Accelerated Embedded AKAZE Feature Detection Algorithm on FPGA

Lester Kalms
Ruhr-Universität Bochum (RUB)
Bochum, Germany
lester.kalms@rub.de

Khaled Mohamed
German University in Cairo (GUC)
Cairo, Egypt
khaled.essam@student.guc.edu.eg

Diana Göhringer
Technische Universität Dresden
Dresden, Germany
diana.goehringer@tu-dresden.de

ABSTRACT

Feature detection is a major operation in various computer vision systems. The KAZE algorithm and its improved version, Accelerated-KAZE (AKAZE), are considered as the first algorithms to detect features by building a scale space using nonlinear diffusion. However, the detection part of the algorithm achieves only 6 fps on an Intel Core i7-4790 processor for an image resolution of 1024x768. This work proposes a pipelined architecture for a hardware accelerator that performs the AKAZE detection algorithm. Firstly, modifications are done to the original algorithm to reduce the amount of computations and memory accesses. Then, the accelerator is implemented on a Xilinx Zynq SoC and achieves 98 fps for the same resolution and a frequency of 100 MHz. Compared to the original algorithm, the design has an error in average inliers ratio by only 4%.

1. INTRODUCTION AND MOTIVATION

Detecting features in an image is considered as a fundamental step in many computer vision applications such as object recognition, image registration and motion tracking [15]. There are two basic criteria for choosing a detection algorithm. The first criterion is the ability of the algorithm to be used in widely varying applications, while the second one is the repeatability of the algorithm [15]. Repeatability is defined as the ability of detecting the same feature in different locations invariantly to rotation, scale, and change in brightness or contrast.

KAZE and accelerated KAZE (AKAZE), introduced in [2] and [3] respectively, are the first algorithms to use the non-linear diffusion in multi-scale feature detection. Other approaches use linear diffusion to create the scale-space. According to Alcantarilla et al. [3], both KAZE and AKAZE show better repeatability and performance than other algorithms such as ORB [19], BRISK [13], SIFT [14] and SURF [6].

The main problem with AKAZE is the computation intensiveness. For an image of resolution of 1024x768, we measure that the detection part of the original algorithm achieves only 6 fps in average on an Intel Core i7-4790 processor. In order to achieve higher frame rates and meet real-time requirements, this paper introduces a hardware acceleration of the detection part of the AKAZE algorithm. Additionally, the detected features (or key-points) are sent to the FREAK descriptor [1], which is processed in software and used instead of the MLDB descriptor of AKAZE. The FREAK descriptor is faster to compute and has a much lower memory load.

The paper is organized as follows: Section 3 shows a brief description of the AKAZE detection algorithm. Section 4 introduces the modifications done on the original algorithm. A detailed hardware implementation is described in section 5. In Section 6, the performance of the design and the hardware utilization are represented. Section 7 concludes this work and gives an outlook.

2. STATE OF THE ART

Various algorithms are introduced to detect image features in multi-scale framework. SIFT [14] and SURF [6] are the most widely used detectors in computer vision. Both are using the Gaussian (linear) diffusion to build the scale-space. Other algorithms, such as FAST [18], ORB [19] and BRISK [13], are proposed to reduce the cost of heavy computations. Meanwhile, all these algorithms, in addition to AKAZE, are not applicable for high frame rate especially for high resolution images. Hardware implementations are designed to accelerate those algorithms. A throughput of 30 fps is obtained using SIFT and SURF for VGA resolution (640x480) [8, 9, 10, 16]. An architecture based on ORB is designed in [12] producing 55 fps for VGA resolution. The highest throughput achieved is 127 fps for full HD images using optimized AKAZE algorithm [11]. One missing part in this paper is the computation of the contrast factor, which is essential in building the non-linear scale-space. Furthermore, the hardware testing methodology of this paper is very vague.

3. AKAZE FEATURE DETECTION

The AKAZE detection algorithm consists of 3 parts: computing the contrast factor, building the non-linear scale-space and detecting features. They are shown in the lower part of Figure 1. This section gives a summarized introduction to the different parts.

