






Real-Time Embedded Object Tracking with Discriminative Correlation Filters Using Convolutional Features

Michał Danilowicz  and Tomasz Kryjak  

Embedded Vision Systems Group, Computer Vision Laboratory,
Department of Automatic Control and Robotics,
AGH University of Science and Technology, Krakow, Poland
{daniłowi,tomasz.kryjak}@agh.edu.pl

Abstract. Object tracking is an essential element of visual perception systems. It is used in advanced video surveillance systems (AVSS), autonomous vehicles, robotics, and many more. For applications such as autonomous robots, the system must be implemented on some embedded platform with limited computing performance and power. Furthermore, sufficiently fast response is required from the tracking system in order to perform some real-time tasks. Discriminative Correlation Filter (DCF) based tracking algorithms are popular for such applications, as they offer state-of-the-art performance while not being too computationally complex. In this paper, an FPGA implementation of the DCF tracking algorithm using convolutional features is presented. The ZCU104 board is used as a platform, and the performance is evaluated on the VOT2015 dataset. In contrast to other implementations that use HOG (Histogram of Oriented Gradients) features, this implementation achieves better results for 64×64 filter size while being able to potentially operate at higher speeds (over 467 fps per scale).

Keywords: Discriminative correlation filter · Object tracking · FPGA · Real-time image processing

1 Introduction

Object tracking is one of the basic tasks of computer vision. In general, it can be described as determining the objects' positions in consecutive frames. Tracking is used in many civilian applications (autonomous vehicles, advanced surveillance, robotics, human-computer interfaces) and military applications (air defence, targeting systems, missile control systems). Depending on what data the tracking system has and what we expect at the output, there are several subtypes of this problem. We can assume tracking of only one object (VOT – Visual Object Tracking) or several (MOT – Multiple Object Tracking), and decide whether it is necessary to reidentify the object after it has been lost (long-term vs. short-term tracking). It is also important if we track classes of objects known in advance

(model-based tracking) or whether we should be ready to track any arbitrarily indicated fragment of an image. A final distinction is whether we only use the current and previous frames (casual tracker), or whether we also have access to future frames of the image (for example, post-processing of a video surveillance camera recording).

This paper addresses the issue of single object, short-term, model-free and online tracking, which is the premise of the *short-term challenge* of the VOT Challenge [16]. In addition to the effectiveness of the task itself in predicting the displacement of an object between successive frames, another very important parameter of the tracking system is the processing speed. If the time between successive predictions of the tracking system is too long, it may result in a too large change in the position or appearance of the object, resulting in poor performance. Moreover, in some systems, the energy efficiency is also crucial. Examples are solutions for autonomous vehicles in the broad sense.

In recent years, the use of convolutional networks for the generation of image features in computer vision algorithms has become increasingly popular. These features usually allow algorithms to achieve greater efficiency than the image itself or the so-called *hand-crafted features* like HOG (Histogram of Oriented Gradients) or Colour Attributes [10]. However, convolutional networks, particularly deep networks, require a lot of computing power to work in real-time. It is helpful to use platforms that support parallel computing, such as FPGA (*Field-Programmable Gate Array*) or GPU (*Graphics Processing Unit*). Specifically, the first one provides the ability to obtain high processing speed and low energy consumption, thanks to the possibility of optimising the computational architecture and precision of calculations to a specific algorithm. Quantisation of neural networks, i.e. reduction of the number of bits in the representation of processed data and model parameters, allows for a significant reduction of computational and memory complexity of algorithms with little loss of performance [3,21].

In this paper, a hardware implementation of the deepDCF tracking algorithm is presented. Using the FINN compiler for neural network acceleration and parallel computations in FPGA devices we were able to achieve an average processing speed of 467,3 fps (frames per second) per scale at 64×64 filter size.

The main contributions of this paper include:

- Optimisation and analysis of a deepDCF tracking algorithm for implementation on an embedded computing platform and evaluation on the VOT2015 dataset.
- Implementation of a tracking system based on correlation filters using convolutional network features. The system outperforms other similar approaches in tracking performance and speed.

To our knowledge, no paper has been published in which the deepDCF algorithm has been implemented in an FPGA.

The remainder of this paper is organised as follows. Section 2 describes object-tracking methods using correlation filters. Section 3 discusses the state-of-the-art of implementing correlation filters on embedded FPGA platforms. Section 4

presents the evaluation of the software model, the quantisation process of the convolutional network, and the hardware implementation. The last section contains a discussion of the results obtained and directions for further research.

2 Object Tracking with Correlation Filters

In this section, we present the first algorithm in the correlation filter family – MOSSE (Minimum Output Sum of Squared Error) [4], and its subsequent improvements that have been implemented on embedded platforms. It should be noted that there are also other modifications to the algorithm, such as SRDCF (Spatially Regularized Discriminative Correlation Filters), however, they have a much higher computational complexity and are therefore currently not considered for implementation on embedded vision platforms.

