Real-time implementation of adaptive correltaion filter tracking for 4K video stream in Zynq UltraScale+ MPSoC

ananymous
ananymous
ananymous
ananymous

***Abstract*—In this paper a hardware-software implementation of adaptive correlation filter tracking for a 3840 x 2160 @ 60 fps video stream in a Zynq UltraScale+ MPSoC is discussed. Correlation filters gained popularity in recent years because of their efficiency and good results in the VOT (Visual Object Tracking) challenge. An implementation of the MOSSE (Minimum Output Sum of Squared Error) algorithm is presented. It utilizes 2-dimensional FFT for computing correlation and updates filter coefficients in every frame. The initial filter coefficients are computed on the ARM processor in the PS (Processing System), while all other operations are preformed in PL (Programmable Logic). The presented architecture was described with the use of Verilog hardware description language.**

***Index Terms*—FPGA, Zynq SoC, image processing, object tracking, real-time processing, correlation filter, MOSSE**

I. Introduction

Object tracking is an important component of many vision systems. Example applications are tracking of: people in advanced video surveillance systems (AVSS), players (football, basketball, hockey) or objects (aircraft, drones, tanks) in military systems. Information about the object location in consecutive frames allows for a more complete analysis of its behaviour (eg. human action recognition). The first stage of each tracking algorithm is initialization (i.e. indication of the object or objects of interest). This is often done manually by selecting the area of interest with a bounding box. Another solution is to apply a detector – for example people or face – and use its output for tracking initialization.

In the aforementioned applications, key parameters are: execution time (usually real-time processing is preferable), energy efficiency and the ability to adapt the algorithm to different scenarios. Computing platforms that allow to meet these requirements to a large extent are reconfigurable FPGAs (Field Programmable Gate Arrays) devices, as well as programmable heterogeneous systems such as Zynq SoC (System on Chip) or Zynq UltraScale+ MPSoC (Multi-Processor System on Chip) from Xilinx. They enable high definition video stream processing (up to 4K or Ultra High Definition (UHD) – 3840×2160 @ 60 fps) in real time, are characterized by relatively low energy demand (especially in comparison to high-end general purpose processors or high-end programmable graphics processors), as well as the possibility of reconfiguration – which allows to easily improve the used algorithms (in contrast to ASIC (Application Specific Integrated Circuits) solutions). It is worth noting that these parameters are particularly important in the case of mobile platforms such as autonomous drones, where energy consumption translates directly into the operational possibility and real-time is important for tracking fast-moving objects (e.g. other drones).

In this paper the implementation of a correlation filter tracker based on the MOSSE (Minimum Output Sum of Squared Error [1]) algorithm in the Zynq UltraScale+ MPSoC system, which works in real time for a video stream of 3840×2160 @ 60 fps, is presented. The main contributions of this paper are :

- the first described in the literature implementation of the MOSSE method in reprogramming logic[1],
- a real-time tracking system for a video stream with 4K resolution in 4 ppc (pixel per clock) format,
- hardware verification of the presented solution.

The reminder of this paper is organized as follows. In Section II previous research related to object tracking is briefly discussed. In Section III the general correlation filter approach, as well as the MOSSE algorithm are presented. The proposed hardware-software implementation is described in Section IV and analysed, as well as compared with state-of-the art in Section V. The paper ends with a conclusion and possible further research directions.

II. Previous work

Due to the great practical significance, tracking is one of the main trends in computer vision. Over the years, many different methods have emerged and their detailed discussion goes beyond the scope of this paper. A review of older methods is presented in the work [2] and of more recent in the article [3]. Among the "classic" methods worth mentioning are: simple feature based tracking, KLT, MeanShift, CAMShift, Particle Filter and Kalman Filter. It should be noted that some of these

removed for blind review

[1]The authors of the article know about only one other implementation of correlation filters in FPGA. This will be more widely commented in Section V

methods are available in the Matlab package and the popular OpenCV image processing library. Among the "new" methods, two approaches dominate, which are also sometimes combined with each other: correlation filters (with various features, also obtained using deep convolutional neural networks) and deep neural networks (e.g. Siamese).

Hardware implementation of tracking algorithms in FPGA devices is discussed by many researchers. In a recent work [4] a system based on the DSST (Discriminative Scale Space Tracking) method was proposed, using the HOG (Histogram of Oriented Gradient) features. This approach is based on a correlation filter framework. The grayscale and HOG features are used to estimate the position, while HOG features for scale estimation. The solution works in real-time, processing 153 frame per second in HD resolution.