3.1 Computing the Contrast Factor

The contrast factor is significantly important in building the non-linear scale-space. Firstly, the image is smoothed by a

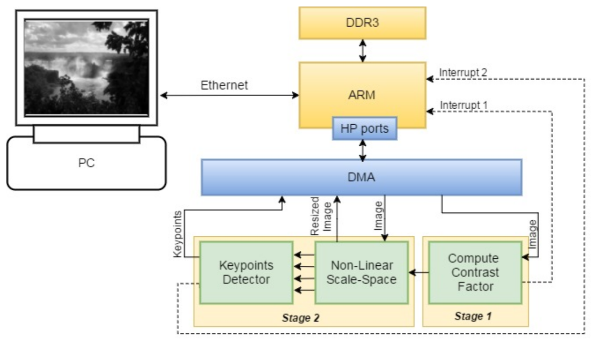


Figure 1. Block diagram of the complete system

Gaussian filter. Second step is to calculate the maximum absolute gradient value of the smoothed image, called h_{max}. This is done by looping on the whole image, calculating the absolute value of the gradient of each pixel. Third step is to fill a histogram of 300 bins with the gradient values divided by the parameter h_{max}. The following step is to loop on the histogram to find the histogram index "i" at which the 70% percentile of the gradient histogram is achieved. The contrast factor can then be calculated by :

$$k = \frac{h_{max} \cdot i}{300} \tag{1}$$

3.2 Building the Nonlinear Scale-Space

The scale-space levels are built by numerically solving the partial differential equation (Eq. 2) iteratively using the Fast Explicit Diffusion scheme (FED) with varying time steps τ_j (Eq. 3).

$$\frac{\partial L}{\partial t} = div(g(|\nabla L_\sigma|) \cdot \nabla L) \tag{2}$$

$$L^{j+1} = (I + \tau_j A(L^j))L^j \tag{3}$$

The function $g(|\nabla L_\sigma|)$ is called the conductivity function (or flow function) and $A(L)$ is the matrix notation of this function. The conductivity function is chosen to be:

$$g(|\nabla L_\sigma|) = \frac{1}{1 + \frac{|\nabla L_\sigma|^2}{k^2}} = \frac{k^2}{k^2 + |\nabla L_\sigma|^2} \tag{4}$$

where k is the contrast factor. To prevent using two division operations, Eq. 4 has been adjusted. For each pixel, computing the conductivity function (Eq. 4) requires 1 division, 2 multiplication and 2 addition operations. Additionally, each FED step (Eq. 3) needs 5 multiplication and 12 add/subtract operations for each pixel. That is what makes AKAZE computationally intensive and, consequently, time consuming. The scale-space in AKAZE (see Figure 2) is a pyramidal framework. It consists of octaves and each octave consists of sublevels. Each octave is quarter the size of the previous.

3.3 Feature Detector

AKAZE uses the determinant of the Hessian (DoH) blob-detector. After building the non-linear scale-space, DoH images are computed at increasing scale sizes as the sublevels increases. That is why AKAZE down-samples the images after certain sublevels.

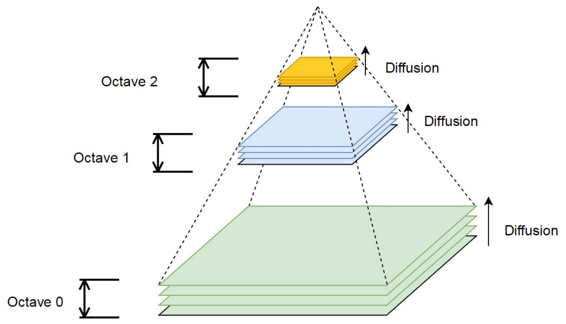


Figure 2. Scale-space representation in AKAZE (3 octaves, 4 sub-levels)

The keypoints (or features) are extracted by comparing the pixels in DoH images with a surrounding window of size 3x3. If the value of this pixel is greater than the other 8 surrounding pixels and a pre-defined threshold, the pixel is compared spatially with keypoints in the same sublevel in a window of radius σ^2 to exclude the repeated keypoints that exist inside the same circle. The same comparison is made with the upper and lower level and the radius depends on the sublevel.