The following algorithms share a simple concept. The tracked object model is initialised in the first frame of the video sequence. In subsequent tracking frames, a filter response is obtained by correlating the current object model with a part of the image around the last known object position. The location of the maximum correlation value in the response is used to predict the new object's position. Also, the model is updated taking into account the new, potentially changed, appearance of the object.

2.1 MOSSE

The goal of the MOSSE algorithm is to find an optimal filter $w \in \mathbb{R}^{M \times N}$ (where $M \times N$ is the size of the filter and the tracked region) which is defined by the following LS (*Least Squares*) regression problem:

$$\arg \min_{\hat{w}} \sum_{i=1}^N \|\hat{w}^* \odot \hat{x}_i - \hat{y}_i\|^2 \quad (1)$$

where \odot means element-wise multiplication, $\hat{\cdot}$ denotes a discrete Fourier transform (DFT) of some signal and $*$ is a complex conjugate. The problem is defined in the frequency domain because using the FFT algorithm (Fast Fourier Transform) and the convolution theorem, the computational complexity of the correlation can be lowered from $\mathcal{O}(M^2N^2)$ to $\mathcal{O}(MN \log MN)$. The training set consists of (x_i, y_i) pairs, where $x_i \in \mathbb{R}^{M \times N}$ is a grayscale image patch centered around the target object in the first frame. The samples are generated by applying random affine transformations to the initial object's appearance. For regression targets $y_i \in \mathbb{R}^{M \times N}$, a discrete two-dimensional Gaussian is used.

The problem (1) has a closed-form solution given by:

$$\hat{w} = \frac{\sum_{i=1}^N \hat{y}_i^* \odot \hat{x}_i}{\sum_{i=1}^N \hat{x}_i^* \odot \hat{x}_i} \quad (2)$$

The prediction of object's position is done by computing filter response in pixel coordinates space by using inverse, discrete Fourier transform (IDFT):

$$g = \mathcal{F}^{-1}(\hat{w}_{t-1}^* \odot \hat{x}_t) \quad (3)$$

After every prediction, the filter is updated using a running average to address changes in the object's appearance:

$$\hat{a}_t = \eta \hat{y}^* \odot \hat{x}_t + (1 - \eta) \hat{a}_{t-1} \quad (4)$$

$$\hat{b}_t = \eta \hat{x}_t^* \odot \hat{x}_t + (1 - \eta) \hat{b}_{t-1} \quad (5)$$

where $\hat{w}_t = \frac{\hat{a}_t}{\hat{b}_t}$ (element-wise division), and $\eta \in [0, 1]$ is a learning rate parameter.

2.2 KCF

The correlation filter tracker was further improved by considering tracking as a linear ridge regression problem [13, 14]. The method is called KCF (Kernelized Correlation Filter) and the goal is to find a linear function $f(z) = w^T z$ which minimises the error between samples $x_i \in \mathbb{R}^d$ and regression targets $y_i \in \mathbb{R}$:

$$\epsilon = \sum_i (f(x_i) - y_i)^2 + \lambda \|w\|^2 \quad (6)$$

An interesting conclusion from these works is, that if data matrix X is circulant, the regression problem (6) is equivalent to the MOSSE filter (1) for one sample ($n = 1$). The advantage of such representation of the problem (6) is a possibility to solve it in some nonlinear space $\varphi(x)$ using the *kernel trick* [19]. In brief, a kernel function $\kappa(x, z)$ must be defined which acts as a dot product in non-linear space $\varphi(x)$. For that purpose, a so-called *kernel correlation vector* is computed:

$$k^{xz} = \exp \left(-\frac{1}{\sigma^2} \left(\|x\|^2 + \|z\|^2 - 2\mathcal{F}^{-1} \left(\sum_l^C \hat{x}_l^* \odot \hat{z}_l \right) \right) \right) \quad (7)$$

For tracking, the filter is initialised by:

$$\hat{\alpha} = \frac{\hat{y}}{\hat{k}^{xx} + \lambda} \quad (8)$$

Prediction:

$$\mathbf{f}(z) = \mathcal{F}^{-1}(\hat{k}^{xz} \odot \hat{\alpha}) \quad (9)$$

Update:

$$\hat{\alpha}_t = (1 - \eta) \hat{\alpha}_{t-1} + \eta \hat{\alpha} \quad (10)$$

$$\hat{x}_t = (1 - \eta) \hat{x}_{t-1} + \eta \hat{x} \quad (11)$$

The algorithm offers an improvement in tracking performance compared to MOSSE with a little extra computational complexity that comes from the need to compute the kernel correlation vector (7). Computing the IDFT \mathcal{F}^{-1} and DFTs \hat{x}_l^* is necessary in the MOSSE algorithm anyway and the exponent function operation can be for example stored in LUTs (Look Up Tables) on the target hardware platform.

