POLITECNICO DI TORINO
Repository ISTITUZIONALE

SA-FEMIP: A Self-Adaptive Features Extractor and Matcher IP-Core Based on Partially Reconfigurable FPGAs for Space Applications

Original
SA-FEMIP: A Self-Adaptive Features Extractor and Matcher IP-Core Based on Partially Reconfigurable FPGAs for Space Applications / DI CARLO, Stefano; Gambardella, Giulio; Prinetto, Paolo Ernesto; Rolfo, Daniele; Trotta, Pascal. - In: IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS. - ISSN 1063-8210. - STAMPA. - 23:10(2015), pp. 2198-2208. [10.1109/TVLSI.2014.2357181]

Availability:
This version is available at: 11583/2571938 since: 2015-11-16T15:58:59Z

Publisher:
IEEE / Institute of Electrical and Electronics Engineers

Published
DOI:10.1109/TVLSI.2014.2357181

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

12 October 2024

SA-FEMIP: a Self-Adaptive Features Extractor and Matcher IP-core based on Partially Reconfigurable FPGAs for Space Applications

Stefano Di Carlo, *Senior Member, IEEE,* Giulio Gambardella, *Student Member, IEEE,* Paolo Prinetto, *Senior Member, IEEE,* Daniele Rolfo, *Student Member, IEEE,* Pascal Trotta, *Student Member, IEEE.*

***Abstract*—Video-Based Navigation (VBN) is increasingly used in space-applications to enable autonomous Entry, Descent and Landing of aircrafts. VBN algorithms require real-time performances and high computational capabilities, especially to perform Features Extraction and Matching (FEM). In this context, Field-Programmable Gate Arrays (FPGAs) can be employed as efficient hardware accelerators. This paper proposes an improved FPGA based FEM module. On-line self-adaptation of the parameters of both the image noise filter and the features extraction algorithm is adopted to improve the algorithm robustness. Experimental results demonstrate the effectiveness of the proposed self-adaptive module. It introduces a marginal resource overhead and no timing performance degradation when compared to the reference state-of-the-art architecture.**

***Index Terms*—Video-based navigation, image processing, FPGA, hardware acceleration, space applications.**

I. Introduction

UPCOMING plans for solar system exploration in the next 30 years are expected to include landing, sample and return missions to moons, planets, asteroids, and comets [1]. In recent space missions Spirit, Opportunity, and Curiosity, the spacecraft descending trajectory and the final landing point were precomputed and fixed during the mission planning, enabling to reach a maximum landing precision, quantified in terms of area in which the spacecraft is likely to land (landing ellipse) of 20 km [2]. Spacecraft autonomous precision landing capabilities able to reduce the landing ellipse to sub-kilometer accuracy would provide safe and affordable access to landing sites that promise the highest science return and pose minimal risk to the spacecraft [3]. Video-Based Navigation (VBN) is an area of computer vision that exploits image frames captured by cameras and image processing algorithms to assist navigation in several application domains, including robotics, unmanned vehicles, and avionics [4] [5]. The wide availability of cameras on spacecrafts makes VBN a very interesting approach for the implementation of autonomous Entry, Descent and Landing (EDL) control systems for next generation space missions.

VBN algorithms extract geometrical information from a set of real-time sampled image frames. They basically perform two activities named *Feature Extraction and Matching* (FEM), and *Motion Estimation* (ME). During FEM, each frame is processed to detect those pixels that represent *features* of interest for the image (e.g., corners or edges of surfaces). The detected features are then compared to extract those that can be recognized in two consecutive images (*matching points*). Eventually, the ME algorithms analyze the detected matching points and estimate the relative position and orientation of the camera (fixed with respect to the moving object). To increase accuracy, ME algorithms require very accurate matching points distributed across the entire frame [6]. While ME algorithms are not computationally intensive, FEM algorithms require high computation capability to guarantee high frame rates and therefore high accuracy. Hence, very efficient hardware accelerators for this task are mandatory.

This paper proposes SA-FEMIP, an optimized FPGA-based self-adaptive FEM architecture based on the well known Harris feature extractor algorithm [7]. This architecture extends the solution proposed by the authors in [8] by enabling self-adaptation of the parameters of the image noise filter and the feature extraction algorithm. Self-adaptation enables to better optimize the FEM algorithm to the environmental conditions, thus increasing the robustness with respect to noise and variations of external conditions that are typical of the space environment. Adaptation is obtained introducing very marginal overhead and guaranteeing high operational rates. This is achieved by resorting to the Dynamic Partial Reconfiguration (DPR) capabilities of modern space-qualified FPGAs. Experimental results clearly show that SA-FEMIP enables increased accuracy and performance compared to previous architectures, two key characteristics for the implementation of VBN systems for next generation space missions.

The rest of the paper is organized as follows: Section II overviews related works, while Sections III, IV and V introduce the proposed architecture and its main improvements and peculiarities. Section VI shows the results obtained from an extensive experimental campaign and, finally, Section VII summarizes the main contributions and concludes the paper.