4. MODIFIED DESIGN

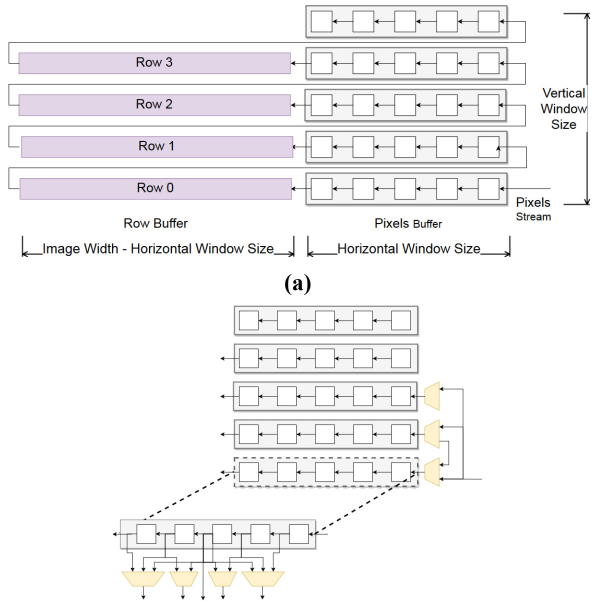
Four major modifications are made to the original implementation to reduce computations, memory access and array comparisons.

4.1 Fixed Point Implementation

The original algorithm is implemented in software using 32-bit floating point numbers. The implementation of floating point calculations is very costly in terms of hardware resources. So, the whole implementation of AKAZE is retyped to fixed point representation. To keep the same accuracy, all produced images, such as derivative, flow and smoothed images, are in 16-bit format except for the DoH images that are in 32-bit format.

4.2 Computing the Contrast Factor

The parameter h_{max} is used two times: the first time is when normalizing the gradient values to be fit into 300-bin histogram and another time to be multiplied by the ratio $i/300$ as in (Eq. 1). From measurements with different image and random input data, it is found that h_{max} does not exceed the value 0.5. The original implementation requires looping the whole image twice, one to find h_{max} and another to fill the histogram. Instead, in the modified design the image is looped only once. This can be done by normalizing the gradient values to 0.5 instead of h_{max}. The histogram length is not more than 300 bins and can take any arbitrary length. So, a new parameter $nbin_{max}$, which indicates the maximum length that is achieved, is taken into consideration. Now, only one loop is needed to fill the histogram with the new normalized gradient values and to



(b)

Figure 3. Block diagram of 5x5 window generator

calculate $nbin_{max}$ and h_{max} for the next step. The next step is to compute the contrast factor by (Eq. 6):

$$k = \frac{h_{max} \cdot i}{nbin_{max}} \tag{6}$$

This new method doubles the speed of computing the contrast factor and reduces the memory access by half. Another change is to approximate the square root operation, for computing the gradient, into one division, one multiplication and one summation, as in (Eq. 7), where L_{max} and L_{min} are the maximum and minimum values of the first derivatives L_x and L_y respectively.

$$\nabla L = \sqrt{{L_x}^2 + {L_y}^2} \approx |L_{max}| + \frac{1}{2}\frac{L_{min} \cdot L_{min}}{|L_{max}|} \tag{7}$$

Additionally, the contrast factor is redefined to be 68.75% percentile instead of 70%. This replaces multiplication by 0.7 to be multiplication by 0.6875, or in other words multiplication by (1/2 +1/8 +1/16), which can be achieved by adding and shifting instead of multiplying.