2.3 DSST

Scale estimation in the tracking system is typically done by predicting the filter at multiple scales. In that case, the object scale corresponds to the largest correlation score obtained. In the paper [8], the concept of DSST (Discriminative Scale Space Tracking) is presented. The method uses an additional correlation filter dedicated to predicting the change in scale. Training and prediction samples $\mathbf{x} \in \mathbb{R}^{S \times D}$ are constructed by generating one-dimensional feature vectors $x_s \in \mathbb{R}^D$ of the object for several scales $s \in [1, S]$. Each vector is extracted from an image patch of size $\beta^n H \times \beta^n W$ centered around the object's position. $\beta > 1$ is a scale factor parameter (typically around 1.01) and $n \in \left\{ \left\lfloor -\frac{S-1}{2} \right\rfloor, \dots, \left\lfloor \frac{S-1}{2} \right\rfloor \right\}$.

This solution to scale prediction has shown better performance while also reducing computational complexity compared to estimating the filter in multiple scales.

2.4 Convolutional Features

Another improvement to the MOSSE algorithm was to use multidimensional image features [9, 14] like histograms of oriented gradients (HOG) or generated by a convolutional neural network. Such algorithms are often called DCF (*Discriminative Correlation Filters*). In such a case, for initialisation, update and prediction, multidimensional samples $x \in \mathbb{R}^{[C \times H \times W]}$ are used, as well as D filters, each for one feature channel.

$$\hat{w}^l = \frac{\hat{y}^* \odot \hat{x}^l}{\sum_{l=1}^C \hat{x}^{l*} \odot \hat{x}^l} \quad (12)$$

$$\hat{a}_t^l = \eta \hat{y}^* \odot \hat{x}_t^l + (1 - \eta) \hat{a}_{t-1}^l \quad (13)$$

$$\hat{b}_t = \eta \sum_{l=1}^C \hat{x}_t^{l*} \odot \hat{x}_t^l + (1 - \eta) \hat{b}_{t-1} \quad (14)$$

$$g = \mathcal{F}^{-1} \left(\sum_{l=1}^C \hat{w}_{t-1}^{l*} \odot \hat{x}_t^l \right) \quad (15)$$

In the paper [6], the *deepDCF* algorithm is presented which utilises a convolutional network to generate image features in the DCF framework. A vgg-2048 [5] model trained for the classification task was used on the ImageNet [11] dataset. The image features x in Eqs.(12)–(15) are generated by the network after applying a preprocessing consisting of scaling the image to a fixed size (in [6], a 224×224 window was used) and normalisation. The output features are then multiplied by a Hann window before applying them to the tracking algorithm.

Only by using convolutional features (in fact, only the first layer), the simple DCF algorithms gave better tracking performance than more complicated

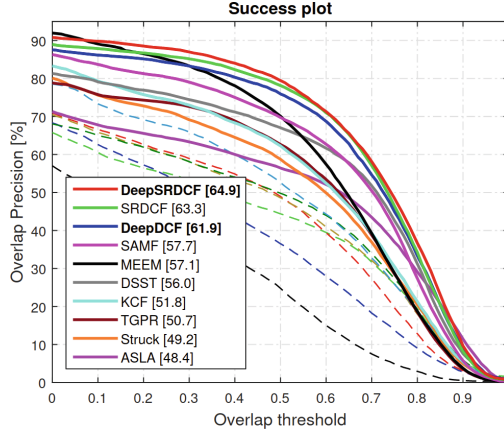


Fig. 1. The comparison of tracking performance of correlation filter based algorithms on the OTB-50 dataset (source: [6]).

methods. The comparison on the OTB-50 dataset [23] with other state-of-the-art tracking algorithms, including other correlation filter methods, is shown in Fig. 1. It is worth mentioning that the KCF and DSST algorithms can be applied regardless of the feature extraction method as long as the features are spatially correlated.

3 Previous Work

The implementation of correlation filters in FPGA devices has been addressed in a number of research papers. The paper [22] presents the implementation of the DCF + DSST algorithm on the Zynq ZedBoard platform (xc7z020clg484-1). The architecture implemented with the Vivado HLS tool offers image processing with a resolution of 320×240 at an average rate of 25.38 fps. An FPGA implementation of the **HOG** generation presented in the article [12] is used as image features. The authors analyse the basic computational steps of the algorithm: SVD (Singular Value Decomposition), QR decomposition (used in feature dimensionality reduction) and the determination of the two-dimensional discrete Fourier transform. The architecture of the QR decomposition algorithm has been optimised and uses 2.3 times less computational resources. The SVD computation has been accelerated nearly 3.8 times with respect to the known FPGA implementation [18], but consumes about twice as many computational resources. The authors did not provide information about the tracking performance after applying the proposed optimisations.