Other reported solutions are based on "classic" tracking methods. In the paper [5] the authors propose a system using the CAMShift algorithm, based on colour information in the HSV colourspace. In addition, they also implement the Kalman filter to estimate the movement of the object and verify the tracking results. The authors reported average processing speed of 309.91 frames per second for a HD image. In the paper [6] a two step approach was proposed. First motion detection is performed and then objects are classified using LBP (Local Binary Patterns) and HOG features in two scales (this is an example of tracking by detection). The system operates in real-time 60 fps and for a 640×480 resolution. In [7], the authors propose a modification of the KLT – MKLT (Motion-enhanced KLT) algorithm, adding information about the object's movement and filtering static points in this way. The hardware implementation processes about 33 fps in VGA resolution. In [8] the authors proposed a particle filter based object tracking algorithm, with parametrizable number of particles. The solution processes 650 fps of 320×240 resolution.

III. Correlation filter tracking – MOSSE

The idea of tracking with the use of correlation filters is very simple and intuitive. At the initialization stage, a certain part of the object is indicated – this is then called template or filter. Then, during the tracking, a correlation is made between the template and subsequent frames or parts of frames in the surrounding of the previous location – this is called a search window. In the last step, the maximum correlation response is found. It corresponds with the new location of the object.

However, the described approach has two major disadvantages. First of all, a single image of the object is used for initialization and the filter is not updated during tracking, which makes the solution prone even to minor changes in the appearance (shape, size, rotation). Secondly, it ensures a strong peak for the object, but it does not guarantee that it will be small for background patches. These problems determine the low usefulness of such an approach in practice.

Therefore, the described idea was further developed. Example solution are: UMACE (Unconstrained Minimum Average Correlation Energy) [9], ASEF (Average of Synthetic Exact Filters) [10] and MOSSE (Minimum Output Sum of Squared Error) [1]). These algorithms work similarly, but differ in the way of calculating and updating filter coefficients. Test results presented in the article [1] suggest that MOSSE is the most effective of these three filters. It should be noted that these methods are characterized by low computational complexity and good performance. For example, the analysis of the results of VOT (Video Object Tracking) 2017 ([11]) and 2018 ([12]) competitions shows that the latest methods based on the CF concept allow to achieve very good results. It is worth noting, however, that these approaches are usually a fusion of "classic" correlation filters and methods of feature extraction based on convolutional neural networks. Their implementation may be a possible next step of this work.

In order to ensure computational efficiency, the fast Fourier transform is used. It allows to replace the correlation operation in spatial domain with element-wise multiplication of two matrices in Fourier domain (Convolution Theorem). Let F be the Fourier transform of the image f, H the Fourier transform of the filter h, $\odot$ element-wise multiplication and $*$ the complex conjugate. Correlation is defined as:

$$G = F \odot H^* \tag{1}$$

The output G is converted back to spatial domain (inverse two-dimensional Fourier transform), where the maximum indicating the new location of the object is found. Also, to compensate for changes in the object appearance, the filter is updated. The general idea of correlation filter trackers is depicted in Figure 1.

In the following subsections, particular components of MOSSE are discussed. A broader description and justification is presented in paper [1]. In addition, implementations of MOSSE available on GitHub: in Matlab [13] and in Python [14] are very useful and were used in this work as a reference.

A. Preprocessing

In order to eliminate the so-called edge effect associated with the FFT transform and to improve the tracking stability, the following operations are applied during pre-processing: transforming the input image with a logarithmic function (greyscale image is used during tracking), normalization and multiplying by a 2D window (e.g. Hamming, cosine).

B. Initialization

The idea of the MOSSE algorithm is based on designing the filter h, so that for learning samples f_i, the desired answer g_i is obtained. The authors assumed that g_i has the form of a two-dimensional Gaussian distribution centred on the tracked target. The filter in Fourier domain is given by:

$$H_i^* = \frac{G_i}{F_i} \tag{2}$$

where division is performed element-wise.

