SAFE: a Self Adaptive Frame Enhancer FPGA-based IP-core for real-time space applications

SAFE: a Self Adaptive Frame Enhancer FPGA-based IP-core for real-time space applications

Stefano Di Carlo*, Giulio Gambardella*, Piergiorgio Lanza†, Paolo Prinetto*, Daniele Rolfo*, Pascal Trotta*
*Politecnico di Torino
Dipartimento di Automatica e Informatica
Corso Duca degli Abruzzi 24, I-10129, Torino, Italy
Email: {name.familyname}@polito.it
†Thales Alenia Space Italia S.p.a.
Strada Antica di Collegno 253, I-10146 Torino, Italy
Email: {name.familyname}@thalesaleniaspace.com

***Abstract*—Video-based navigation is an increasingly used procedure with hard real-time requirements and high computational effort. In this field, FPGA hardware acceleration supplies low-cost and considerable performances enhancement. Video-based navigation algorithms extrapolate and correlate features from images, relying on their accuracy. Image enhancement provides more defined and contrasted frames, assuring high precision feature extraction. The paper introduces an FPGA-based self-adaptive image enhancer. The IP-core is suitable for hard-real time applications, such as space applications, thanks to the guaranteed high-throughput.**

I. Introduction

Video-based navigation (VBN) is a well-studied area of computer vision, spanning many application fields including robotics [1], unmanned vehicles [2], automotive [3] and medical [4]. VBN is also gaining increased importance in space-applications. Future space-missions will increasingly adopt VBN systems to assist the Entry, Descent and Landing (EDL) phase of space modules (e.g., spacecrafts), thus enhancing the precision of automatic EDL navigation systems.

The information required to estimate the motion of the vehicle are extracted from consecutive images. For each image special *features* (e.g., edges, corners) are extracted. Then, the *matching* task correlates the features extracted between two consecutive images to estimate the relative motion.

The *Feature Extraction and Matching* (FEM) must be performed at a very high speed, in order to guarantee a high frame rate. This task is often accelerated by hardware modules to satisfy hard real-time requirements. These modules are increasingly implemented resorting to FPGA devices, exploiting FPGA flexibility and very low Non Recurrent Engineering (NRE) costs. This trend is true for both commercial and mission critical applications, such as aerospace [5].

When employed in peculiar environments, VBN systems require ad-hoc methodologies to aware the EDL system from noise and unknown conditions (i.e., illumination intensity and light direction). In fact, in these situations the FEM algorithms could produce fake results that may led to a reduced motion estimation accuracy, or even, in the worst case, destructive and irreversible events.

To overcome such problems, image enhancement algorithms are required before feeding the FEM module with the acquired frames from the on-board camera.

Image enhancement processes include a collection of techniques that improve the appearance of an image. The image is modified to achieve a better suited image for subsequent analysis (i.e., FEM).

Image enhancement can be performed in *intensity*, *spatial* or *frequency* domains. Among the available techniques, the ones that better improve FEM algorithms are those working in the intensity domain.

Histogram Equalization and *Histogram Stretching* [6] proved to be two of the most effective Image Enhancement Techniques (IET).

An image histogram is a graphical representation of the tonal distribution in a digital image. It plots the number of pixels in the image (vertical axis) that present a particular intensity value (horizontal axis).

Histogram Equalization (HE), in particular Linear HE, changes the intensity value of each pixel to produce a new image with a more uniform image histogram (i.e., the image covers most of the brightness dynamic range). A better distributed histogram increases the image contrast, especially if the original image has close intensity values. The method is useful in images with backgrounds and foregrounds that are both bright or both dark, since these images are characterized by narrow and smoothed histograms.

The Histogram Stretching (HS) technique is based on redistribution of the pixel intensities to spread their values on the entire spectrum of colors. It increases the contrast among pixels, but it becomes uneffective when the input image features a wide histogram.

Both HE and HS are based on the statistics of the entire image. They require a high computational effort, since repetitive operations must be performed on each pixel. For a hardware implementation, after the image is analyzed, a complete buffering of the image is needed to apply the transformation, leading to a significant increase of memory occupation and latency (see Section II).