The authors of the article [24] implemented a three-scale KCF algorithm based on HOG features using the Vivado HLS tool. They used the Zynq ZCU102 MPSoC platform and achieved 30 fps for a 960×540 resolution. A brief analysis of the parallelisation of HOG feature generation operations by using the

PIPELINE, *ARRAY_PARTITION* and *DATAFLOW* directives of the HLS tool was performed. Attention was also drawn to the possibility of parallelising the computation of kernel feature correlation (Eq. (7)), detection (Eq. (9)) and filter update (Eq. (8)). No optimisation of the Fourier transform calculation was performed, and the function available in the HLS library was used. The effectiveness of the algorithm was compared with other state-of-the-art methods on the UAV123 set, however, only a selected part of the test sequences was used. The comparison is not very reliable if only for the reason that the authors obtained a better result with the KCF algorithm than with SRDCF, which is directly a better algorithm in other comparisons in the literature [7].

The work [17] presents an implementation of the KCF + DSST algorithm using the Vivado HLS tool. A processing rate of 25 fps was achieved, although it is not clear for which frame size and filter size. The HOG features were used. Only a qualitative (visualisation of sample frames from the sequence) evaluation of the tracking performance on sequences prepared by the authors and selected from the OTB set was presented. No quantitative evaluation and comparison with other methods or implementations in view of the applied optimisations was provided.

The publication [15] describes the implementation of the MOSSE algorithm in one scale on the Zynq UltraScale+ MPSoC ZCU104 platform. The Verilog hardware description language was used, which generally allows for lower FPGA resource requirements (for example, for the dot product operation [3]). Filter initialisation procedure was implemented on the processing system of the platform due to iterative operation and the need to implement affine transformations. The two-dimensional discrete Fourier transform was implemented by utilising two Xilinx FFT modules for one-dimensional signals and a BRAM to transpose the data. The system operates on a real-time video stream at 60 fps for 64×64 filter size.

The paper [25] presents a single scale KCF + HOG algorithm on the ZYNQ-7000 (xc7z100ffg900-2) platform implemented in Vivado HLS. A simpler linear kernel function was used and the filter update mechanism was abandoned in favour of lower computational complexity. An evaluation of the tracking efficiency of the implemented system was performed based on a set of own 5 sequences containing drones. No comparison of effectiveness with state of the art on common benchmarks was provided. A processing speed of 41 fps was obtained.

In the work [20], the authors describe an FPGA implementation of the DCF + DSST filter on the XC7K325T FPGA device. The system achieves processing speed of 153 fps on 33 image channels (one grayscale and 32 HOG) with filter size of 32×32 . However, the paper does not include any evaluation results of the tracking performance. Also, no implementation details are mentioned (HLS or VHDL, Verilog).

The works presented in this section mostly lack evaluation of tracking quality and comparison to other implementations. All described hardware implementations are using HOG or grayscale features, which impacts processing speed

or tracking performance. Additionally, most implementations utilise HLS (High Level Synthesis) tools for development, which introduces resource usage overhead compared to hardware description languages approaches like VHDL or Verilog. This issue was discussed in detail in [3] in the case of dot product computation.

In this paper, we present the use of convolutional features to achieve higher tracking performance with less computational complexity than HOG feature-based solutions, which further allows for higher processing speed.

4 The Proposed CF Implementation

The main concept of this paper is to prove that choosing a convolutional network as a feature extractor for correlation filter tracking not only gives better performance than HOG features, but also can be efficiently accelerated on FPGA to achieve high processing speeds. The work started with the Python implementation of a software model of the deepDCF algorithm. The environment was chosen mainly due to the presence of libraries suitable for testing and training neural networks such as PyTorch. It was also possible to use the official evaluation tools for the VOT Challenge, which are available in Python. In addition, it was possible to use the Brevitas tool, which is a wrapper for the PyTorch library and performs neural network calculations, with a fixed precision (for instance, 8 bits or even 1 bit).