In order to find the right filter H, the following optimization problem is solved:

$$\min_{H^*} = \sum_i |F_i \odot H^* - G_i| \tag{3}$$

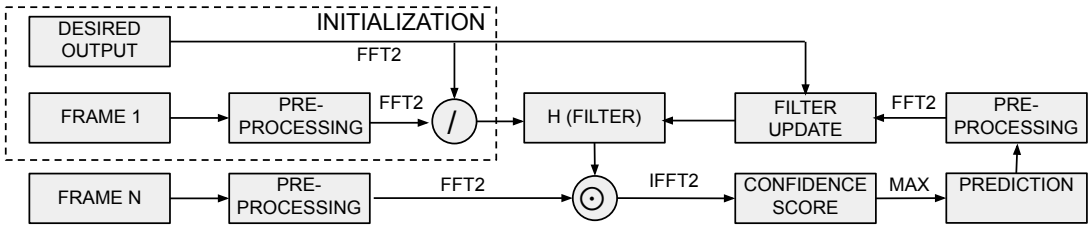

Fig. 1: Scheme of a correlation filter based tracker

The solution is in the following form (details in [1]):

$$H^* = \frac{\sum_i G_i \odot F_i*}{\sum_i F_i \odot F_i*} \tag{4}$$

It is worth noting that the nominator is the correlation between the input and the desired output. The denominator is the energy spectrum of the input.

In summary, the initialization procedure involves the following steps:

- selection of a ROI with the object,
- generating the reference filter response g (e.g. Gauss 2D),
- executing in a loop for $i \in 1..N$ iterations the following steps:
  - random affine transform of the ROI (to introduce minor disturbances – rotation, scaling and translation) – so subsequent f_i are created,
  - the same transform for g – g_i,
  - preprocessing of f_i and FFT calculation – F_i,
  - FFT calculation of g_i – G_i,
  - implementation of Equation (4).

C. Tracking

Tracking in the MOSSE method consists of the following steps:

- choosing a search window around object's position in previous frame,
- performing preprocessing and Fourier transform – F_k,
- convolving with the filter (in frequency domain) – $\hat{G}_k$,
- calculating inverse Fourier transform – $IFFT(\hat{G}_k)$,
- finding maximum of the correlation function, i.e. the new location of the object – $\max_{(x,y)} \hat{g}_k(x, y)$.

D. Filter parameters update

During tracking, the appearance of the object may change due to rotation, scaling, pose changes or different lighting conditions. Thus, it is advisable to update the parameters of the filter (H). In MOSSE the running average is used. The update for the k'th frame is given by the formula:

$$H_k^* = \frac{A_k}{B_k} \tag{5}$$

where A_k and B_k are define as (η – learning coefficient):

$$A_k^* = \eta(G \odot F_{k\ new}^* + (1-\eta)A_{k-1}) \tag{6}$$

$$B_k^* = \eta(F_{k\ new} \odot F_{k\ new}^* + (1-\eta)B_{k-1}) \tag{7}$$

It should be noted that $F_{k\ new}$ is a window (after preprocessing and FFT) centred on the **new** object's coordinates computed in the current tracking iteration (if the object moves $F_{k\ new} \neq F_k$).

IV. Hardware-software implementation of the MOSSE tracker

The MOSSE algorithm is divided into two modules: initialization and tracking together with filter update – c.f. Section III. When analysing the initialization procedure, it can be noticed that its hardware implementation would be cumbersome, due to iterative operation and the need to implement affine transformations. Therefore, in the current solution, it was decided to implement it in the processor system. Other operations were implemented in programmable logic (PL).

The system is designed for 4K resolution and the ZCU 104 board with Zynq UltraScale+ MPSoC XCZU7EV-2FFVC1156 from Xilinx. A general scheme is presented in Figure 2. The input is a 4K signal from a camera or computer's graphics card and the output a 4K LCD monitor. The processing system (PS) is responsible for initialization and overall control, while programmable logic (PL) for all other operations. It should be emphasized that due to the specific nature of the 4K signal, it is not possible to process it in the 1 pixel per clock (ppc) format. This issue is discussed in more detail in our previous paper [15]. Therefore, 2ppc or 4ppc have to be used for this resolution. In this work the second approach is applied.

The current version of the system supports a 64×64 tracking window. However, adapting the solution to power of 2 window sizes (not necessary square) is rather straightforward. For other sizes padding should be applied (due to FFT properties). Allowing other tracking window sizes will be part of our future work.

