad (14)$$

The permutation  $P^i$  doesn't affect the norm of  $\mathbf{x}'$  due to Parseval's identity. Since  $\|\mathbf{x}\|^2$  and  $\|\mathbf{x}'\|^2$  are constant w.r.t.  $i$ , Eq. 14 is in the same form as for dot-product kernels. Following the same derivation as in Section 3.1, we arrive at the general solution for RBF kernels

$$\mathbf{k}^{\text{rbf}} = h(\|\mathbf{x}\|^2 + \|\mathbf{x}'\|^2 - 2\mathcal{F}^{-1}(\mathcal{F}(\mathbf{x}) \odot \mathcal{F}^*(\mathbf{x}'))). \quad (15)$$

In particular, we have, for the Gaussian kernel,

$$\mathbf{k}^{\text{gauss}} = \exp\left(-\frac{1}{\sigma^2}(\|\mathbf{x}\|^2 + \|\mathbf{x}'\|^2 - 2\mathcal{F}^{-1}(\mathcal{F}(\mathbf{x}) \odot \mathcal{F}^*(\mathbf{x}')))\right). \quad (16)$$

For an  $n \times n$  image, direct kernel computation at  $n^2$  locations would take  $\mathcal{O}(n^4)$  operations, however the corresponding frequency-domain solution brings this complexity down to only  $\mathcal{O}(n^2 \log n)$ .

The generic formulas we derived for each kernel will quickly compute the  $\mathbf{k}$  and  $\bar{\mathbf{k}}$  terms in KRLS training (Eq. 7) and detection (Eq. 9). We expect them to be of general interest, however, and be useful for other kernel methods.

### 3.3 The linear case

The simplest kernel function,  $\kappa(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$ , which is just the dot-product in the original space, is worth investigating. It produces a linear classifier that does not make use of the Kernel Trick, so we can compute  $\mathbf{w}$  explicitly, instead of implicitly as  $\boldsymbol{\alpha}$ . Plugging it into the KRLS equations, we obtain:



**Table 2:** Tracker precisions at a threshold of 20 (percentage of frames where the predicted location is within 20 pixels of the ground truth). This threshold was used by Babenko et al. [3]. The best precision for each sequence is highlighted in bold.

	MILTrack	Struck	MOSSE	MOSSE <sup>2</sup>	Proposed method
coke11	0.61	0.97	0.71	0.71	<b>1.00</b>
faceocc	0.46	0.96	0.21	<b>1.00</b>	<b>1.00</b>
faceocc2	0.69	0.95	0.53	0.93	<b>1.00</b>
surfer	0.98	0.97	0.37	<b>0.99</b>	<b>0.99</b>
sylvester	0.90	0.95	0.78	0.90	<b>1.00</b>
tiger1	0.83	<b>0.94</b>	0.26	0.30	0.61
tiger2	<b>0.93</b>	0.91	0.25	0.22	0.63
dollar	0.82	0.96	0.39	<b>1.00</b>	<b>1.00</b>
girl	0.31	0.95	0.83	<b>0.99</b>	0.59
david	0.56	<b>0.92</b>	0.77	0.34	0.49
cliffbar	0.89	0.44	0.37	0.56	<b>0.97</b>
twinings	0.98	<b>1.00</b>	0.20	<b>1.00</b>	0.93

$$\mathbf{w} = \mathcal{F}^{-1} \left( \frac{\mathcal{F}(\mathbf{x}) \odot \mathcal{F}^*(\mathbf{y})}{\mathcal{F}(\mathbf{x}) \odot \mathcal{F}^*(\mathbf{x}) + \lambda} \right). \quad (17)$$

This is a kind of correlation filter that has been proposed recently, called Minimum Output Sum of Squared Error (MOSSE) [12, 15], with a single training image. It is remarkably powerful despite its simplicity.

Note, however, that correlation filters are obtained with classical signal processing techniques, directly in the Fourier domain. As we have shown, Circulant matrices are the key enabling factor to extend them with the Kernel Trick.

## 4 Experiments

We used the techniques described above to implement a simple tracking system. Many obvious improvements, like failure detection, motion and uncertainty models (eg., particle filter), or feature extraction, were deliberately left out. This was done to reduce the confounding factors to a minimum, and provide an accurate validation of the learning algorithm.