4.1 CNN Quantisation Using Knowledge Transfer

First, the quantisation of the convolutional layer generating features for the filter was performed. For this purpose, the PyTorch library was used to implement learning on the ImageNet set. The training was organised in the knowledge transfer style, i.e. it assumes the presence of a teacher model performing the computation in full precision and a quantised student model. The teacher model was the first layer (including maxpooling and ReLU (Rectified Linear Unit)) of the VGG11 network, pre-trained for the classification task on the ImageNet set. In preliminary experiments, it was noted that reducing the precision in the representation of weights and activations to four bits did not introduce a large increase in learning error. Additional experiments could be conducted to test the effect of different degrees of quantisation of the feature-generating network on tracking quality. Furthermore, the architecture of the student was identical to that of the teacher. The student model was also initialised with the weights of the teacher model. The cost function was the mean square error between the features returned by the teacher model and the student model, while the training was carried out with the SGD (Stochastic Gradient Descent) algorithm with parameters $learning_rate = 0.01$, $momentum = 0.9$, $weight_decay = 10^{-4}$.

4.2 Software Model Evaluation on VOT2015

The VOT (Visual Object Tracking) challenge environment was used to evaluate the tracking performance of the software model. Our results were compared with

those of the KCF and DSST algorithms published by the organisers of the VOT challenge 2015 [2]. Accuracy (A) represents the average IoU (Intersection over Union) between the object position returned by the algorithm and the reference position in each image frame (both described by bounding boxes). Robustness (R), on the other hand, is the ratio of frames in which the object was lost to all frames in the tested sequence. The decisive metric in ranking the algorithms in the competition is EAO (Expected Average Overlap), which takes into account both accuracy and robustness. The data is summarised in Table 1.

Table 1. The table compares our software model to the state-of-the-art correlation filter tracking algorithms used in FPGA implementations. The **bolded, underlined** model was implemented in hardware and discussed in Sect. 4. The evaluation was done on the VOT 2015 dataset. Arrows denote whether more is better \uparrow or less is better \downarrow for a given evaluation metric. Using just 8 channels of convolutional features and 3 scales for 64×64 filter gives better results than KCF and DSST filters on HOG features used in current hardware implementations.

Algorithm	Features	ROI size	A \uparrow	R \downarrow	EAO \uparrow
deepDCF (multiscale, not precised) (original impl. [6])	CONV (96 channels float precision)	224×224 (112×112 filter)	0.48	1.75	(not given)
deepDCF (5 scales) (our implementation)	CONV (32 channels 4bit quantisation)	224×224 (112×112 filter)	0.505	1.829	0.207
deepDCF (5 scales) (our implementation)	CONV (64 channels 4bit quantisation)	224×224 (112×112 filter)	0.484	1.879	0.203
deepDCF (3 scales) (our implementation)	CONV (32 channels 4bit quantisation)	128×128 (64×64 filter)	0.494	1.92	0.184
deepDCF (3 scales) (our implementation)	CONV (8 channels 4bit quantisation)	128×128 (64×64 filter)	0.491	2.082	0.183
deepDCF (3 scales) (our implementation)	CONV (16 channels 4bit quantisation)	128×128 (64×64 filter)	0.487	1.975	0.174
DSST (evaluated by VOT commitee)	HOG	$2 \times$ target size (same as filter)	0.54	2.56	0.17
KCF (evaluated by VOT commitee)	HOG	$2.5 \times$ target size	0.48	2.17	0.17
deepDCF (3 scales) (our implementation)	CONV (4 channels 4bit quantisation)	128×128 (64×64 filter)	0.456	2.611	0.145

The results in Table 1 confirm that the use of features from a single convolutional layer instead of HOG provides better results. The thesis is further strengthened by the fact that the mechanisms of better and faster scale prediction (DSST) and nonlinear regression (KCF) can also be used with convolutional features, which is one of the directions of our further work. In addition, an interesting finding is that it was possible to reduce the channels used by the filter to eight (in the deepDCF work there were 96 channels originally) without a signif-

icant decrease in tracking performance. The difference becomes only significant when the number of channels is reduced to 4.

4.3 Multichannel MOSSE Filter Implementation on FPGA

Based on the software model, the deepDCF algorithm was implemented in the SystemVerilog hardware description language. The work started with the analysis of the project [15], in which the single-channel MOSSE algorithm was implemented. The solution used as a video source a 4K video stream fed to the programmable logic (PL) through an HDMI port. The first change was to switch to communication between the PL and the processing system (PS, ARM-based in the considered device) to send image data and receive the new object’s position. This makes hardware debugging easier because one can easily verify intermediate data like image features and current filter coefficients using DMA in PYNQ environment. The input images are cropped by the PS and the ROI is send to the PL via DMA transfer. However, it is also possible to restore the original video source by adding a module that crops the object from the image and scales it to the desired size. This is one of our future steps.

The top-level diagram is shown in Fig. 2. DMA communicates with the PS through the memory-mapped AXI interface and provides AXI Stream ports to send video to the convolutional network module and to receive the filter response. The task of the PS is to read a video frame, crop an image patch at the current position of the object, and send this fragment to the PL. The image is processed by the convolutional network and the filter module, which finally returns the object’s position displacement and any possible change in scale.