A. Initalization

The initialization procedure is implemented in the processor system (PS) of the Zynq US+ MPSoC (in the current version on one ARM core). The data from the programmable part (PL) is transferred via a two-port BRAM memory (`PATCH TO PS` in Fig. 2). In order to improve the implementation of operations such as scaling or rotation, a 96×96 pixels patch is saved (not 64×64). The PS reads data via an AXI BRAM Controller module.

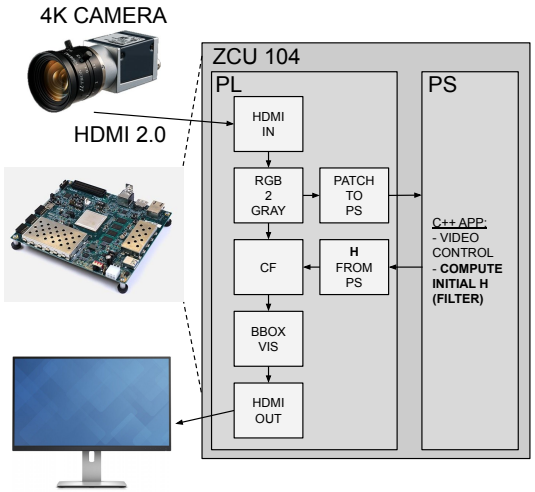

Fig. 2: Scheme of the proposed HW/SW tracking system

The steps described in Subsection III-B have been implemented in C ++ and added to the application prepared by Xilinx (video pass-through example). The KISS FFT [16] library was used to calculate the FFT. Compatibility was obtained with a Matlab reference model.

The calculated coefficients A (complex) and B (real) are converted to fixed-point format (sign and 63 fractional bits) and written to a dual-port BRAM via another AXI BRAM Controller (`H FROM PS` in Fig. 2). From there they are read by the tracking module.

B. Tracking and filter update

The scheme of the proposed module is shown in the Figures 3 and 4. It can be divided into two main parts. The first one is used to calculate the new location of the object based on correlation (tracking – red blocks) and the second to update the filter coefficients (update – blue blocks).

The inputs to the module are:

- *input stream* – 4K video input from a camera or graphic card in 4 pixel per clock format (AXI Stream is used),
- *rst* – system reset,
- *track* – flag that enables the changing position of the tracking window,
- *operate* – flag that allows to execute the algorithm,
- *update* – flag that allows to update filter coefficients.

In addition, modules common to both parts are: *Position counting* (determining position of the currently processed pixels), *Update global position* (object's position update between consecutive frames), *Initialize FFT (Divide /64)* (set configuration and scaling factors for FFTs), *Initialize IFFT* (set configuration and scaling factors for IFFTs).

Tracking: At the beginning, two modules (*xInRange* and *yInRange*) check that the currently processed pixels are in the window around the object's position in the previous frame. If so, the corresponding pixels are passed to the data stream *ROI Stream* and then used to calculate the correlation. The

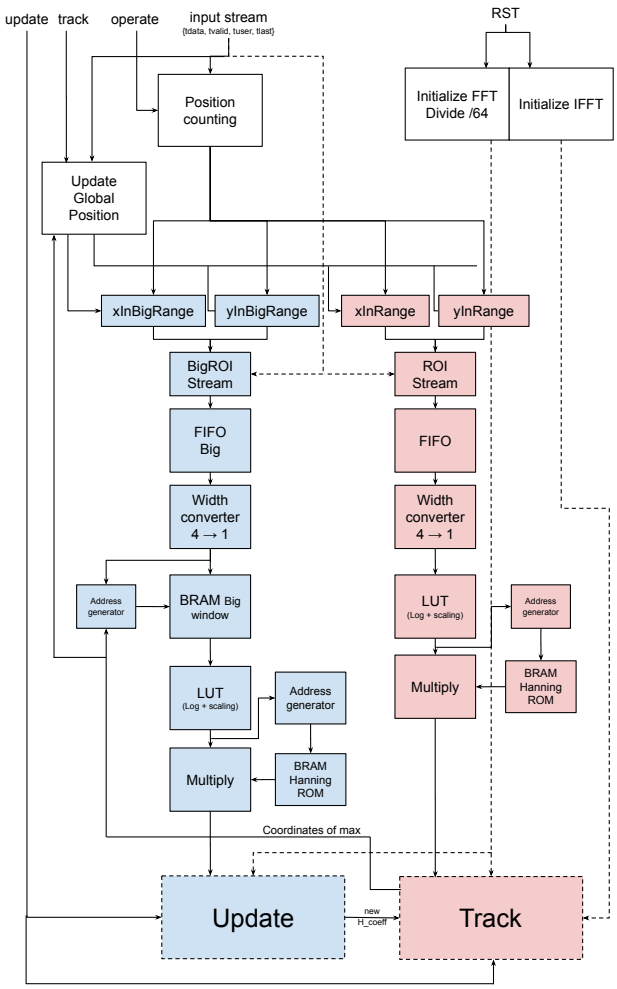

Fig. 3: Scheme of the proposed HW/SW tracking system (1)