This work proposes SAFE: a Self Adaptive Frame Enhancer FPGA-based IP-core for real-time applications. It presents a novel self adaptive algorithm which avoids the need of image buffering and local storing.
SAFE avoids the internal buffering of the image, because the enhancement is performed on the basis of tone statistics gathered from the previous image (i.e., histogram associated to the previous processed frame). This approach introduces a negligible error (see Section IV) thanks to the limited differences between two consecutive frames, guaranteed by the high frame rate reachable by SAFE. This high throughput is achieved exploiting a highly parallelized internal architecture. Moreover, unlike previous works, SAFE is able to autonomously select the best IET to be applied. This significantly improves its applicability to several different environmental conditions.
Thanks to its high throughput, SAFE is a suitable option to enhance image quality and to improve the performances of the FEM algorithms, without impacting the overall system latency.
The paper is organized as follows: in section II the actual state-of-the-art is analyzed; in section III the proposed mathematical algorithm and the SAFE internal architecture are described; in section IV experimental results are reported and, in section V, conclusions are drawn.

II. Related Work

Many FPGA implementations of image enhancement algorithms have been proposed: [7] [8] [9] [10].
In [7] brightness control, contrast adjustment and histogram equalization FPGA implementations are presented. Low area occupation is shown but the complete image (100x100 pixels) has to be internally stored.
In [8] a very efficient real-time HE technique is presented. The smart transformation function is implemented thanks to 256 16-bit counters and a enabling hierarchical decoder. This approach clearly increases the resource usage. Furthermore, the complete image (256x256 pixels) is stored in a 64 kb ROM in the FPGA device.
[9] presents an Adaptive Histogram Equalization (AHE) algorithm: a method for the local contrast enhancement based on bilinear interpolation. The area occupation of the proposed AHE is considerable and the noise in background homogeneous regions is amplified.
[10] presents an implementation of Contrast Limited Adaptive Histogram Equalization (CLAHE) that limits the noise increment provided by AHE. The drawback is that a great amount of internal memory is required to store data for the bilinear interpolation.

III. SAFE Architecture

SAFE is a highly parallelized FPGA-based IP core that receives in input a 1024x1024 grey scale image (e.g., from a CMOS camera) and outputs the enhanced image. SAFE has a standard 32-bit input interface, making it compliant with the most common bus interfaces. Input pixels are received as a set of 32-bit packets, without any header or padding bit. Thus, four pixels can be received in parallel from a standard camera with 8 bit-per-pixel (bpp) resolution. Also, SAFE receives in input two parameters: *HW* and *BW*. *HW* defines the threshold associated with the histogram width (i.e., the distance between the minimum and maximum intensity inside the image histogram). *BW* defines the threshold referred to the difference between two consecutive image histogram bar (HB) values. These two parameters are required for automatically selecting the best frame enhancement technique (i.e., HS or HE), depending on the input image statistics.
SAFE is composed of three main blocks (Fig. 1): the *Histogram Calculator*, the *Histogram Analyzer* and the *Equalizer / Stretcher*.

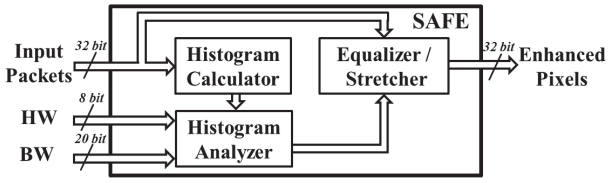


Figure 1: SAFE block diagram

The *Histogram Calculator* computes the histogram of the input image, providing the value of each bar. It simply counts the occurrence of each pixel intensity, in order to compute the bar values.
The *Histogram Analyzer* analyzes the image histogram in order to select the best IET to be applied. It scans the histogram to find the minimum and maximum intensities and the maximum difference between two consecutive bar values. By comparing this two quantities with the HW and BW thresholds, it selects the best IET (see Alg. 1).
The *Equalizer / Stretcher* performs both HE and HS on the input image, but it provides in output the image enhanced by the algorithm selected by the *Histogram Analyzer*.
For the sake of completeness, in the sequel, the mathematical operations required by the two enhancement algorithms are summarized.
The HS transformation function is reported in (1).