The FINN [3, 21] tool was used to implement the trained convolutional layer in the FPGA. This is an experimental environment from AMD Xilinx for implementing neural networks in selected MPSoCs¹. The tool is based on the finnhlslib library [1] in which basic modules are defined, and a compiler that transforms the network architecture description from Brevitas to a graph composed of these basic modules. FINN also offers the generation of a processing system driver to communicate with the FPGA, but this feature was not used in this project.

The schematic of the DCF multichannel filter module is shown in Fig. 3. All channels of a given DCNN feature pixel are fed in parallel to the module input. The BRAM modules were used as read-only memories for the Hann window parameters and for the two-dimensional Gaussian distribution pre-calculated in the software model. The input feature channels are split into parallel Channel filter modules, each implements Eqs. (12)–(15) for its channel. For the prediction step (logic highlighted in green in the diagrams), the filter responses from each channel are summed, and then the inverse Fourier transform (Eq. (15)) is computed. The prediction is followed by the filter update step (logic highlighted in red), for which the sum of the energy spectrums over all input channels must

¹ In previous FINN versions, Alveo boards were also supported (up to v0.7).

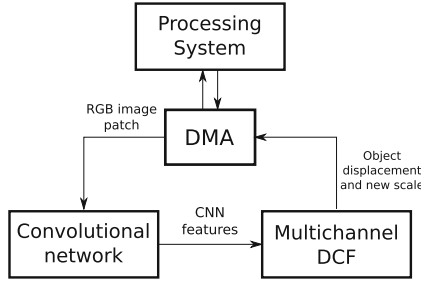


Fig. 2. Top level diagram of the implemented design. The image patch containing the tracked object is prepared by the Processing System by cropping and resizing the video frame. It is sent to the Programmable Logic via DMA (Direct Memory Access) module which streams the data to the convolutional network module generated by FINN. The tracking is done in the Multichannel DCF module which outputs the predicted object displacement back to the DMA.

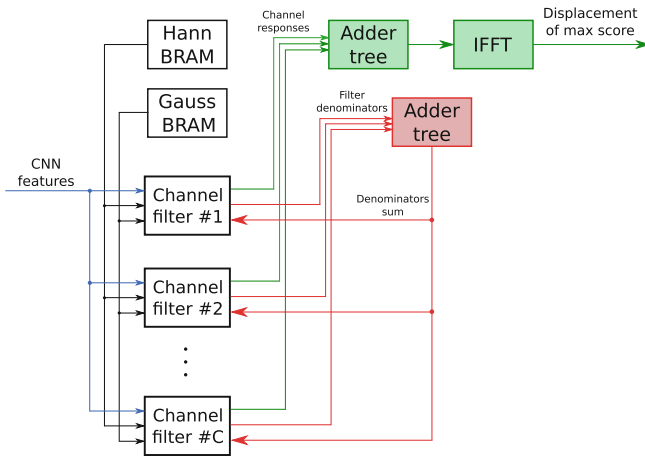


Fig. 3. Diagram of the main filter module. One of the advantages of the algorithm is the possibility of full parallelisation among channels. Hann BRAM and Gauss BRAM are used as read-only memory for storing pre-computed windowing function and the Gaussian distribution.

be computed (Eq. (14)). Since each filter channel is updated independently, the sum from the red adder tree is returned to the Channel filter module.

The Channel filter module is part of the design proposed in [15] with some modifications. The schematic is shown in Fig. 4. Although object features of size 64×64 are processed for prediction (this is also the size of the filter), a wider image context is sent to the module because the update must be performed on a new object position. For this purpose, the entire wider image context is written to BRAM in parallel. After the prediction is completed (i.e., after the responses from

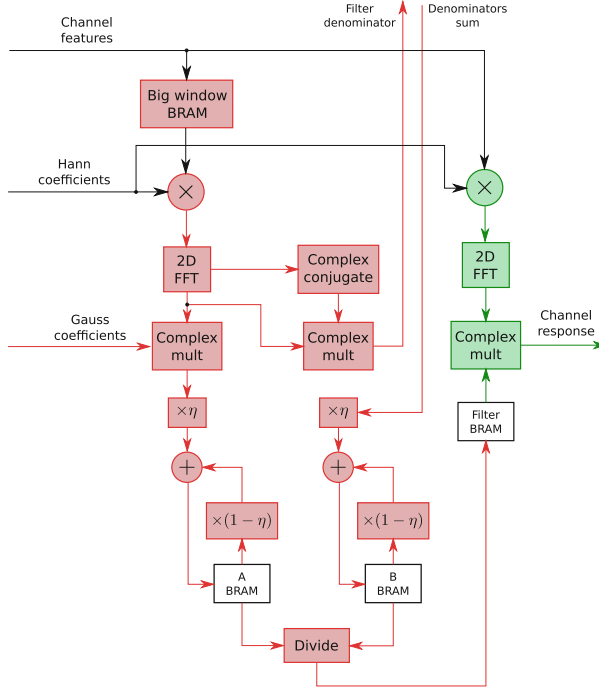


Fig. 4. Diagram of the module responsible for a single input feature channel. It implements multichannel DCF filter Eqs. (12)–(15). Logic responsible for update and prediction are highlighted in red and green respectively. Because the filter needs to be updated at the new, predicted object location, a wider context of the object features must be saved in the Big window BRAM.