4.3 Keypoint Comparison

Each detected keypoint in AKAZE is compared to all other keypoints in the keypoint buffer, to avoid replication of neighboring keypoints. This method has a complexity of $O(n^2)$ for n keypoints, which makes it very computationally intensive. The modified design takes the advantage that the keypoints are detected in ascending order with respect to the vertical position. So, adaptive searching pointers are used to determine the beginning and ending of the keypoints to be compared in the buffer. This adaptive searching decreases the complexity dramatically from $O(n^2)$ to be approximately $O(n \cdot log(n))$. If the keypoints are only compared to the lower and the same level, the number of matches differs by about 1%. So, comparing with the upper level is excluded.

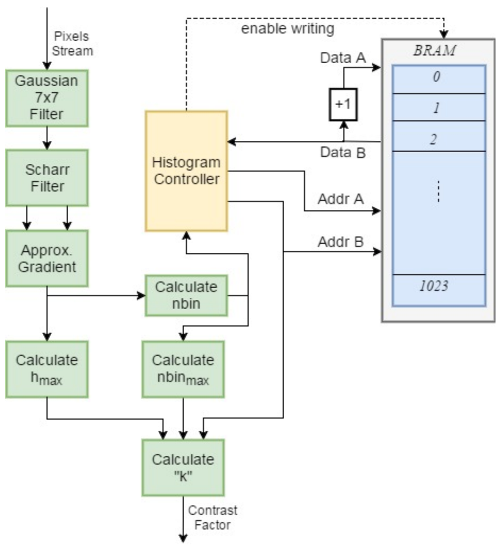


Figure 4. Block diagram of stage 1 of the accelerator

4.4 FREAK Descriptor

Fast Retina Keypoint (FREAK) is a descriptor that is inspired by the human visual system and more precisely the retina [1]. In our experiments, we have compared the MLDB descriptor of AKAZE with other descriptors, like BRIEF [7], BRISK or FREAK. The FREAK descriptor is in general faster to compute and has a lower memory load and is more robust than the other algorithms. The greatest advantage over the MLDB descriptor is the low memory load. Since the MLDB descriptor needs the computed images and derivatives of all levels in scale-space to be stored into memory, which is critical in an embedded system with limited memory resources and bandwidth. Our software tests have shown that the FREAK descriptor achieves comparable or better results than other descriptors in terms of correct matches for AKAZE.

5. HARDWARE IMPLEMENTATION

This section describes different hardware blocks used to build the AKAZE accelerator on an FPGA, in addition to the software used to test the accelerator.

5.1 System Overview

The system consists of three parts (see Figure 1): PC platform, ARM processor and the hardware accelerator. Both ARM and accelerator are embedded into the Zynq-7000 chip on the ZedBoard [5]. The PC platform contains the software that reads the images, sends them to the ARM via Ethernet and then receives the keypoints. It also contains the description and matching parts that use these keypoints to test the system.

The accelerator has two pipelined stages: stage 1 computes the contrast factor and stage 2 builds the non-linear scale space and detects the keypoints. The ARM sends an image (first frame) to stage 1, and when it finishes, it sends an