II. Related Works

Feature extraction is the most complex activity performed by FEM algorithms. Several feature extraction algorithms have been proposed in the literature (e.g., Beaudet [9], SUSAN [10], Harris [7], SURF [11] and SIFT [12]). From the algorithmic point of view, SURF and SIFT are probably the most robust solutions since they are scale- and rotation-invariant. This

S. Di Carlo, G. Gambardella, P. Prinetto, D. Rolfo, P. Trotta are with the Department of Control and Computer Engineering, Politecnico di Torino, Corso Duca degli Abruzzi 24, I-10129 Torino, Italy. E-mail: {stefano.dicarlo, giulio.gambardella, paolo.prinetto, daniele.rolfo, pascal.trotta}@polito.it.

means that features can be matched between two consecutive frames even if they have differences in terms of scale and/or rotation. However, due to their complexity, hardware implementations are very resource hungry. As an example, [13] and [14] propose two FPGA-based implementations of the SURF algorithm. The architecture proposed in [13] consumes almost 100% of the LUTs available on a medium sized Xilinx Virtex 6 FPGA, without guaranteeing real-time performances. Similarly, the architecture proposed in [14] consumes about 90% of the internal memory of a Xilinx Virtex 5 FPGA. It saves logic resources, but it is able to real-time process images with a resolution limited to 640x480 pixels. Another example is presented in [15], where an FPGA-based implementation of the SIFT algorithm is presented. It is able to real-time process 640x480 pixel images, consuming about 30,000 LUTs and 97 internal DSPs in a Xilinx Virtex 5 FPGA.

Among the available feature extraction algorithms, Harris is probably the best trade-off between precision and complexity [16]. Under the assumption of small differences between consecutive frames (i.e., high frame rates or small camera displacements), its accuracy is comparable to SURF and SIFT, with a significant lower complexity. Since high frame-rates are mandatory during the EDL phase to allow real-time correction of the descending trajectory, Harris is a very good candidate to implement a high-speed and low-area FEM accelerator block for space-applications [17]. For each pixel (x, y) of a frame, Harris computes the so called *corner response* $R(x, y)$ according to the following equation[1]:

$$R(x, y) = \mathrm{Det}(N(x, y)) - k \cdot \mathrm{Tr}^2(N(x, y)) \qquad (1)$$

where k is an empirical correction factor equal to 0.04, and $N(x, y)$ is the second-moment matrix, which depends on the spatial image derivatives L_x and L_y, in the respective directions (i.e., x and y) [7]. Pixels with high corner response have high probability to represent a corner (i.e., an image feature) of the selected frame and can be selected to search for matching points between consecutive frames.

In [8] we presented an FPGA-based FEM core called FEMIP, based on the standard Harris algorithm, overcoming limitations of previous state-of-the-art implementations [17]–[19]. It guarantees high frame rates (up to 33 fps), with very limited hardware resources and without resorting to external co-processors. Nevertheless, FEMIP parameters are fixed at design time and do not allow to adapt to the continuous environmental changes (e.g., light, noise, etc.) that are typical of extreme space missions. The architecture presented in this paper, named SA-FEMIP, overcomes this limitation by introducing adaptation capability to the architecture presented in [8], thus obtaining high FEM robustness with respect to both noise and image characteristic variations.

III. SA-FEMIP Architecture

This section shortly introduces the SA-FEMIP architecture discussing where and how adaptation to environmental conditions has been introduced.

[1]$\mathrm{Det}(X)$ denotes the determinant of matrix X, and $\mathrm{Tr}(X)$ denotes the trace of matrix X

SA-FEMIP is a pipelined architecture that processes a 32-bit input stream representing a sequence of 1024x1024 grey scale frames with 10 bit per pixel (bpp) resolution (see Fig. 1). Frame size and resolution are those provided by almost all space-qualified CMOS cameras [20]. Images are received in a raster format, line-by-line from left to right and from top to bottom. The output of SA-FEMIP is the set of matching points identified in the processed frames. The SA-FEMIP pipeline includes three main functional blocks: the *Reconfigurable Gaussian Filter*, the *Adaptive Harris Feature Extractor*, and the *Feature Matcher*. Moreover, SA-FEMIP includes an input/output interface to communicate with an external memory used to temporarily store images filtered by the *Reconfigurable Gaussian Filter* and later required during the feature matching step.

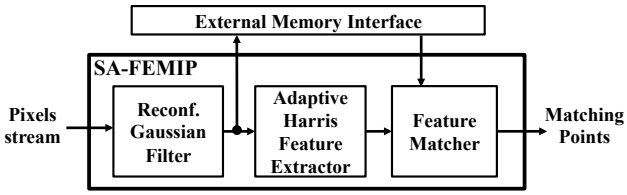


Fig. 1: SA-FEMIP computational pipeline

The *Reconfigurable Gaussian Filter* performs Gaussian smoothing of the input image. It reduces the image noise level, thus improving the feature extraction accuracy [21]. Gaussian filtering is performed by means of a 2D-convolution of the input image with a 7x7 Gaussian kernel mask [21] according to (2). A 7x7 kernel is enough to approximate a two-dimensional Gaussian function with variance $\sigma_f^2 \leq 2$ [22], which enables to forcefully reduce the noise that strongly affects images taken in space environments.

$$FI(x, y) = \sum_{i=0}^{s-1} \sum_{j=0}^{s-1} I\left(\delta x + i, \delta y + j\right) * K(i, j) \qquad (2)$$