the individual channels are summed up and the IFFT is calculated), the position of the feature patch is known and needs to be read from Big window BRAM.

To implement the two-dimensional Fourier transform, the IP provided by Xilinx was used to calculate the one-dimensional transform of each row of input data. These results are then stored in BRAM, from which they are read column-wise into a second one-dimensional transform module.

The hardware implementation was validated using the software model and yielded the same results on sequences from the VOT2015 set. The FPGA resource consumption of the system implementation for an eight-channel 64×64 filter is shown in Table 2. We used 32-bit fixed point precision in the calculations required by the filter and the system currently operates in one scale.

Table 2. Resource utilisation for the implemented tracking system with 64×64 filter and 8 feature channels. Notice that the convolutional layer module uses only a small portion of all FPGA resources utilised by the system.

Resource	Used	Available	% utilisation	% utilisation (CNN only)
LUT	156663	230400	68,00	5,40
LUTRAM	15436	101760	15,17	1,77
FF	334373	460800	72,56	3,18
BRAM	270,5	312	86,70	1,92
DSP	480	1728	27,78	0

For the above implementation, an average reprogrammable logic processing speed of **467.3 fps** was obtained for a 375 MHz clock (single ROI processing). The processing time is limited by the convolutional network module, which currently computes one output pixel at a time. Even faster feature extraction could be achieved by computing multiple output pixels in parallel. Assuming sequential processing of the 3 scales, tracking speeds reaching 150 fps can be expected, which exceeds the speeds achieved by existing hardware implementations.

5 Conclusion

In this paper, we presented a real-time FPGA implementation of deepDCF tracking algorithm. We evaluated the performance of the proposed solution and compared it with other similar approaches on the VOT2015 benchmark. The use of convolutional features in correlation filter-based object tracking offered an improvement in comparison to the often used HOG features. Next, we implemented the proposed method in a SoC FPGA device, which allowed us to take advantage of the computation parallelisation and quantisation. We also demonstrated that the models generated by the FINN compiler can be successfully used with one’s own design implemented in a hardware description language.

The used filter size of 64×64 offers higher accuracy (EAO 0.183 on the VOT2015 benchmark) than the algorithms implemented on FPGAs in the other articles discussed here while maintaining an average processing speed of 467.3 fps per scale. It is possible to select a larger filter, for example 112×112 to achieve even higher tracking quality but at the expense of processing speed. A lower FPGA clock could also be used to achieve lower power consumption depending on the particular application of the tracking system.

As part of future research, we plan to implement sequential processing of several scales or an application of the DSST filter. It is also worth investigating the possibility of using a nonlinear KCF filter by adding a kernel function computation to the existing implementation. We will also investigate the impact of the number of bits in the representation of the filter computation, as this potentially could reduce the FPGA resource consumption. The source code

of the implementation is available at https://github.com/mdanilow/MOSSE_fpga/tree/deep_features.

Acknowledgment. The work presented in this paper was supported by the National Science Centre project no. 2016/23/D/ST6/01389 entitled “The development of computing resources organisation in latest generation of heterogeneous reconfigurable devices enabling real-time processing of UHD/4K video stream” and AGH University of Science and Technology project no. 16.16.120.773.