$$I_{Stretched}(x,y) = C \cdot (I(x,y) - min\_HB) \tag{1}$$

where $I_{Stretched}(x,y)$ is the stretched pixel intensity in the (x,y) position, $I(x,y)$ is the pixel intensity in the (x,y) position, min_HB is the minimum intensity in the previous image histogram, and C is a scale factor equal to:

$$C = \frac{2^{bpp} - 1}{max\_HB - min\_HB} \tag{2}$$

In (2) max_HB is the maximum intensity value in the previous image histogram and *bpp* is the bit-per-pixel resolution. The transformation function performed during HE is:

$$I_{Equalized}(x,y) = k \cdot \sum_{j=0}^{I(x,y)} HB_j \tag{3}$$

Algorithm 1 Histogram Analyzer operations

$found \leftarrow FALSE$
$max\_BW \leftarrow 0$
$previous\_HB \leftarrow HB(0)$
for $i = 0 \rightarrow 255$ **do**
  if $(HB(i) \neq 0) \wedge (found = FALSE)$ **then**
    $min\_HB \leftarrow HB(i)$
    $found \leftarrow TRUE$
  end if
  $actual\_BW \leftarrow \mid HB(i) - previous\_HB \mid$
  if $actual\_BW > max\_BW$ **then**
    $max\_BW \leftarrow actual\_BW$
  end if
  $previous\_HB \leftarrow HB(i)$
end for
$found \leftarrow FALSE$
for $((i = 255 \rightarrow 0) \wedge (find = FALSE))$ **do**
  if $HB(i) \neq 0$ **then**
    $max\_HB \leftarrow HB(i)$
    $found \leftarrow TRUE$
  end if
end for
$real\_HW \leftarrow max\_HB - min\_HB$
if $(real\_HW < HW) \wedge (max\_BW > BW)$ **then**
  $IET \leftarrow HS$
else
  $IET \leftarrow HE$
end if

where $I_{Equalized}(x, y)$ is the equalized pixel intensity in the (x, y) position, HB_j is the value of the j-th HB and k is equal to:

$$k = \frac{2^{bpp} - 1}{1024 * 1024} \tag{4}$$

In the following subsections all the implementation details of the SAFE modules are deeply analyzed.

A. Histogram Calculator

The *Histogram Calculator* module gets in input the pixels of the image through a 32-bit interface. Since each pixel of the received image is represented by 8 bits, 4 pixels can be analyzed in parallel, to speed up the computation task.
This module outputs the values associated with the HBs, that are then buffered in the next module (see Section III-B).
The parallel internal architecture of the *Histogram Calculator* is shown in Fig. 2.
It is composed of 4 *BRAM Buffers*, 4 20-bits adders, a 4 inputs 20-bits adder, 4 2-to-1 8 bits multiplexers, 4 2-to-1 20 bits multiplexers and a *Controller* that manages the overall histogram computation process.
The histogram computation is performed in two steps. First, the *Controller* sets to zero the *reset* signal. In this way input pixels act as addressing signal for the 4 *BRAM Buffers*. Buffers are implemented as dual-port Block RAMs, provided by Xilinx FPGA Virtex architectures [11]. Each BRAM Buffer has a