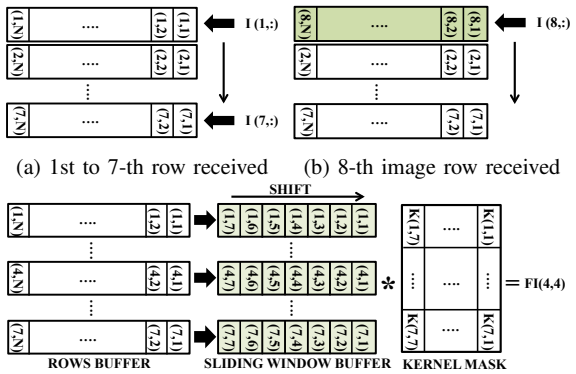
In (2), $FI(x, y)$ is the filtered pixel in position (x, y), I represents the input image, s is the kernel size ($s = 7$ in this architecture), $K(i, j)$ is the kernel factor in position (i, j), and δx and δy are computed according to the following equation:

$$\delta x, \delta y = x, y - \left(\frac{s-1}{2}\right) \qquad (3)$$

The static filter architecture presented in [8], which represents the base for the proposed *Reconfigurable Gaussian Filter*, is briefly recalled here. The reader may refer to [8] for further details on the architecture. The input pixels stream is stored in an internal buffer called *Row Buffer*, composed of 7 FPGA Block-RAMs (BRAMs) [23] each one able to store a full image row[2]. Rows are buffered using a circular policy as reported in Fig. 2. Pixels of a row are loaded from right to left, and rows are loaded from top to bottom (Fig. 2a). When the buffer is full, the first row of the buffer is used again (Fig. 2b). When the first 7 rows of the image are ready in the *Row Buffer* the actual pixel filtering starts. At this stage, pixels of

[2]The number of rows of the buffer is equal to the kernel size.

the central row (row number 4) can be processed and filtered. It is worth to remember here that, using a 7x7 kernel matrix, a 3-pixel wide border of the image is not filtered, and related pixels are therefore discarded during filtering. For each pixel to filter, a 7x7 image patch is extracted from the *Row Buffer* and stored in the *Slide Window Buffer* (i.e., a buffer composed of 49 10-bit registers). This can be efficiently done if one considers that the image is received in a raster way as shown in Fig. 2c. At each clock cycle, a full *Row Buffer* column is shifted into the *Sliding Window Buffer* (Fig. 2c). After the 7th clock cycle, the first image block is ready and the *Sliding Window Buffer* is convolved with the *Kernel Mask*. At each following clock cycle, a new *Row Buffer* column enters the *Sliding Window Buffer* and a new filtered pixel of the row is produced. While this process is carried out, new pixels continue to feed the *Row Buffer*, thus implementing a fully pipelined computation. From (2), taking into account the considered kernel size (i.e., 7x7 pixels), 49 multiplications are required to produce a filtered pixel $FI(x, y)$. In our architecture, all multiplications are executed in parallel within a single clock cycle. Since kernel factors have been internally represented through constants, 49 constant-multipliers are instantiated. After that, an adder tree (similar to the one presented in [24]) adds the 49 multiplication results to produce the filtered pixel.



(a) 1st to 7-th row received (b) 8-th image row received

(c) Slide window filling for Pixel (4,4)

Fig. 2: *Gaussian Filter* internal buffers architecture. (i,j) indicates the pixel coordinates.

The main drawback of this architecture is that a fixed Gaussian filter variance (σ_f^2) works well if the noise level of the processed frames is known a priori. As an example, a high filter variance is useful for high noise levels. Instead, for low noise levels the images are oversmoothed, thus reducing the accuracy of the feature extraction and matching modules [21]. To overcome this problem, SA-FEMIP exploits FPGA DPR to adapt the variance σ_f^2 of the *Reconfigurable Gaussian Filter* frame-by-frame, based on the estimated noise affecting the input frame (see Section IV). This represents the first main contribution to the FEM adaptation introduced in the SA-FEMIP architecture.

The *Adaptive Harris Feature Extractor* implements the Harris corner detector. It processes the filtered pixels, received from the *Reconfigurable Gaussian Filter*, and computes the frame features. Each feature is represented by its coordinates (x, y) in the frame, and by the related corner response $R(x, y)$, computed according to eq. (1). The computed corner responses must be thresholded in order to identify those features that potentially represent a real corner. However, the value of the threshold strongly depends on the image environment (e.g., Mars or Moon) and condition (e.g., brightness, noise, contrast). To provide a certain level of adaptation, [8] introduced a self-adaptive threshold. The threshold TH is initialized at 0 at startup (i.e., all features are accepted). It is then updated based on the number of features extracted from the current image, and applied to the next frame. In particular, for each frame, the number of selected features (NF) is compared with the number of expected features (TF) set to 3,000 in our specific test application. If the two numbers are equal with a tolerance (δ) the threshold is already optimized. If not, the new threshold is computed as $TH = TH + ((TF - NF) * (0.5/TF) * TH)$. The reader may refer to [8] for additional details.

Computing the threshold for the next frame based on information on the current frame is acceptable thanks to the high frame rate of the proposed architecture, that guarantees marginal differences in consecutive frames. However, if the image presents a single small rugged region, the extracted features, and subsequently the extracted matching points, will be concentrated in that limited region. This leads to poor information extracted from the input frames, and therefore to errors in the *Motion Estimation* phase. This drawback derives from the usage of a single global threshold for an entire frame. The *Adaptive Harris Features Extractor* (AHFE) component proposed in Section V, implements an adaptive cell-based thresholding that relies on frame partitioning to apply different thresholds to different portions of the frame. This ensures that the extracted features uniformly cover the overall frame.