From now on, we will assume two-dimensional images. A thorough proof is given in Appendix A.3. In practice it means that the 2D Fourier transform can replace the 1D FT in all the previous equations.

### 4.1 Pre-processing

The proposed method can operate directly on the pixel values, with no feature extraction. However, since the Fourier transform is periodic, it does not respect the image boundaries. The large discontinuity between opposite edges of a non-periodic image will result in a noisy Fourier representation. A common solution is to band the original  $n \times n$  image ( $x^{\text{raw}}$ ) with a cosine (or sine) window:

$$x_{ij} = (x_{ij}^{\text{raw}} - 0.5) \sin(\pi i/n) \sin(\pi j/n), \quad \forall i, j = 0, \dots, n-1 \quad (18)$$

Values near the borders will be weighted to zero, eliminating discontinuities.

## 4.2 Training outputs

During training, we must assign a label to each sample. In tracking-by-detection, samples near the target center are positive and others are negative. But since the square loss of KRLS allows for continuous values, we don't need to limit ourselves to binary labels. The line between classification (binary output) and regression (continuous output) is essentially blurred.

Given the choice of a continuous training output, we will use a Gaussian function, which is known to minimize ringing in the Fourier domain [21]. The output will be 1 near the target location  $(i', j')$ , and decay to 0 as the distance increases, with a bandwidth of  $s$ :

$$y_{ij} = \exp(-((i - i')^2 + (j - j')^2)/s^2), \quad \forall i, j = 0, \dots, n-1 \quad (19)$$

The continuous labeling yields spatially smooth classifier responses, which results in more accurate position estimates than binary labeling (Table 2).

## 4.3 Overview

The tracker follows a simple pipeline. A window of a fixed size (double the target size) is cropped from the input image, at the estimated target location. No feature extraction is performed, other than a cosine window on the raw pixel values (Eq. 18). The target is located by evaluating Eq. 9 and finding the maximum response. Eq. 7 is then used to train a new model ( $\alpha$  and  $\mathbf{x}$ ).

To provide some memory, the new model is integrated by linearly interpolating the new parameters with the ones from the previous frame. We found that this scheme, adapted from the work of Bolme et al. [12], is enough for our purposes. Future work will explore other ways to aggregate samples over time.

## 4.4 Evaluation

We compared the proposed method with several state-of-the-art trackers, on 12 challenging videos. We used available ground truth data to compute precisions.

The best way to evaluate trackers is still a debatable subject. Averaged measures like mean center location error or average bounding box overlap can yield unintuitive results, for example penalizing an accurate tracker that fails for a small amount of time more than an inaccurate tracker.

Babenko et al. [3] argue for the use of precision plots. The plots show, for a range of distance thresholds, the percentage of frames that the tracker is within that distance of the ground truth. These plots are easy to interpret. More accurate trackers have high precision at lower thresholds, and if a tracker fails it will never reach a precision of 1 for a large range. They are shown in Fig. 2.

---

**Algorithm 1 : MATLAB code for our tracker, using a Gaussian kernel**  
 It is possible to reuse some values, reducing the number of FFT calls. An implementation with GUI is available at: <http://www.isr.uc.pt/~henriques/>