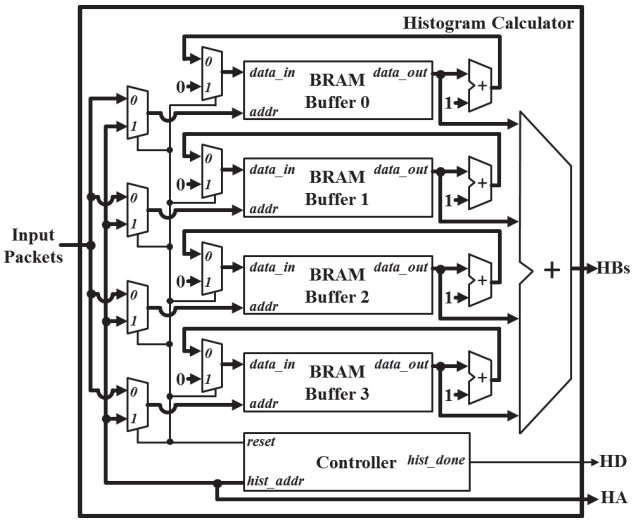


Figure 2: Histogram Calculator internal architecture

dimension of 512 rows x 32-bit.
The 32-bit input packets are split into 4 8-bit words, each representing a pixel value. Each received pixel addresses the *BRAM Buffer* associated with its position in the input packet (e.g., the pixel in the least significant byte of the input packet addresses the *BRAM Buffer 0*). The value of the location addressed by the input pixel value is read, incremented, and then rewritten in the same location in a single clock cycle exploiting the dual-port nature of the buffer. In this way, each buffer row acts as a counter.
When an entire image is received, each buffer row contains a partial HB value.
In the second computation step, the partial HBs are merged to compute the image histogram. *BRAM Buffers* are scanned starting from location 0. At each clock cycle, the 4 partial HB values are read and summed. In this way, the final value of each HB is computed. After each location has been read, it is forced to 0. This ensures that counters are resetted to the initial conditions, allowing the computation of a new histogram.
In addition to HB values, the *Histogram Calculator* outputs the *HA* and the *HD* signals. The former represents the index associated with the output HB value, while the latter is asserted when the histogram calculation task is completed.
The first computation step performed by the *Histogram Calculator* requires the number of clock cycles needed to receive 1024x1024 pixels, while the second step requires only 256 clock cycles to read the buffers. Thus, the total number of clock cycles required by the *Histogram Calculator* for computing the image histogram counts up to:

$$N_{clock} = \frac{1024 * 1024}{4} + 256 = 262400 \tag{5}$$

B. Histogram Analyzer

The *Histogram Analyzer* receives in input *HBs*, *HA* and *HD* signals from the *Histogram Calculator* module. In addition, it receives the two SAFE input thresholds: *BW* and *HW*. This

module extracts, from the image histogram associated with the previous frame, all the information required to calculate and select the best IET to be applied to the current frame. The internal architecture of the *Histogram Analyzer* module is shown in Fig. 3.

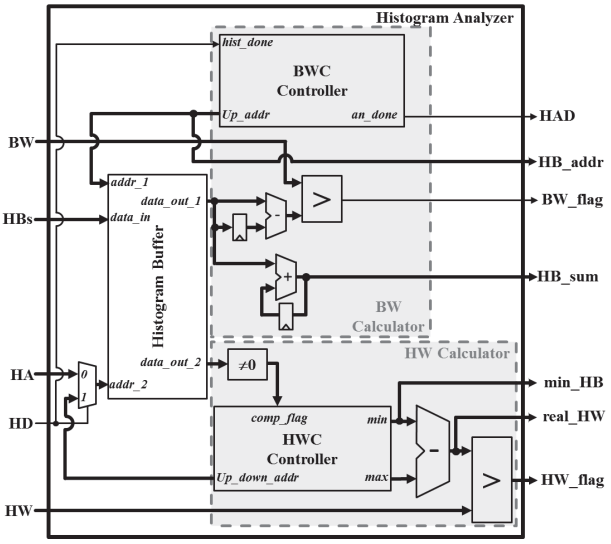


Figure 3: Histogram Analyzer internal architecture

The module is composed of three main components: the *Histogram Buffer*, the *BW Calculator* and the *HW Calculator*. The *Histogram Buffer* is implemented using a BRAM, required to store the complete histogram. Each HB received from the previous module is stored into a BRAM row. During this phase, *HD* is 0, thus *HA* acts as an address signal for the *data_in* port.

When the entire histogram is stored, the *HD* signal is asserted and the histogram analysis can start. During this phase, *HW Calculator* and *BW Calculator* work in parallel.

The *HW Calculator* performs two tasks: it finds the minimum and the maximum intensity values in the image histogram and it computes the histogram width. First, it scans the *Histogram Buffer* to find the first non-zero value, that represents the minimum intensity value (i.e., *min_HB*). Then, the same operation is repeated in the reverse scanning order, to find the maximum intensity value. Minimum and maximum values are then subtracted to calculate the histogram width (i.e., *real_HW*). This quantity is then compared with *HW* and, if it is greater, the *HW_flag* signal is asserted. The comparing task is performed exploiting an 8-bit comparator, and the entire process is managed by the *HWC Controller*.

The *BW Calculator* scans the *Histogram Buffer* in order to find the maximum difference between two adjacent HBs. This quantity is compared with the *BW* threshold and, if it is greater, the *BW_flag* signal is asserted.

In parallel to this task, *BW Calculator* computes che *HB_sum*, that represents the value of the sum of the equalization transformation function (i.e., $\sum_{j=0}^{I(x,y)} HB_j$ in (3)).

In order to perform these tasks, *BW Calculator* is composed of a 20-bit subtractor, a 28-bit accumulator, a 20-bit comparator and a Finite State Machine (i.e., *BWC Controller*). Finally, the *BWC Controller* asserts the *HAD* signal when the histogram scanning is completed. The entire histogram analysis process requires just 256 clock cycles, since the histogram is composed of 2^8 HBs (i.e., pixel intensities can range from 0 to 255).