Eventually, the *Feature Matcher* of Fig. 1 receives and stores in an internal buffer the features extracted by the *Adaptive Harris Feature Extractor*. All stored features are then analyzed to discard the ones that are too close to each other. Only the strongest feature in a 3x3 pixel neighborhood is considered and stored in an internal buffer called *NMS Buffer* (*Non-Maxima-Suppression (NMS)* phase). The *NMS Buffer* stores up to 1,000 features coordinates using 4 BRAMs. It is split in two sub-buffers named *Frame 1 Features* buffer and *Frame 2 Features* buffer alternatively used to store features associated with two consecutive images that must be analyzed and matched. Features stored in the two *NMS sub-Buffers* are then compared using a cross-correlation-based approach (*Matching* phase) to identify the matching points. Only potentially correlated features are actually compared. Analyzing the speed of a space-module during the descending phase, and considering the high input frame rate used to sample images, we identified that a feature can perform a maximum movement of 17 pixels between two consecutive images [25] [2]. Thus, two features can be considered as potentially correlated if they are both in a 35x35 pixel neighborhood in the two considered images. Cross-Correlation is therefore only computed on these features, thus reducing the computational load and speeding up the matching task. Basically, the *Feature Matcher* scans the *Frame 1 Features* buffer and the *Frame 2 Features* buffer looking for two correlated features. It compares the coordinates

associated with a feature contained in one of the two buffers with all the coordinates in the other buffer. Whenever two potentially correlated features are found, their un-normalized Cross-Correlation is computed using the intensity of all pixels contained in the two 11x11 pixels windows surrounding the two correlated features. These values (previously stored by the *Gaussian Filter*) are loaded from the external memory. Finally, the Cross-Correlation results are thresholded, in order to eliminate fake matchings. If the calculated Cross-Correlation value is less than a given threshold, the coordinates of the correlated features are stored inside an internal buffer. The reader may refer to [8] for additional details on this block.

IV. Reconfigurable Gaussian Filter

The *Reconfigurable Gaussian Filter* exploits FPGA DPR to implement frame-by-frame adaptation of the filter variance σ_f^2, based on the estimated noise affecting the input images.

In literature, many works propose adaptive filters [26] [27] [28] [29]. Among the proposed approaches, those based on evolutionary algorithms are the most promising, in terms of timing performances and hardware resources usage [30]. Nevertheless, they provide very good results if the processed images are similar to the one used during the training phase of the evolutionary algorithm. Instead, if the received image characteristics (e.g., illumination conditions, tonal distribution, etc.) cannot be predicted, as in the harsh space environment, the filtering performances become very poor [31] [32].

[33] presents a Gaussian filter architecture able to self-adapt σ_f^2 pixel-by-pixel depending on the noise level affecting the processed image. This approach ensures a higher level of adaptivity with respect to evolutionary filters. However, it wastes a lot of hardware resources, making it not suitable for space-applications that require high optimization. To overcome this issue, we exploit FPGA DPR to provide filter adaptation while saving area and power consumption. Basically, the proposed approach estimates the level of noise affecting the input image using the same algorithm adopted in [33]. The noise level estimated for the current frame is used to select the filter variance that would guarantee optimum filtering results. This filter variance is then used to filter the next input image, allowing adaptation of the filter parameters frame-by-frame during the entire descending sequence. The adaptation of the filter variance is achieved by reconfiguring the 49 constant multipliers required to perform the convolution of the image with the Gaussian kernel (see Section III). This significantly saves hardware resources with respect to a solution that uses 49 generic multipliers in which the Gaussian kernel constants are selected using multiplexers driven according to the selected filter variance. Fig. 3 shows the architecture implementing the proposed approach.

The *Reconfigurable Gaussian Filter* is composed of: (i) the *Noise Variance Estimator* (NVE), (ii) the *Reconfiguration Manager*, and (iii) the *Gaussian Filter*. The *Gaussian Filter* is implemented as described in Section III, but, in order to enable its reconfiguration, the 49 multipliers are enclosed in an FPGA reconfigurable module (RM in Fig. 3). A reconfigurable module is a portion of an FPGA design

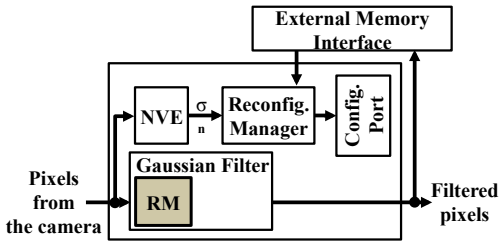


Fig. 3: Reconfigurable Gaussian Filter hardware architecture

that can be reconfigured at run-time, without impacting the behavior of the rest of the design. While the *Gaussian Filter* processes the input image, the NVE estimates the noise level. The NVE is implemented adopting an architecture similar to the one presented in [33]. However, we compute here the noise standard deviation (σ_n) instead of the noise variance (σ_n^2). This change has not functional effects since σ_n is not actually used to perform calculations like in [33] during the filtering process. When a full frame has been processed, the NVE provides the current estimated σ_n to the *Reconfiguration Manager*. The *Reconfiguration Manager* exploits this value to look-up into a bitstream address table and to select the proper configuration for the multipliers inside the RM. The multipliers reconfiguration is accomplished by reading the multipliers configuration bitstream from the external memory, choosing the configuration associated with the estimated standard deviation. The bistream is then used to program the reconfigurable module of the FPGA resorting to the FPGA internal Configuration Port (i.e., *ICAP* [34] in Xilinx FPGAs). It is worth to highlight here that using the noise standard deviation instead of the noise variance strongly reduces the NVE area, avoiding the multiplier required to compute σ_n^2.