stream is then converted from the 4 pixel per clock to 1 pixel per clock format (*Width converter 4 → 1*). To prevent data loss without increasing the frequency, a FIFO precedes the conversion module.

Next the image patch is pre-processed. Firstly, the appropriate value from the LUT is read. In this way, the applying of *log* function and scaling of input data is performed. Based on the number of pixels that have been pre-processed, the address for ROM containing the Hanning window coefficients (*BRAM Hanning ROM*) is calculated. Then the pixels after the LUT correction are multiplied by the appropriate window coefficient (*Mulitply*).

Data after pre-processing is sent to the input of the *FFT1* calculation module. An *IP Core* provided by Xilinx is used. This module calculates a one-dimensional Fourier transform for rows of the input image. The output is written to the *BRAM Transpose* memory, from which it is read by columns and sent to the next module calculating the one-dimensional Fourier transform (*FFT2*). Using the output data stream from the second FFT2 module, a corresponding address is generated to the memory *BRAM H_Coeff* containing the current filter

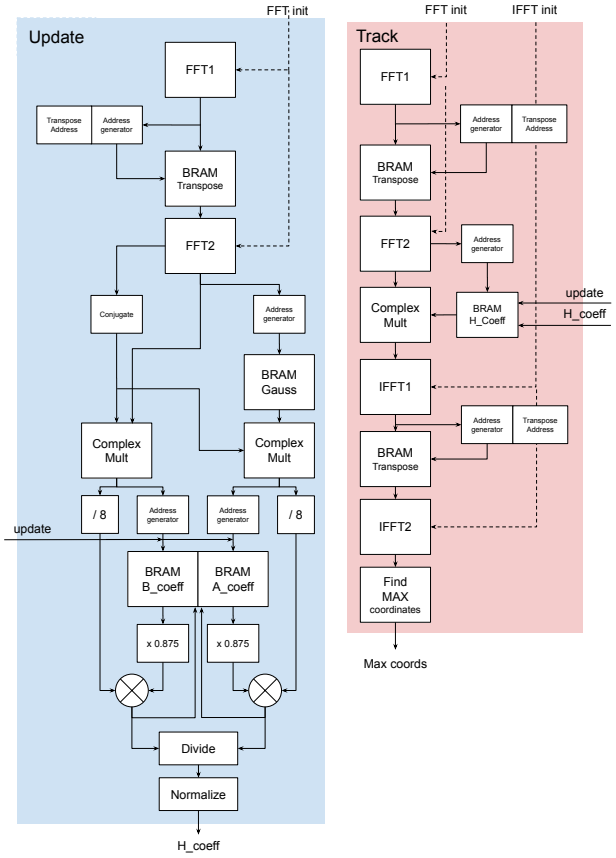

Fig. 4: Scheme of the proposed HW/SW tracking system (2)

coefficients in the frequency domain.

The Fourier transform of the window is multiplied element-wise with the coefficients read from memory (*Complex Mult*). This way, the correlation of the window with the filter in the Fourier domain is obtained. The multiplication result is then sent to the module calculating the one-dimensional inverse Fourier transform (*IFFT1*). The result is stored in the *BRAM Transpose* memory by columns and then read out by lines and sent to the next module calculating the one-dimensional inverse Fourier transform (*IFFT2*). In this way, the correlation of the window and filter in the spatial domain is obtained. Next, the location of the correlation maximum is determined. The computed coordinates are sent to the part responsible for updating the filter coefficients and the global location of the object being tracked.