---

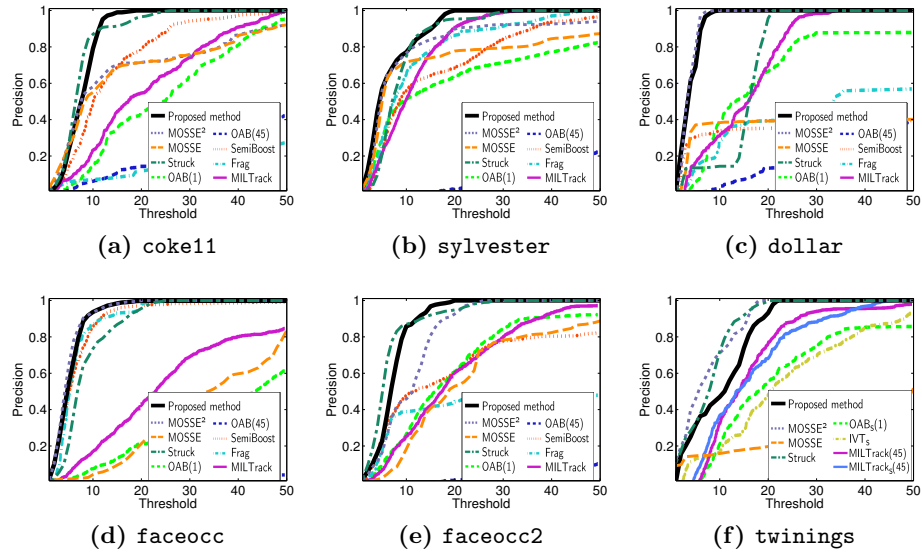
```
% Training image x (current frame) and test image z (next frame)
% must be pre-processed with a cosine window. y has a Gaussian
% shape centered on the target. x, y and z are M-by-N matrices.
% All FFT operations are standard in MATLAB.
```

```
function alphaf = training(x, y, sigma, lambda) % Eq. 7
    k = dgk(x, x, sigma);
    alphaf = fft2(y) ./ (fft2(k) + lambda);
end
```

```
function responses = detection(alphaf, x, z, sigma) % Eq. 9
    k = dgk(x, z, sigma);
    responses = real(ifft2(alphaf .* fft2(k)));
end
```

```
function k = dgk(x1, x2, sigma) % Eq. 16
    c = fftshift(ifft2(fft2(x1) .* conj(fft2(x2))));
    d = x1(:)' * x1(:) + x2(:)' * x2(:) - 2 * c;
    k = exp(-1 / sigma^2 * abs(d) / numel(x1));
end
```

---



**Fig. 2:** Precisions plots for 6 sequences (percentage of frames where the predicted location is within the threshold of the ground truth). Best viewed in color. See the supplemental material for plots of the remaining sequences.

The parameters are *fixed for all videos* to prevent overfitting. We tested our tracker with a Gaussian kernel. A polynomial kernel with appropriate parameters gives similar results, but the Gaussian kernel is easier to adjust, since it has only one parameter with an intuitive meaning. The bandwidth of the Gaussian kernel is  $\sigma = 0.2$ , spatial bandwidth is  $s = \sqrt{mn}/16$  for an  $m \times n$  target, regularization is  $\lambda = 10^{-2}$ , and the interpolation factor for adaptation is 0.075.

We found that MOSSE [12] is tuned only for  $64 \times 64$  images. However, to provide a fair comparison, we made some improvements: regularization  $\lambda = 10^{-4}$ , spatial bandwidth proportional to target size ( $s = \sqrt{mn}/16$ ), no failure detection and no randomized initial samples. This is essentially our system with a linear kernel (Sec. 3.3). We called it MOSSE<sup>2</sup>. All other parameters are the same as with the Gaussian kernel. It has high accuracy on many sequences, but ours shows equal or greater accuracy in 10 of the 12 sequences (see Table 2).

For non-deterministic trackers, we take the median of the precisions over 5 runs. The sequences `twinnings` and `cliffbar` have large scale changes, so we compare with versions of MILTrack [3], Online Ada-Boost (OAB) [3, Sec. 4] and IVT [22] that track through scale. Even without a notion of scale, the proposed method works well in these videos, as shown in Table 2.

Struck [4] achieves very good results (over 0.9 in most sequences), and outperforms other trackers like MILTrack, OAB, SemiBoost [6] and FragTrack [23]. Still, it has lower accuracy than the proposed method because it optimizes bounding box overlap. The proposed tracker is especially geared for high localization, because circulant matrix theory allows it to encode samples from all locations. This includes, as negative samples, both distant distractors and small displacements of the true target. The frequency-domain representation also allows us to minimize ringing (Sec. 4.2), resulting in spatially smooth responses (Fig. 1). This is not possible with unstructured random sampling.

Please note that the goal is *not* merely to show higher precisions. Indeed, every tracker fails in at least one video. However, we can achieve very competitive results with a much simpler and faster tracker. Most recent trackers rely on heavy optimization methods, and manage budgets of support vectors or similar. Our algorithm has only a few lines of code (Algorithm 1) and runs at hundreds of frames-per-second. We also hope our theoretical analysis is of interest in itself.