During the reconfiguration process, the *Reconfiguration Manager* must access the external memory to retrieve the RM configuration bitstream. To avoid the stall of the processing chain, this access must be scheduled when no other module requires information from the external memory. As shown in the timing diagram of Fig. 4, the external memory is accessed by the *Reconfigurable Gaussian Filter* in write mode to store the computed filtered pixel values. As described in Section III, during this phase the *Reconfigurable Gaussian Filter* and the *Adaptive Harris Feature Extractor* work in pipeline, while the noise variance is computed (Image Filtering, Features Extraction and Noise Estimation activities in Fig. 4). At the end of the feature extraction, the *NMS* phase takes place, and, finally, the *Feature Matcher* performs the *matching* phase where it accesses the external memory in read mode to retrieve the data needed to compute the cross-correlation. It is worth noting that the Image Filtering and the Features Extraction slots are not perfectly aligned due to the latency in loading the internal pipeline of the *Reconfigurable Gaussian Filter*. Looking at Fig. 4, the external memory is always idle during the NMS phase (t_{idle} in Fig. 4). This time slot can be used to reconfigure the filter (R task in Fig. 4) without stalling the processing chain. This means that no timing overhead is introduced in the feature extraction and matching task.

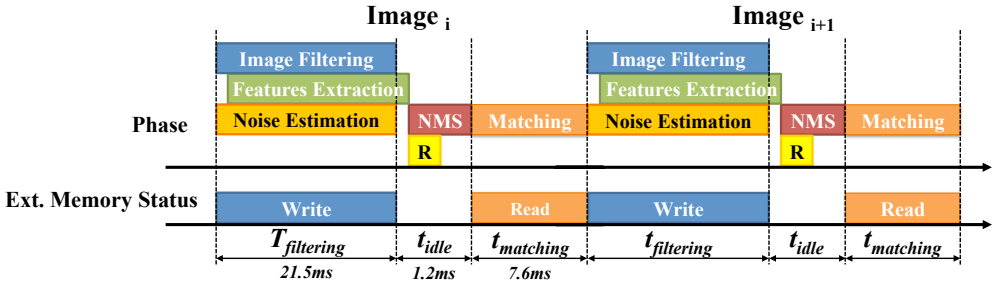


Fig. 4: Timing diagram of SA-FEMIP

Finally, since for each value of σ_f^2 a configuration bitstream must be stored in the external memory, the range of possible σ_f^2 must be discretized according to the available external memory space (see Section VI for detailed information about the size of each bitstream).

V. Adaptive Harris Feature Extractor

This section introduces the hardware architecture of the *Adaptive Harris Feature Extractor* (Fig. 5), focusing on the novel thresholding approach that ensures to uniformly distribute the extracted features on the input frame.

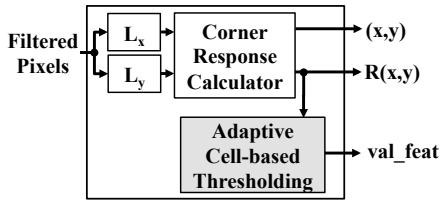


Fig. 5: *Adaptive Harris Features Extractor* internal architecture

The first two modules of the *Adaptive Harris Feature Extractor*, L_x and L_y, compute the spatial image derivatives of the filtered image in the horizontal (L_x) and vertical (L_y) direction, respectively. This operation is performed by convolving the filtered image, received from the *Reconfigurable Gaussian Filter*, with the 3x3 Prewitt kernel [21], using an architecture similar to the one proposed for the Gaussian Filter. Then, the *Corner Response Calculator* module computes the determinant and the trace of the second-moment matrix $N(x, y)$, which are required to calculate the Harris corner response $R(x, y)$ associated with each input pixel. Finally, the *Adaptive Cell-based Thresholding* (ACTH) module thresholds the computed corner responses, asserting the *val_feat* signal when the current processed pixel is above the threshold and therefore represents an actual feature.

Selecting a well distributed set of features within the frame improves the motion estimation accuracy. In order to level the distribution of the extracted features on the processed frames, the ACTH module splits the input image in 64 cells of 128x128 pixels each. It then tries to extract the same number of features from each cell. This goal is achieved exploiting a local threshold for each cell, instead of using a single global threshold for the overall image, as done in [8]. The chosen number of cells represents a good trade-off between accuracy and memory requirements. As it will be discussed in Section VI, it ensures to uniformly cover the input image and, at the same time, to avoid the introduction of a large number of cells that would require a lot of memory to store the related information items.

The ACTH module analyzes information related to the current frame implementing the decision process described in Alg. 1, and computes the local thresholds to use for the following frame. The threshold adaptation process requires to know, for each cell composing the frame, (i) the number of extracted features (NF), (ii) the current threshold (TH) initialized at the highest possible value at startup (i.e., no features are extracted), and (iii) the current number of expected features (TF). In our tests, TF has been initialized to 48 to fix the overall number of expected features per frame (OTF) to about 3,000 features. This value limits the size of the internal buffer used to store the extracted features in the *Feature Matcher* module. Since NF, TH and TF must be defined for each cell of the frame, they are stored in the form of 8x8 matrices, with the matrix elements associated to the defined frame cells.