Update of filter coefficients – update: At the beginning, it is checked whether the currently processed pixel is in a large window (twice the size of the correlation window) around the position of the object in the previous frame (modules *xInBigRange* and *yInBigRange*). If so, these pixels are passed to the data stream (*Big ROI Stream*), in which there are only pixels that will potentially be used to calculate the new filter coefficients. The stream is then converted from the 4 pixel per clock to 1 pixel per clock format (*Width converter* $4 \rightarrow 1$). To prevent data loss without increasing the frequency, a FIFO

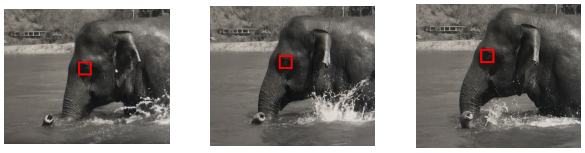

Fig. 5: Exemplary output from the implemented MOSSE algorithm.

was used. The data is then stored in the *BRAM Big window* memory.

When the calculation of the object's new position is completed, the pixel values in the window around this position are read and sent to the modules performing pre-processing and computing the Fourier transform in the same way as in the case of tracking (names of modules are analogous). The complex conjugate of the result of the transform – *Conjugate* – is also determined. Based on the data stream from the described part, an address is generated for the ROM memory containing Fourier transform coefficients of the Gaussian distribution (*BRAM Gauss*).

Then, two multiplications are performed. The first arguments are the result of the transform and its conjugate; the second are Gaussian coefficients in the frequency domain and the conjugate of the calculated Fourier transform (c.f. Equations (6) and (7)). Based on the results of multiplications, addresses to the BRAM memory containing the values of matrices A and B (*BRAM A_coeff* and *BRAM B_coeff*) are generated. These coefficients are read and multiplied by 0.875 using bit shift and subtraction (value taken from [1]). The multiplication results are then divided by 8 using a bit shift. The corresponding values are then added together and written back into the BRAM memory containing the A and B coefficients. The new values of coefficients A and B are divided, and the result is written in the BRAM memory with filter coefficients in the frequency domain (H) - c.f. Equation (5).

Computations of the FFTs are done with the use of 32-bit fixed-point precision. Computation of the new filter coefficients are done with the use of 64-bit fixed-point precision. This makes it possible to achieve calculation accuracy comparable to a double floating-point reference software model. The sum of absolute errors between both implementations was usually around 10^{-7}.

FPGA resource usage for the MOSSE module, as well as the whole system is given in Table II. Figure 5 presents sample output result of the described hardware implementation. Photos were taken from the real-time operation of the algorithm on the ZCU 104 board. The bounding boxes have been bolded. The elephant's eye was tracked, as the currently supported 64×64 window is small in comparison to the whole 4K frame. For the same reason the images have been significantly cropped.

TABLE I: Comparision of hardware implementations of tracking systems

	This work	[4]	[5]	[6]	[7]	[8]
Algorithm	MOSSE	DSST	CAMShift, Kalman	LBP, HOG	MKLT	Particle filter
Resolution	4K	HD	HD	VGA	VGA	320×240
Fps	60	153	309.9	60	33	650
Platform	Zynq US+	Kintex 7	Virtex 5	Virtex 4	Zynq 7020	Virtex 5

TABLE II: Resource utilization and power consumption for the FPGA part of the MOSSE tracker and complete system. All numbers were taken from the post place reports. The number in bracket shows % resource usage for the considered device.

Resource	MOSSE	Complete system
LUT	33453 (14.52%)	71315 (30.95%)
FF	67394 (14.63%)	111435 (24.18%)
BRAM	86.5 (27.72%)	92.5 (29.65%)
DSP	182 (10.53%)	205 (11.86%)
Power	-	8.839 W

V. Analysis and comparison with state-of-the art

Table I presents a comparison of selected hardware implementations of tracking algorithms (c.f. Section II) with the proposed MOSSE module. Due to the large discrepancy in the used methods (and their complexity), as well as the various generations of FPGAs, a direct comparison is not possible.

One exception is the work [4], which describes a tracking module using correlation filters – DSST [4] (according to the authors' knowledge – this is the only CF tracing implementation described in the literature). The solution presented there is theoretically more advanced than the proposed one, because in addition to pixel brightness, it also includes HOG features and performs tracking in many scales. Unfortunately, the article is very short (3 pages), of which the implementation description takes about 1 column of text (including three diagrams). Its analysis does not allow for a reliable assessment of this system. For example it is not clear how the filter is initialized, how the HOG features are calculated, how the filter parameters are updated and on what basis the 153 fps rate for the HD video stream was given. There was also no platform given on which the system was evaluated, but only the FPGA chip model for which it was designed. Due to the above, it is difficult to compare it with our solution.