## 5 Conclusion

We presented a theoretical framework to analyze and explore the consequences of dense sampling in tracking-by-detection. The result is a collection of closed-form, fast and exact solutions for online training, detection, and computation of non-linear kernels. We expect this last contribution to find useful applications outside of tracking. We also hope to have shown that some structures that occur naturally in computer vision, such as Circulants, are still relatively unexplored.

**Acknowledgments.** The authors thank Sam Hare and Boris Babenko, for providing their results. They also acknowledge the FCT project PTDC/EEA-CRO/122812/2010, grants SFRH/BD75459/2010, SFRH/BD74152/2010, and SFRH/BD45178/2008.

## References

1. J.F. Henriques, R. Caseiro, and J. Batista. Globally optimal solution to multi-object tracking with merged measurements. In *ICCV*, 2011.
2. A.R. Zamir, A. Dehghan, and M. Shah. GMCP-Tracker: global multi-object tracking using generalized minimum clique graphs. In *ECCV*, 2012.
3. B. Babenko, M.-H. Yang, and S. Belongie. Robust object tracking with online multiple instance learning. *TPAMI*, 33(8):1619–1632, August 2011.
4. S. Hare, A. Saffari, and P. Torr. Struck: Structured output tracking with kernels. In *ICCV*, 2011.
5. S. Avidan. Support vector tracking. *TPAMI*, 26(8):1064–1072, 2004.
6. H. Grabner, C. Leistner, and H. Bischof. Semi-supervised on-line boosting for robust tracking. In *ECCV*, 2008.
7. A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof. On-line random forests. In *3rd IEEE ICCV Workshop on On-line Computer Vision*, 2009.
8. A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *ACM Computing Surveys*, 38(4):13–58, December 2006.
9. H. Yang, L. Shao, F. Zheng, L. Wang, and Z. Song. Recent advances and trends in visual tracking: A review. *Neurocomputing*, 74(18):3823–3831, November 2011.
10. C.H. Lampert, M.B. Blaschko, and T. Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *CVPR*, 2008.
11. B. Alexe, V. Petrescu, and V. Ferrari. Exploiting spatial overlap to efficiently compute appearance distances between image windows. In *NIPS*, 2011.
12. D. S Bolme, J. R Beveridge, B. A Draper, and Y. M Lui. Visual object tracking using adaptive correlation filters. In *CVPR*, 2010.
13. A. Vedaldi, V. Gulshan, M. Varma, and A. Zisserman. Multiple kernels for object detection. In *ICCV*, 2009.
14. H. Harzallah, F. Jurie, and C. Schmid. Combining efficient object localization and image classification. In *ICCV*, 2009.
15. D. S Bolme, B. A. Draper, and J. R. Beveridge. Average of synthetic exact filters. In *CVPR*, 2009.
16. R. Patnaik and D. Casasent. Fast FFT-based distortion-invariant kernel filters for general object recognition. In *Proceedings of SPIE*, volume 7252, 2009.
17. R. Rifkin, G. Yeo, and T. Poggio. Regularized least-squares classification. *Nato Science Series Sub Series III: Computer and Systems Sciences*, 190:131–154, 2003.
18. B. Schölkopf and A.J. Smola. *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. The MIT Press, 2002.
19. R. M. Gray. *Toeplitz and Circulant Matrices: A Review*. Now Publishers, 2006.
20. R. Patnaik. *Distortion-invariant kernel correlation filters for general object recognition*. PhD thesis, Carnegie Mellon University, 2009.
21. R. C. González and R. E. Woods. *Digital image processing*. Prentice Hall, 2008.
22. D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang. Incremental learning for robust visual tracking. *IJCV*, 77(1-3):125–141, August 2007.
23. A. Adam, E. Rivlin, and I. Shimshoni. Robust fragments-based tracking using the integral histogram. In *CVPR*, 2006.