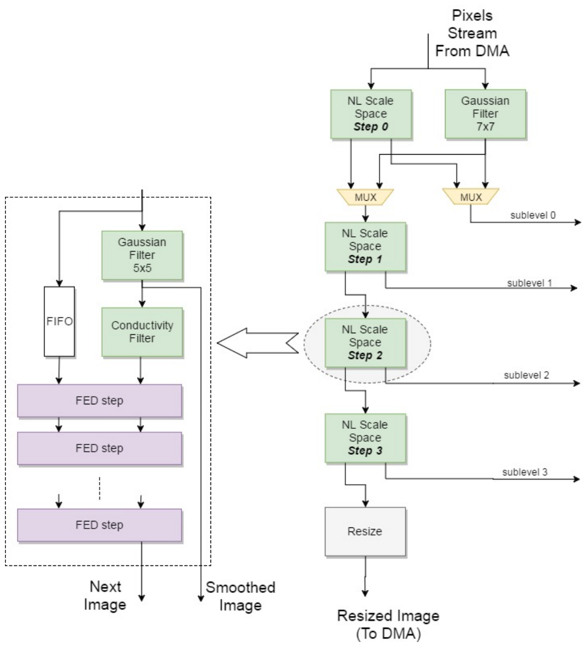


Figure 5. Non-linear scale-space generation

interrupt to the ARM to send the same image to stage 2. The contrast factor of an image is computed only once for a frame. Stage 2 has two outputs that are saved in memory: the keypoints of the first octave and the resized image for the following octave. When stage 2 finishes, it sends an interrupt to the ARM to send the resized image back from the memory to stage 2 to detect the keypoints for the second octave. This process is repeated until all keypoints of different octaves are sent to the ARM. While stage 2 is working on the first frame, stage 1 computes the contrast factor for the second frame. So, both stages are working simultaneously but for different frames.

5.2 Window Generator

Most modules in the implemented hardware design must scan the whole image and use a specific number of pixels as a window to generate its result. The implemented modules use different window sizes ranging from 3x3 to 17x17. An extended architecture of the window generator, like the one proposed in [17], is used. The window generator (see Figure 3a) consists of pixels buffer (on the right) of shift registers and row buffer (on the left), built with BRAM. The output of the window generator is the content of the pixels buffer. In the extended design (see Figure 3b), some multiplexers are added to the pixels buffer to support horizontal and vertical replicated borders. The extended design also supports variable image widths to be used by different octaves.

5.3 Mathematical Operations

In the AKAZE hardware design, there are five mathematical operations that use the output of the window generator: Gaussian, Scharr derivative, conductivity function, FED and determinant of Hessian. Each function

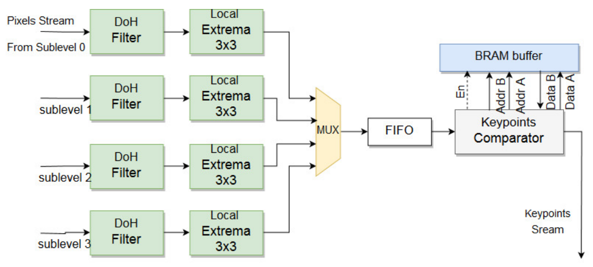


Figure 6. Block diagram of the keypoints detector

of them uses a different size of the window generator according to the kernel size and the scale.

5.4 Contrast Factor

A block diagram of computing the contrast factor is shown in Figure 4. The histogram is a simple dual port BRAM (A for writing and B for reading) of length 1024. The histogram controller is a Finite State Machine (FSM) that has three states: filling histogram, reading histogram and halt. For the first state, the controller enables writing and sets both addresses A and B to the index value *nbin* so that the value at address *nbin* is incremented by one. In the second state, the controller reads the histogram values (through address B) starting from the first index until 68.75% percentile of the gradient histogram is achieved. Then, the contrast factor is calculated by (Eq. 6), where $i = Addr\ B$.

5.5 Non-Linear Scale-Space

The hardware is designed for building 4 sublevels per octave. Each sublevel is generated by a "NL Scale-Space" block. This block consists of (see Figure 5): Gaussian Filter, conductivity function generator and a number of FED steps. Because the pixel stream of the flow image (generated by the conductivity filter) and the original image must be presented simultaneously, a FIFO is needed before the first FED step to compensate the delay of the window generator of the Gaussian and conductivity filters. The number of FED steps varies according to the sublevel and octave. In sublevel 0 for octave 0, there are no FED steps. So, this sublevel is built by only a Gaussian filter with window size 7x7 to remove the noise of the original image. Therefore, multiplexers are used in octave 0 to choose between this Gaussian filter and the normal "NL Scale-Space step 0" block.