Alg. 1 can be split in two main parts. The former (from row 8 to 27) updates the cell threshold values. For every cell (i, j), it compares the number of extracted features $NF[i, j]$ with the number of expected features $TF[i, j]$ ($Disp$ at row 10). If these two values differ no more than a defined tolerance (i.e., the difference is contained in the range $[+\delta, -\delta]$) the threshold is not changed (row 23). Otherwise, the threshold is updated adding $Step$ to its current value (rows 13 and 21). One additional test is performed when the number of extracted features is lower than the number of expected ones (from row 14 to 18). In particular, the updated threshold ($new_TH[i, j]$) is considered valid if it is higher than a lower bound value ($LowTH$). If not, the threshold is not changed (row 15). This avoids to over-reduce the threshold value and to provide in output weak features that could be potentially associated with the noise in the input frame. In fact, if a cell represents a flat part of the planetary surface, a high value of the image gradient, and consequently a high value of the computed corner response, is mainly due to the noise.

The second part of Alg. 1 (from row 28 to 51) optimizes the number of features extracted for each cell in order to obtain a total number of features for the frame as close as

Algorithm 1 Adaptive Cell Thresholding approach

```
Require: NF[8,8]                                  ▷ # extracted features in each cell
Require: TH[8,8]                                  ▷ Threshold value of each cell
Require: TF[8,8]                                  ▷ # target features in each cell
 1: Const N_cell=64                               ▷ # of cell
 2: Const δ=15                                    ▷ Tolerance
 3: Const LowTH=15                                ▷ Threshold lower bound
 4: Const OTF=3000                                ▷ Overall # target features
 5: Curr_EF=∑ NF[i, j]                            ▷ Current overall # extracted features
 6: LowTH_cell[8,8]=[0,...,0]
 7: TF_slack=0
 8: for i=0;i<8;i++ do
 9:     for j=0;j<8;j++ do
10:         Disp=NF[i,j]-TF[i,j]
11:         Step = Disp * (0.5/OTF)*TH[i,j]
12:         if Disp< −δ then
13:             new_TH[i,j]=TH[i,j]+Step
14:             if new_TH[i,j]< LowTH then
15:                 new_TH[i,j]=TH[i,j]
16:                 LowTH_cell[i,j]=1
17:                 TF_slack=TF_slack + | Disp |
18:             end if
19:         else
20:             if Disp> +δ then
21:                 new_TH[i,j]=TH[i,j]+Step
22:             else
23:                 new_TH[i,j]=TH[i,j]
24:             end if
25:         end if
26:     end for
27: end for
28: if TF_slack > 0 then
29:     if TF_slack < N_cell then
30:         TF_slack_cell = 1
31:     else
32:         TF_slack_cell = ⌊ TF_slack ÷ N_cell⌋
33:     end if
34:     for i=0;i<8;i++ do
35:         for j=0;j<8;j++ do
36:             if Curr_EF <= OTF then
37:                 if LowTH_cell[i,j]=0 then
38:                     new_TF[i,j]=TF[i,j]+TF_slack_cell
39:                 else
40:                     new_TF[i,j]=NF[i,j]
41:                 end if
42:             else
43:                 if TF[i,j] = 0 then
44:                     new_TF[i,j]=TF[i,j]
45:                 else
46:                     new_TF[i,j]=TF[i,j]-1
47:                 end if
48:             end if
49:         end for
50:     end for
51: end if
52: return (new_TH, new_TF)
```

possible OTF. To do that, it is worth to remember that all cells that reach the threshold lower bound cannot further update their threshold. If, with this threshold, the number of extracted features for the cell $NF[i,j]$ is lower than the number of expected features for the cell $TF[i,j]$ there is a certain amount of features corresponding to $\mid Disp \mid$ that can be redistributed to other cells with threshold higher than the lower bound. To exploit this, each cell with threshold lower than the lower bound is marked through the $LowTH_cell[i,j]$ flag (row 16) and the number of unused features of these cells is accumulated in the TF_slack parameter (row 17) in order to be redistributed to the other cells, according to the decision process described from row 28 to 51. The TF_slack represents the number of expected features that can be borrowed to the cells that have not reached the threshold lower bound (i.e., $LowTH_cell[i,j] = 0$). The number of features to borrow to each cell (TF_slack_cell) is computed dividing TF_slack by the number of cells composing the image. To ensure a high number of extracted features, the algorithm always borrows at least one feature to each cell with $LowTH_cell[i,j] = 0$ (rows 29 to 33). If the total number of extracted features $Curr_EF$ is lower or equal to OTF (from row 36 to 41), if the cell has not reached the threshold lower bound the number of expected features for the cell is increased of TF_slack_cell (row 38). Otherwise, it is left unchanged (row 40).

Using this approach, the total number of extracted features ($Curr_EF$) could increase more and more due to the borrow mechanism, that increases the $TF[i,j]$ values. To allow a decrease of the $TF[i,j]$ values, and so to maintain the overall number of extracted features around OTF, if $Curr_EF$ exceeds OTF, the target feature value of each cell is decreased by 1 (from row 43 to 47).

The hardware architecture of the ACTH module is shown in Fig. 6.