C. Equalizer / Stretcher

The *Equalizer / Stretcher* module receives in input, from the *Histogram Analyzer*, all the signals required to perform the enhancement of the input image, and it provides the enhanced pixels, grouped in 32-bit packets. The internal architecture is shown in Fig. 4.

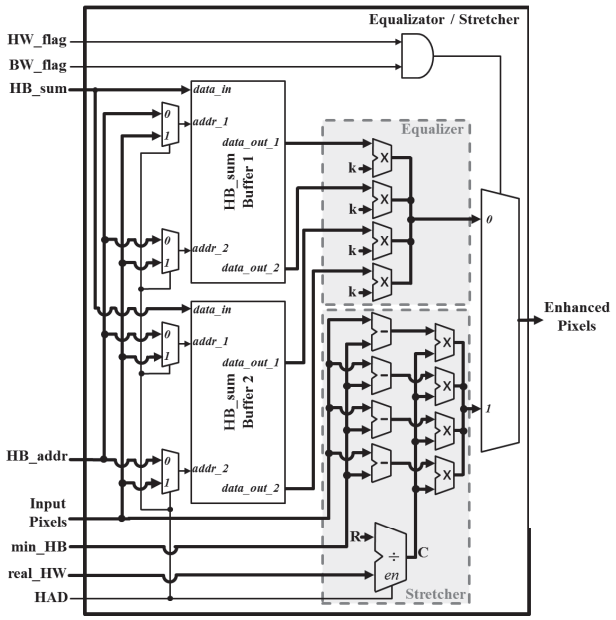


Figure 4: Histogram Equalizer and Stretcher internal architecture

This module is composed of two *HB_sum Buffers*, the *Equalizer* and the *Stretcher* modules.

Initially, the *HAD* signal is 0, thus, the *HB_sum* values provided by the previous module are stored both in *HB_sum Buffer 1* and in *HB_sum Buffer 2*, that are implemented as dual-port BRAMs.

When the *HAD* signal is asserted, a new image can be received and enhanced. The equalization and stretching of the input pixels are done in parallel by the *Equalizer* and the *Stretcher*. *Equalizer* applies the transformation function shown in (3) on each input. The 4 input pixels address the 4 ports provided by the two buffers in order to read out the associated sum value (i.e., $\sum_{j=0}^{I(x,y)} HB_j$ in (3)). The 4 values are then multiplied by the k constant (see (3)). The k constant is represented using the 0.15 fixed-point format, this ensures a very high precision (see Section IV). For the sake of completeness, in the proposed

architecture, the binary value of k is set to:

$$k = \frac{2^{bpp} - 1}{1024 * 1024} = .000000000001000_{(b)} \quad (6)$$

This choice enables to implement the multiplications as shifting operations, providing low resources usage. The result of the multiplication is truncated in order to extract its integer part, that represents the value of the equalized pixel.

Stretcher performs the operation shown in (1). First, it computes the subtraction between the received pixel values and the *min_HB* signal. In parallel, the *C* factor is computed by mean of a 23-bit division between $2^{bpp} - 1$ (i.e., *R* in Fig. 4) and *real_HW* signal. 23 bits are required in order to represent the output of this division in the 8.15 fixed-point format.

Finally, the results of the subtractions and the *C* factor are multiplied exploiting 4 23-bit multipliers. The result of each multiplication is truncated in order to extract the integer part that represents the value of the stretched pixel.

The output of the *Equalizer / Stretcher* module is selected between the outputs of the *Equalizer* or the *Stretcher* using a multiplexer. The select signal of this multiplexer is driven by the logic-and operator between *HW_flag* and *BW_flag*. In this way, the stretched pixels are output if *HW_flag* and *BW_flag* are asserted (i.e., the image histogram is narrow and very peaked), otherwise equalized pixels are provided in output.

The *Equalizer / Stretcher*, for enhancing the input image, requires the number of clock cycles needed to receive 1024x1024 pixels, plus 23 additional clock cycles to perform the *C* factor computation, because the divider is not a combinatorial component. Thus, the overall number of clock cycles required to enhance an image is equal to:

$$N_{clock} = \frac{1024 * 1024}{4} + 23 = 262167 \quad (7)$$

IV. Experimental results

The architecture presented in Section III has been described in VHDL and synthesized on a space qualified Virtex 5 XQR5VFX130 FPGA device, using Xilinx ISE Design Suite 14.3. The selected FPGA provides exceptional hardness to single-event upsets, total immunity to single-event latch-ups, total ionizing doses, and datapath protection from single-event transients, making it completely suitable for space applications.