5.6 Keypoint Detector

The DoH filter is applied on each input image with a different scale, which depends on the sublevel (see Figure 6). Taking the advantage of the linearity property of the convolution process, the second order derivatives can be computed using the second order derivative kernel without computing the first order derivative images. The second order derivative kernel is calculated by convoluting the two derivative kernels. If more than one keypoint is present at the same time from different "local extrema" blocks, the multiplexer chooses the keypoint with the lower sublevel

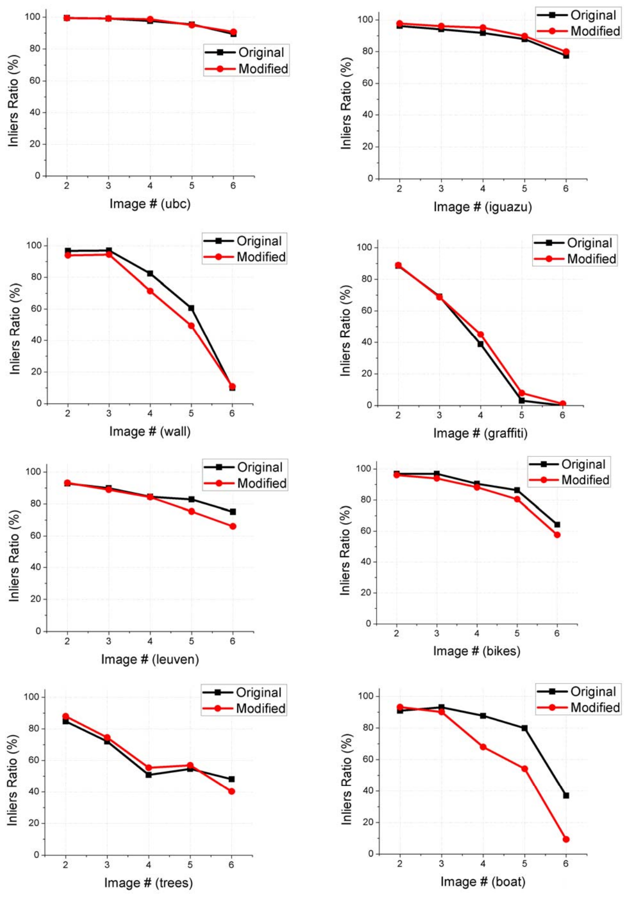


Figure 7. Inliers ratio for different image sets

first. New keypoints are compared to the keypoints stored in the buffer by the adaptive search pointers. The keypoint is a 64-bit vector, which contains the x and y position, the sublevel number and the response value (DoH value). According to the measurements, the buffer length is set to a maximum of 8192 keypoints.

6. RESULTS AND UTILIZATION

6.1 Detection Repeatability

A set of 48 images is used to test the repeatability of the system. The images are in grey-scale and differs in blurring, viewpoint, rotation, brightness, JPEG compression and noise added. They are taken from the Oxford Visual Geometry Group datasets [4] and the “iguazu” dataset presented by Alcantarilla et al. [3]. The original implementation of Alcantarilla et al. [3] has been taken for comparison and compiled using the –O2 optimization flag. The different configuration flags only differ in the contrast factor as described in section 4.2.

Figure 7 shows a comparison between the original algorithm and the modified one for the mentioned datasets with respect to the inliers ratio. That is, the ratio between correct matches to total matches. Generally, our modified algorithm shows the same behavior as the original one for different images. The original algorithm gives 76% average inliers ratio and ours gives 73% for the test datasets. Consequently, an error of only 4% in the average inliers ratio is present. The reduced ratio in some datasets is mainly due to the omitted sub-pixel refinement when using the FREAK descriptor, which can be added without increasing the computation time.