### Appendix A.1: Dense Sampling KRLS derivation

We will use the fact that  $K$  is circulant, replacing Eq. 6 in the generic KRLS solution of Eq. 2. Observing that any identity matrix  $I$  is circulant,  $I = C(\delta)$  with  $\delta = [1, 0, 0, \dots, 0]^T$ , Eq. 2 becomes

$$\boldsymbol{\alpha} = (C(\mathbf{k}) + \lambda C(\boldsymbol{\delta}))^{-1} \mathbf{y} = (C(\mathbf{k} + \lambda \boldsymbol{\delta}))^{-1} \mathbf{y}. \quad (20)$$

The properties of circulant matrices allow element-wise multiplication and inversion in the Fourier domain [19]. Making use of these properties, and the fact that  $\mathcal{F}(\boldsymbol{\delta}) = \mathbb{1}$ , where  $\mathbb{1}$  is an  $n \times 1$  vector of ones,

$$\boldsymbol{\alpha} = (C(\mathcal{F}^{-1}(\mathcal{F}(\mathbf{k}) + \lambda \mathbb{1})))^{-1} \mathbf{y} = C\left(\mathcal{F}^{-1}\left(\frac{1}{\mathcal{F}(\mathbf{k}) + \lambda}\right)\right) \mathbf{y}. \quad (21)$$

The division is performed element-wise. Using Eq. 4, we finally obtain

$$\boldsymbol{\alpha} = \mathcal{F}^{-1}\left(\frac{\mathcal{F}(\mathbf{y})}{\mathcal{F}(\mathbf{k}) + \lambda}\right). \quad (22)$$

### Appendix A.2: Derivation of fast detection formula

If we denote the test image by  $\mathbf{z}$ , detection amounts to classifying all the shifted test images  $\mathbf{z}_i = P^i \mathbf{z}$ . Each response is then given by

$$\hat{y}_i = \sum_j \alpha_j \kappa(P^i \mathbf{z}, P^j \mathbf{x}), \quad (23)$$

since the training samples are  $\mathbf{x}_i = P^i \mathbf{x}$  (Eq. 5). Rewriting it in matrix notation, the vector of all classifier responses is  $\hat{\mathbf{y}} = C^T(\bar{\mathbf{k}})\boldsymbol{\alpha}$ , where  $\bar{\mathbf{k}}$  is the vector with elements  $\bar{k}_i = \kappa(\mathbf{z}, P^i \mathbf{x})$ . We can now apply the convolution property (Eq. 4):

$$\hat{\mathbf{y}} = \mathcal{F}^{-1}(\mathcal{F}(\bar{\mathbf{k}}) \odot \mathcal{F}(\boldsymbol{\alpha})). \quad (24)$$

### Appendix A.3: Generalization of circulant forms

For a matter of clarity, all of our derivations have assumed that the images are one-dimensional. The 2D case, despite its usefulness, is also more difficult to analyze. The reason is that the 2D generalization of a circulant matrix, related to the 2D Fourier Transform, is a Block-Circulant Circulant Matrix (BCCM, ie., a matrix that is circulant at the block level, composed of blocks themselves circulant). All of the properties we used for circulant matrices have BCCM equivalents.

We will now generalize Theorem 1. A 1D image  $\mathbf{x}$  can be shifted by  $i$  with  $P^i \mathbf{x}$ . With a 2D image  $X$ , we can shift both its rows by  $i$  and its columns by  $i'$  with  $P^i X P^{i'}$ . Additionally, in an  $n^2 \times n^2$  matrix  $M$  composed of  $n \times n$  blocks, we will index the element  $i'j'$  of the block  $ij$  as  $M_{(ii'),(jj')}$ .

**Theorem 2.** *The block matrix  $K$  with elements  $K_{(ii'),(jj')} = \kappa(P^i X P^{i'}, P^j X P^{j'})$  is a BCCM if  $\kappa$  is a unitarily invariant kernel.*

*Proof.* Because  $\kappa$  is unitarily invariant, we have  $K_{(ii'),(jj')} = \kappa(X, P^{j-i} X P^{j'-i'})$ . Since  $K_{(ii'),(jj')}$  depends only on  $(j-i) \bmod n$  and  $(j'-i') \bmod n$ ,  $K$  is BCCM.

$K$  can now be constructed as  $C(K')$ , where the  $n \times n$  matrix  $K'$  has elements  $k_{ii'} = \kappa(X, P^i X P^{i'})$ , and  $C(\cdot)$  constructs a BCCM. The relevant solutions can then be re-derived with the 2D FT in place of the 1D FT.