References

1. Finn-hlslib. <https://github.com/Xilinx/finn-hlslib>
2. Kristan, M., et al.: The visual object tracking vot2015 challenge results. In: Visual Object Tracking Workshop 2015 at ICCV2015 (2015)
3. Blott, M., et al.: Finn-r: an end-to-end deep-learning framework for fast exploration of quantized neural networks. *ACM Trans. Reconfigurable Technol. Syst. (TRETS)* **11**(3), 1–23 (2018)
4. Bolme, D.S., Beveridge, J.R., Draper, B.A., Lui, Y.M.: Visual object tracking using adaptive correlation filters. In: 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 2544–2550 (2010). <https://doi.org/10.1109/CVPR.2010.5539960>
5. Chatfield, K., Simonyan, K., Vedaldi, A., Zisserman, A.: Return of the devil in the details: Delving deep into convolutional nets. *CoRR abs/1405.3531* (2014). <http://arxiv.org/abs/1405.3531>
6. Danelljan, M., Häger, G., Khan, F.S., Felsberg, M.: Convolutional features for correlation filter based visual tracking. In: 2015 IEEE International Conference on Computer Vision Workshop (ICCVW), pp. 621–629 (2015). <https://doi.org/10.1109/ICCVW.2015.84>
7. Danelljan, M., Häger, G., Khan, F.S., Felsberg, M.: Learning spatially regularized correlation filters for visual tracking. In: 2015 IEEE International Conference on Computer Vision (ICCV), pp. 4310–4318 (2015). <https://doi.org/10.1109/ICCV.2015.490>
8. Danelljan, M., Häger, G., Khan, F.S., Felsberg, M.: Discriminative scale space tracking (2016)
9. Danelljan, M., Häger, G., Shahbaz Khan, F., Felsberg, M.: Accurate scale estimation for robust visual tracking. In: Proceedings of the British Machine Vision Conference. BMVA Press (2014). <https://doi.org/10.5244/C.28.65>
10. Danelljan, M., Khan, F.S., Felsberg, M., Van De Weijer, J.: Adaptive color attributes for real-time visual tracking. In: 2014 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1090–1097 (2014). <https://doi.org/10.1109/CVPR.2014.143>
11. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: a large-scale hierarchical image database. In: *CVPR09* (2009)
12. Hahnle, M., Saxen, F., Hisung, M., Brunsmann, U., Doll, K.: Fpga-based real-time pedestrian detection on high-resolution images. In: 2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 629–635 (2013)
13. Henriques, J., Caseiro, R., Martins, P., Batista, J.: Exploiting the circulant structure of tracking-by-detection with kernels, vol. 7575, pp. 702–715 (2012). https://doi.org/10.1007/978-3-642-33765-9_50

14. Henriques, J.F., Caseiro, R., Martins, P., Batista, J.: High-speed tracking with kernelized correlation filters. *IEEE Trans. Pattern Anal. Mach. Intell.* **37**(3), 583–596 (2015). <https://doi.org/10.1109/TPAMI.2014.2345390>
15. Kowalczyk, M., Przewlocka, D., Kryjak, T.: Real-time implementation of adaptive correlation filter tracking for 4k video stream in zynq ultrascale+ mp soc. In: 2019 Conference on Design and Architectures for Signal and Image Processing (DASIP), pp. 53–58 (2019). <https://doi.org/10.1109/DASIP48288.2019.9049203>
16. Kristan, M., et al.: A novel performance evaluation methodology for single-target trackers. *IEEE Trans. Pattern Anal. Mach. Intell.* **38**(11), 2137–2155 (2016). <https://doi.org/10.1109/TPAMI.2016.2516982>
17. Liu, X., Ma, Z., Xie, M., Zhang, J., Feng, T.: Design and implementation of scale adaptive kernel correlation filtering algorithm based on hls. In: 2021 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC), pp. 1–5 (2021). <https://doi.org/10.1109/ICSPCC52875.2021.9564815>
18. Mohanty, R., Gonnabhaktula, A., Pradhan, T., Kabi, B., Routray, A.: Design and performance analysis of fixed-point jacobi svd algorithm on reconfigurable system. *IERI Procedia* **7**, 21–27 (2014). <https://doi.org/10.1016/j.ieri.2014.08.005>
19. Schölkopf, B., Smola, A.: Smola, A.: Learning with Kernels - Support Vector Machines, Regularization, Optimization and Beyond, vol. 98. MIT Press, Cambridge (2001)
20. Song, K., Yuan, C., Gao, P., Sun, Y.: Fpga-based acceleration system for visual tracking. *CoRR* abs/1810.05367 (2018). <http://arxiv.org/abs/1810.05367>
21. Umuroglu, Y., et al.: Finn: a framework for fast, scalable binarized neural network inference. In: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA 2017, pp. 65–74. ACM (2017)
22. Walid, W., Awais, M.U., Ahmed, A., Masera, G., Martina, M.: Real-time implementation of fast discriminative scale space tracking algorithm. *J. Real-time Image Process.*, 1–14 (2021)
23. Wu, Y., Lim, J., Yang, M.H.: Online object tracking: a benchmark. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2013)
24. Yang, H., Yu, J., Wang, S., Peng, X.: Design of airborne target tracking accelerator based on kcf. *J. Eng.* **2019** (2019). <https://doi.org/10.1049/joe.2018.9159>
25. Yang, K., Xie, M., An, J., Zhang, X., Su, H., Fu, X.: Correlation filter based uav tracking system on fpga. In: IET International Radar Conference (IET IRC 2020), vol. 2020, pp. 671–676 (2020). <https://doi.org/10.1049/icp.2021.0758>