Table I lists the area occupation on the target device, highlighting the resources usage of every SAFE building block.

Table I: Resource Usage for *Xilinx Virtex 5 XQR5VFX130*

Module	FPGA Area Occupation			Max Freq.
	LUTs	DSP	BRAMs	[MHz]
HC	248 (0.19%)	- (-)	5 (0.84%)	253.3
HA	181 (0.14%)	- (-)	2 (0.34%)	172.4
E/S	250 (0.19%)	8 (2.5%)	- (-)	177.1
Total	679 (0.52%)	8 (2.5%)	7 (1.18%)	

The maximum input frame rate that can be sustained by SAFE is related to the number of clock cycles needed to analyze and enhance the input images. The time slot required between the acquisition of two consecutive frames is equal to the time required to compute the input image histogram, to analyze it, and to calculate the scale factor *C* for the image stretching. Since the maximum operating frequency is equal to 172.4 MHz, the time required to accomplish these tasks is:

$$T_{exec} = \frac{N_{CLK\_HC} + N_{CLK\_HA} + N_{CLK\_DIV}}{f_{MAX}} = \\ = \frac{262400 + 256 + 23}{172.4\ MHz} = 1.524\ ms \quad (8)$$

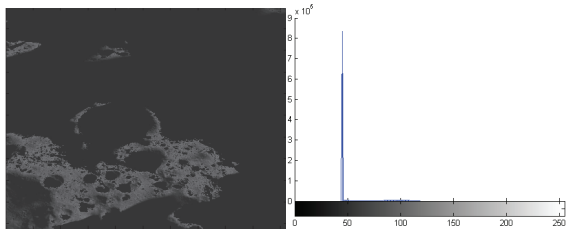
This execution time led to a maximum frame rate of 656 frame-per-second (fps).

This result clearly shows that, in an images processing chain, the image enhancement far exceeds the typical frame rate of 25 fps of the other building blocks, and of the camera, as well.

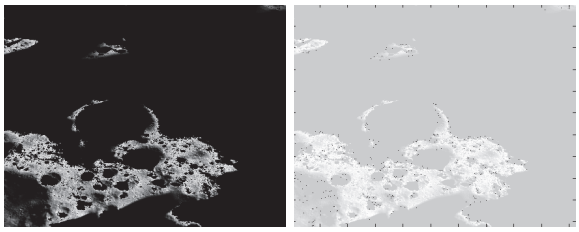
Simulations of the hardware have been done using Modelsim SE 10.0c. Simulation results have been validated comparing them with the values obtained by a MATLAB model of the described architecture. The validation involved a set of 50 different images of several planetary environment provided by *Thales Alenia Space* company.

Fig. 5 and Fig. 6 show examples of frames enhanced by the two available algorithms. In Fig. 5 the best result is achieved by applying the HS, while in Fig. 6 the HE should be preferred. Starting from the original frame shown in Fig. 5a, the relative histogram is computed. As shown in Fig. 5b, in this case the histogram is narrow and extremely peaked. Thus, a HS approach should be preferred, since it outputs an image with high contrast (see Fig. 5c), with respect to the equalized one (see Fig. 5d).

By analyzing the original frame shown in Fig. 6a and the associated histogram (Fig. 6b), one can notice that the histogram

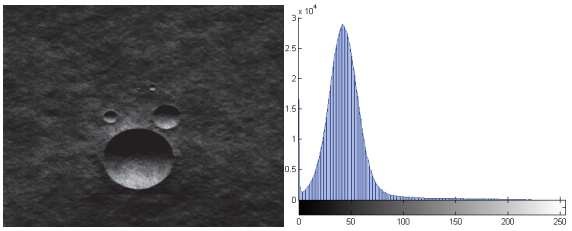


(a) Original (b) Histogram

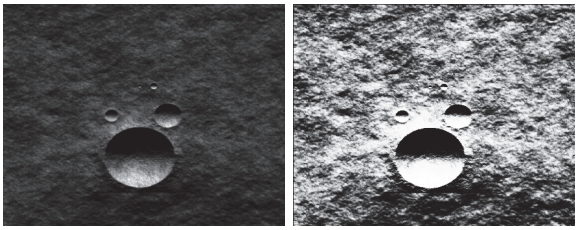


(c) Stretched (d) Equalized

Figure 5: Test of frame enhancement (best results by HS)



(a) Original (b) Histogram



(c) Stretched (d) Equalized

Figure 6: Test of frame enhancement (best results by HE)

is smoothed and relatively wide. Thus, the HE will give better results in term of image contrast, as can be inferred by visually comparing Fig. 6d and Fig. 6c.