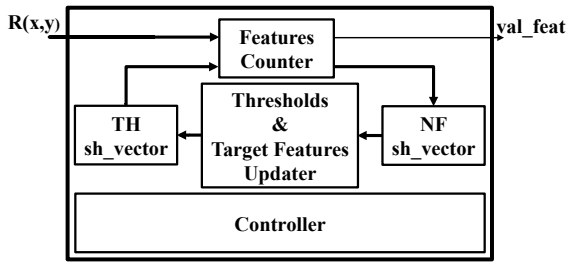


Fig. 6: Adaptive Cell-based Thresholding hardware architecture

It is composed of four main modules: (i) the *Features Counter*, (ii) the *Thresholds & Target Features Updater*, (iii) the *TH sh_vector*, and (iv) the *NF sh_vector*. The *Thresholds & Target Features Updater* module implements Alg. 1, while the *Features Counter* performs the actual thresholding of each corner response $R(x,y)$ received from the *Corner Response Calculator* (see Fig. 5). This module reads the thresholds associated with each image cell (i.e., $TH[i,j]$ in Alg. 1) that are stored in the *TH sh_vector*, and compares them with the received corner responses, asserting the *val_feat* signal if $R(x,y)$ is higher than the threshold associated with the image cell containing the currently processed pixel.

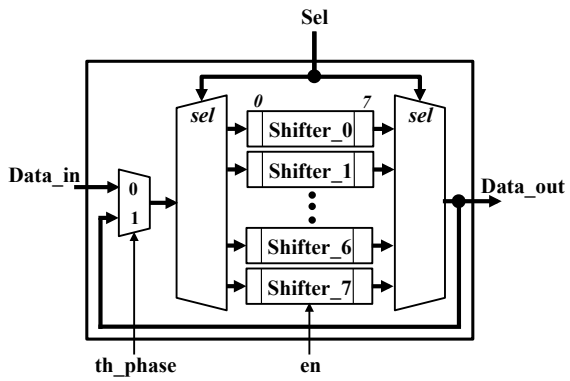


Fig. 7: TH and NF shifter vector hardware architecture

The *TH sh_vector* module is implemented as in Fig. 7. It is composed of eight 8-positions shift registers connected as circular buffers. Each shift register stores eight threshold values associated with a row of image cells (it is worth to remember that the image is split in 64 cells organized in 8 rows with 8 cells each, and a threshold value is associated with each cell). The *en* signal enables the 1-position right shifting operation, while the *Sel* signal selects which shift register must be rotated. These two control signals are driven in order to provide in output the threshold associated with the image cell of the currently processed pixel. Since the image is received in a raster way, and each image cell is composed of 128x128 pixels, *en* is asserted for a clock cycle every 128 received corner responses (i.e., whenever we move from a cell to the following one). Instead, *Sel* selects the next shift register (i.e., the next row of image cells) after 128x1024 received corner responses (i.e., whenever a complete row of image cells has been processed). To avoid loosing the threshold values, during the thresholding phase each shift register composing the *TH sh_vector* acts as a circular buffer through the multiplexer driven by the *th_phase* signal (see Fig. 6). Instead, during the thresholds updating phase, the content of the *TH sh_vector* is overwritten (exploiting the *Data_in* port) with new thresholds values computed by the *Thresholds & Target Features Updater* module.

Simultaneously to the thresholding task, the *Features Counter* counts (through an accumulator) the number of extracted features for each image cell (i.e., $NF[i,j]$ in Alg. 1), and stores these values inside the *NF sh_vector*. The *NF sh_vector* is implemented as the *TH sh_vector* (Fig. 7), and both modules share the input control signals. Whenever we move from the current image cell to the next one, the content of the internal accumulator is stored inside the *NF sh_vector*, and it is initialized with the output value provided by the *NF sh_vector*. At the end of the operations described by Alg. 1, a local reset is asserted to clear the content of the *NF sh_vector* in order to prepare it for the next image processing cycle. All aforementioned control signals are generated by the *Controller* module (Fig. 6), which also coordinates the operations of all modules included in the *ACTH*.

VI. Experimental Results

To estimate the hardware resources and the timing performances, the SA-FEMIP architecture has been synthesized and implemented, resorting to *Xilinx ISE Design Suite 14.6*, on a space-qualified Xilinx *Virtex 4-QV VLX200* FPGA device that, together with the *Virtex 5-QV VFX130* FPGA, represents the state-of-the-art architecture for space-qualified reprogrammable FPGAs. The reason to select the Virtex 4 architecture instead of the newer Virtex 5 is twofold. First the SA-FEMIP architecture has been designed to be integrated and tested inside the Thales Alenia Space Avionic Testbench (ATB), i.e., a hardware infrastructure emulating the on-board computing platform of a spacecraft. The ATB is equipped with a *Gaisler Research GR-CPCI-XC4V* development board [35]. This board integrates a Xilinx *Virtex 4 VLX200* FPGA, which provides the same internal logic architecture of the space-qualified version. Second, implementing SE-FEMIP on a Virtex 4 FPGA allowed us to perform fair comparisons with other published architectures, thus highlighting the benefits of the introduced improvements. Post place and route simulations have been done using *Modelsim SE 10.0c* to annotate the switching activities of internal nodes, and the *Xilinx XPower Analyzer* has been exploited of power consumption estimation.

Table I compares the proposed adaptive architecture with the fixed architecture proposed in [8] (FEMIP). Comparison is performed in terms of area overhead by considering internal logic and memory resources (i.e., registers, Look-Up Tables (LUTs), and Block-RAMs (BRAMs) [23]). Percentages in Table I represent the used portion of the hardware resources available in the Xilinx *Virtex 4-QV VLX200* FPGA. It is important to point out that the synthesis of both *FEMIP* and *SA-FEMIP* architectures has been forced to avoid the use of DSPs. The reasons for this choice will be better elaborated later in this section. Power consumption is analyzed considering an operating frequency of 60 MHz for both architectures.