It is also worth noticing a large discrepancy in the given *fps* parameters. Some authors provide a theoretical maximum frequency estimated on the basis of the maximum logic frequency and some real values (i.e. verified in hardware). In our opinion, the real value should be given as the basic one, and the estimation should be treated only as supplementary information.

VI. Summary

The article presents a hardware-software implementation of the MOSSE algorithm, which is an example of correlation filter based tracker. Thanks to the parallelization of calculations, real-time 3840×2160 @60 fps video stream processing has been achieved. The solution was verified in the hardware on the ZCU 104 evaluation board with the Xilinx Zynq UltraScale+ MPSoC device. The presented solution is the basis for further research on the hardware implementation of these very effective tracking algorithms. As a part of future work, the following are planned: optimization of the number of used FFT modules and BRAM memory, implementation of support for different window sizes, implementation of PSR (Peak to Sidelobe Ratio) calculation for tracking quality evaluation, adding additional features (HOG, calculated by a deep convolutional neural network), adding multiple scale tracking and analysis of the impact of calculation accuracy on resource usage, tracking performance and energy consumption. It is planned to use the module on an autonomous drone that could track, for example, another fast moving drone.

References

[1] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 2010, pp. 2544–2550.

[2] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *ACM Comput. Surv.*, vol. 38, no. 4, Dec. 2006. [Online]. Available: http://doi.acm.org/10.1145/1177352.1177355

[3] M. Fiaz, A. Mahmood, and S. K. Jung, "Tracking noisy targets: A review of recent object tracking approaches," *CoRR*, vol. abs/1802.03098, 2018. [Online]. Available: http://arxiv.org/abs/1802.03098

[4] K. Song, C. Yuan, P. Gao, and Y. Sun, "Fpga-based acceleration system for visual tracking," in *2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, Oct 2018, pp. 1–3.

[5] P. Gao, R. Yuan, Z. Lin, L. Zhang, and Y. Zhang, "A novel low-cost fpga-based real-time object tracking system," in *2017 IEEE 12th International Conference on ASIC (ASICON)*, Oct 2017, pp. 654–657.

[6] T. Sledevi, A. Serackis, and D. Plonis, "Fpga-based selected object tracking using lbp, hog and motion detection," in *2018 IEEE 6th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*, Nov 2018, pp. 1–5.

[7] W. Chen, Y. Ma, Z. Chai, M. Chen, and D. He, "An fpga-based real-time moving object tracking approach," in *Algorithms and Architectures for Parallel Processing*, S. Ibrahim, K.-K. R. Choo, Z. Yan, and W. Pedrycz, Eds. Cham: Springer International Publishing, 2017, pp. 65–80.

[8] P. Engineer, R. Velmurugan, and S. Patkar, "Parameterizable fpga framework for particle filter based object tracking in video," in *2015 28th International Conference on VLSI Design*, Jan 2015, pp. 35–40.

[9] A. Mahalanobis, B. V. K. V. Kumar, S. Song, S. R. F. Sims, and J. F. Epperson, "Unconstrained correlation filters," *Appl. Opt.*, vol. 33, no. 17, pp. 3751–3759, Jun 1994. [Online]. Available: http://ao.osa.org/abstract.cfm?URI=ao-33-17-3751

[10] D. S. Bolme, B. A. Draper, and J. R. Beveridge, "Average of synthetic exact filters," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, June 2009, pp. 2105–2112.

[11] M. Kristan, A. Leonardis, J. Matas, M. Felsberg, R. Pfugfelder, L. C. Zajc, and T. V. et al., "The visual object tracking vot2017 challenge results," 2017.

[12] M. Kristan, A. Leonardis, J. Matas, M. Felsberg, R. Pfugfelder, L. C. Zajc, T. Vojir, G. Bhat, A. Lukezic, A. Eldesokey, G. Fernandez, and et al., "The sixth visual object tracking vot2018 challenge results," 2018.

[13] A. Moudgil, "https://github.com/amoudgl/mosse-tracker," 2019.

[14] T. Dai, "https://github.com/tianhongdai/mosse-object-tracking," 2019.

[15] removed for blind review.

[16] M. Borgerding, "https://github.com/mborgerding/kissfft," 2019.