After tuning *HW* and *BW* thresholds, the validation campaign has demonstrated that SAFE is always able to select the best IET to be applied on the input frames. Moreover, it has also proven that the frame enhancement achieved by the proposed architecture is completely satisfying. In fact, the statistics (i.e., image histograms) of two consecutive images are quite closed, even if they are taken by the camera at 1 fps. Comparisons have been done between frames enhanced with SAFE (i.e., exploiting the information associated to the previous frame), and frames enhanced with a standard approach (i.e., exploiting the information of the associated histogram). This comparison showed that SAFE introduces just a maximum error of 0,39% on the intensity of the pixels, w.r.t. the standard approach.

V. Conclusion

This paper presented SAFE: an innovative FPGA-based IP-CORE for real-time applications. Thanks to its self adaptivity, SAFE is able to enhance the frames captured by a camera without storing internally the whole image. Moreover, depending on the properties of the image, it automatically chooses the best enhancement operation to be applied.

The experimental results show how the different algorithms can heighten the contrast, in order to improve the subsequent feature extraction and matching task. The very high frame rate achieved by the IP-core allows its use in hard real-time application. Furthermore, the area occupation of the whole design is negligible when compared to the total amount of resources available in the FPGA device.

Acknowledgment

The authors would like to express their sincere thanks to the whole Software Unit design team of Thales Alenia Space Italia S.p.a. for their helpful hints and guidelines.

References

[1] L. Brewster, R. Howard, A. Johnston, C. Carrington, J. Mitchell, and S. Cryan, "Multi-sensor testing for automated rendezvous and docking sensor testing at the flight robotics lab," in *Proc. of 2008 IEEE Aerospace Conference*, pp. 1–7, 2008.

[2] M. Mohamed, S. Patra, and A. Lanzon, "Designing simple indoor navigation system for uavs," in *Proc. of 19th Mediterranean Conference on Control Automation (MED)*, pp. 1223–1228, 2011.

[3] H. Wu, Y. Liu, and J. Rong, "A lane detection approach for driver assistance," in *Proc. of 1st International Conference on Information Engineering and Computer Science (ICIECS)*, pp. 1–4, 2009.

[4] D. Mirota, H. Wang, R. Taylor, M. Ishii, G. Gallia, and G. Hager, "A system for video-based navigation for endoscopic endonasal skull base surgery," *IEEE Transactions on Medical Imaging*, vol. 31, pp. 963–976, 2012.

[5] S. Habinc, "Suitability of reprogrammable FPGAs in space applications - feasibility report," tech. rep., Gaisler Research, 2002.

[6] R. Lakshmanan, M. Nair, M. Wilscy, and R. Tatavarti, "Automatic contrast enhancement for low contrast images: A comparison of recent histogram based techniques," in *Proc. of 1st International Conference on Computer Science and Information Technology (ICCSIT)*, pp. 269–276, 2008.

[7] S. Sowmya and R. Paily, "FPGA implementation of image enhancement algorithms," in *Proc. of 2011 International Conference on Communications and Signal Processing (ICCSP)*, pp. 584–588, 2011.

[8] A. Alsuwailem and S. Alshebeili, "A new approach for real-time histogram equalization using FPGA," in *Proc. of 2005 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*, pp. 397–400, 2005.

[9] K. Kokufuta and T. Maruyama, "Real-time processing of local contrast enhancement on FPGA," in *Proc. of 19th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 288–293, 2009.

[10] P. Ferguson, T. Arslan, A. Erdogan, and A. Parmley, "Evaluation of contrast limited adaptive histogram equalization (CLAHE) enhancement on a FPGA," in *Proc. of 8th IEEE International SoC Conference*, pp. 119–122, 2008.

[11] Xilinx Corporation, *Virtex-5 FPGA User Guide - UG190 v5.4*, 2012.