Table 1. System utilization

Block		FF	LUT	BRAM	DSP
Gaussian	5x5	626	720	2	7
	7x7	593	843	3	10
Conductivity		1248	966	1	2
FED		641	327	2	5
DoH	Level 0	1829	1452	4	2
	Level 1	3279	1601	6	2
	Level 2	3279	1598	6	2
	Level 3	5249	1942	8	2
Local Extrema		352	365	2	0
Resize Image		175	187	0.5	0
Contrast Factor		2932	2704	4.5	13
NL Scale-Space		19806	13803	55.5	136
Keypoints Detection		15576	8438	48	8
Total System		38314 (36%)	24945 (47%)	108 (77%)	157 (71%)

6.2 System Throughput and Resources

The hardware is synthesized and implemented using Xilinx Vivado 2016.4 and the ZedBoard [5] as the evaluation and development board. The implemented design gives a maximum operating frequency of 100 MHz. The throughput of the design depends on the size of the input frame. Eq. 7 is used to calculate the time of the system:

$$T_{tot} = \frac{4}{3}W(H+n)(1-4^{-O})/freq \qquad (8)$$

Where W is the image width, H is the image height, O is the number of octaves and n is a factor representing the initial delay required to fill the row buffers in the window generators. The parameter n is estimated to be 4 for stage 1 and 25 for stage 2 (see Figure 1). By measuring the computation time of the accelerator using the ARM processor, it is found that (Eq. 8) is completely accurate. In our system, we are using an image resolution of 1024×768 and 2 octaves. Stage 2 determines the overall computation time, since O is only 1 in stage 1. So, the accelerator needs a total time of 10.15 ms or 98 fps.

Table 1 shows resource utilization for basic building blocks and the overall system as given by the implementation tool. The results show that it is possible to fit AKAZE feature detection algorithm on a low-cost FPGA board for embedded systems, with remaining resources for light weighted a descriptor like FREAK.

7. CONCLUSION AND FUTURE WORK

This paper introduces a real-time hardware implementation for the AKAZE detection algorithm. The design consists of two parallel stages embedded on an FPGA connected with an ARM processor. The system is evaluated and tested on a ZedBoard and runs on a frequency of 100 MHz. For a resolution of 1024 × 768, the system gives a throughput of 98 fps while keeping the same accuracy as the original algorithm implemented on software. In future work, the FREAK descriptor can be implemented in hardware and integrated in parallel to the AKAZE accelerator as a third stage. Moreover, using the sub-pixel refinement to reallocate the keypoints shows better repeatability results. The sub-pixel refinement uses the DoH images, so this can be implemented in parallel to the local extrema block.

8. ACKNOWLEDGMENT

This work is part of the TULIPP project, which is funded by the European Commission under the H2020 Framework Programme for Research and Innovation under grant agreement No 688403.

9. REFERENCES

[1] Alahi, A., Ortiz, R., Vandergheynst, P. 2012. Freak: Fast retina keypoint. In *Computer Vision and Pattern Recognition (CVPR)* (Jun. 2012), 510-517. DOI= http://dx.doi.org/10.1109/cvpr.2012.6247715.

[2] Alcantarilla, P. F., Bartoli, A., Davison, A. J. 2012. KAZE features. In *European Conference on Computer Vision* (Oct. 2012), 214-227. DOI= http://dx.doi.org/10.1007/978-3-642-33783-3_16.

[3] Alcantarilla, P. F., Solutions, T. 2013. Fast explicit diffusion for accelerated features in nonlinear scale spaces. In *Proceedings of the British Machine Vision Conference 2013* (2013), 13.1-13.11. DOI= http://dx.doi.org/10.5244/C.27.13.

[4] Anon. Visual Geometry Group Home Page. Retrieved February 20, 2017 from http://www.robots.ox.ac.uk/~vgg/data/data-aff.html.

[5] Anon. ZedBoard HW Users Guide. Retrieved February 20, 2017 from http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf.

[6] Bay, H., Tuytelaars, T., Van Gool, L. 2006. Surf: Speeded up robust features. In *European Conference on Computer Vision* (May 2006), 404-417. DOI= http://dx.doi.org/10.1007/11744023_32.

[7] Calonder, M., Lepetit, V., Strecha, C., Fua, P. 2010. Brief: Binary robust independent elementary features. *European Conference on Computer Vision* (Sep. 2010), 778-792. Springer Berlin Heidelberg. DOI= http://dx.doi.org/10.1007/978-3-642-15561-1_56.

[8] Chiu, L. C., Chang, T. S., Chen, J. Y., Chang, N. Y. C. 2013. Fast SIFT design for real-time visual feature extraction. *IEEE Transactions on Image Processing* 22, 8 (Aug. 2013), 3158-3167. DOI= http://dx.doi.org/10.1109/tip.2013.2259841.

[9] Huang, F. C., Huang, S. Y., Ker, J. W., Chen, Y. C. 2012. High-performance SIFT hardware accelerator for real-time image feature extraction. *IEEE Transactions on Circuits and Systems for Video Technology 22*, 3 (Mar. 2012), 340-351. DOI= http://dx.doi.org/10.1109/tcsvt.2011.2162760.

[10] Jeon, D., Henry, M. B., Kim, Y., Lee, I., Zhang, Z., Blaauw, D., Sylvester, D. 2014. An energy efficient full-frame feature extraction accelerator with shift-latch FIFO in 28 nm CMOS. *IEEE Journal of Solid-State Circuits* 49, 5 (May 2014), 1271-1284. DOI= http://dx.doi.org/10.1109/jssc.2014.2309692.

[11] Jiang, G., Liu, L., Zhu, W., Yin, S., Wei, S. 2015. A 127 fps in full HD accelerator based on optimized AKAZE with efficiency and effectiveness for image feature extraction. *Proceedings of the 52nd Annual Design Automation Conference (DAC)* (Jun. 2015), 1-6. DOI= http://dx.doi.org/10.1145/2744769.2744772.

[12] Lee, K. Y., Byun, K. J. 2013. A hardware design of optimized orb algorithm with reduced hardware cost. *Advanced Science and Technology Letters Volume 43* (Dec. 2013), 58-62. DOI= http://dx.doi.org/10.14257/astl.2013.43.11.

[13] Leutenegger, S., Chli, M., Siegwart, R. Y. 2011. BRISK: Binary robust invariant scalable keypoints. *2011 IEEE International Conference on Computer Vision (ICCV)* (Nov. 2011), 2548-2555. DOI= http://dx.doi.org/10.1109/iccv.2011.6126542.

[14] Lowe, D. G. 2004. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision* 60, 2 (Nov. 2004), 91-110. DOI= http://dx.doi.org/10.1023/b:visi.0000029664.99615.94.

[15] Manjunath, B. S., Shekhar, C. and Chellappa, R. 1996. A new approach to image feature detection with applications. *Pattern Recognition* 29, 4 (Apr. 1996), 627-640. DOI= http://dx.doi.org/10.1016/0031-3203(95)00115-8.

[16] Na, E. S., Jeong, Y. J. 2013. FPGA Implementation of SURF-based Feature extraction and Descriptor generation. *Journal of Korea Multimedia Society* 16, 4 (Apr. 2013), 483-492. DOI= http://dx.doi.org/10.9717/kmms.2013.16.4.483.

[17] Rettkowski, J., Boutros, A., Göhringer, D. 2015. Real-time pedestrian detection on a xilinx zynq using the HOG algorithm. *ReConFigurable Computing and FPGAs (ReConFig)* (Dec. 2015), 1-8. DOI= http://dx.doi.org/10.1109/reconfig.2015.7393339.

[18] Rosten, E., Drummond, T. 2005. Fusing points and lines for high performance tracking. *Tenth IEEE International Conference on Computer Vision (ICCV05) Volume 2* (Oct. 2005), 1508-1515. DOI= http://dx.doi.org/10.1109/iccv.2005.104.

[19] Rublee, E., Rabaud, V., Konolige, K., Bradski, G. 2011. ORB: An efficient alternative to SIFT or SURF. *2011 IEEE International Conference on Computer Vision (ICCV)* (Nov. 2011), 2564-2571. DOI= http://dx.doi.org/10.1109/iccv.2011.6126544.