TABLE I: Resources usage and power consumption of *FEMIP* and SA-FEMIP, implemented on a *Xilinx XQR4VLX200 Virtex 4 FPGA device*

	Module	FPGA Area Occupation			Max.Freq.	Power
		Registers	LUTs	BRAMs	[MHz]	[W]
FEMIP [8]	*GF*	696 (0.30%)	5,896 (3.31%)	7 (2.08%)	118.36	0.064
	HFE	1,106 (0.62%)	11,081 (6.22%)	6 (1.79%)	62.55	0.407
	FM	2,432 (1.36%)	656 (0.37%)	19 (5.65%)	101.30	0.037
	Total	**4,234 (2.38%)**	**17,633 (9.89%)**	**32 (9.52%)**	62.55	**2.002**
Proposed	*RF*	939 (0.53%)	7,448 (4.18%)	10 (2.98%)	118.36	0.083
	AHFE	1,362 (0.76%)	12,468 (7.00%)	6 (1.79%)	62.55	0.462
	FM	2,432 (1.36%)	656 (0.37%)	19 (5.65%)	101.30	0.037
	Total	**4,733 (2.66%)**	**20,576 (11.55%)**	**35 (10.42%)**	62.55	**2.097**
Overhead	**Total**	**499 (0.28%)**	**2,943 (1.66%)**	**3 (0.90%)**	0	**0.095**

Table I shows that SA-FEMIP FPGA occupation is around 10% for logic and memory resources, and the overhead w.r.t. FEMIP is less than 2%. This overhead is due to the additional modules required to perform adaptation in the *Reconfigurable Gaussian Filter* (RF) described in Section IV and the additional hardware required to implement the *Adaptive Harris Features Extractor* (AHFE) presented in Section V. In particular, in the RF, the increased occupation is due to the NVE and the Reconfiguration Manager modules. Instead, in the AHFE, the area overhead is introduced by the usage of a more complex thresholding approach, with respect to the simple one adopted in FEMIP. It is worth to highlight here that an effort has been placed to limit the registers overhead. The AHFE architecture strongly relies on shift registers structures to implement the required vectors and matrices included in Alg. 1. This kind of component can be efficiently implemented in Xilinx FPGAs, exploiting the Xilinx *SRL* capability of the Look-Up Tables (LUTs) [23]. This capability makes it possible to use LUTs as shift registers instead of a chain of Flip-Flops, saving hardware resources. As an example, a single LUT, in a *Xilinx Virtex-4*, can act as a 16x1-bit shift register avoiding a chain of 16 flip-flops [23].

The maximum operating frequencies of each module reported in Table I demonstrate that no timing penalty is introduced in SA-FEMIP by the introduction of the adaptivity

features.

The power consumption of each module reported in Table I does not take into account the contribution of the clock circuitry and the leakage. These contributions are included in the overall power consumption. By comparing the power consumption of SA-FEMIP with the one of FEMIP a very limited overhead equal to 4.75% is observed. It is worth noting that the power consumption of the RF module does not include the power used during the partial reconfiguration process. However, according to [36] the reconfiguration process consumes few tens of mW, only.

Eventually, the throughput, in terms of frames-per-second (fps), is the same (i.e., 33 fps) for both FEMIP and SA-FEMIP.

In Table II, the performances and the area occupation of SA-FEMIP have been compared with FEIC [17] [37]. FEIC is a Feature Extraction and matching Integrated Circuit, based on the Harris algorithm, that University of Dundee developed for the European Space Agency (ESA) in the framework of the Navigation for Planetary Approach and Landing (NPAL) project. LUTs and BRAMs used by FEIC are reported for a Virtex II device (as in [37]), but the internal logic and memory architecture is the same as in Virtex 4 family devices. The reported data confirm the great improvements of the proposed architecture, both in terms of resources usage and throughput.

TABLE II: Resource usage and throughput of *FEIC* and SA-FEMIP for a *Xilinx XQR4VLX200 Virtex 4 FPGA device*

	Resource Usage		**Max. Speed**
	LUTs	**BRAMs [KB]**	**[fps]**
Proposed	20,576	78.75	33
FEIC [37]	50,688	162.5	20
Improvements	**-59.4%**	**-51.5%**	**+65%**

The low area occupation of SA-FEMIP allow designers to exploit the free hardware resources to apply fault mitigation strategies to increase the reliability of the design, a key requirement in space applications. Several fault-mitigation strategy against Single Event Upset (SEU) can be applied on FPGA devices. Following [38], these techniques can be classified as (1) *netlist level techniques* or (2) *register transfer level techniques*.

Netlist level techniques include different types of Triple Modular Redundancy (TMR) techniques [38]. Triplication can be limited to the sequential elements of the circuit (i.e., *Sequential logic TMR*) introducing for each register of the design two additional registers and a 3-input voter. Otherwise, the full design can be triplicated (i.e., *Global TMR*) introducing a hardware overhead equal to the 200% of the original design.

Register transfer level techniques aim at protecting the Finite State Machines (FSMs) of the design (e.g., *Safe FSM Coding*, and *3-Hamming distance enhancement in FSMs*). Usually, the overhead introduced by these techniques is one order of magnitude lower than the one associated with the TMR techniques.

In general, the total hardware overhead, even if a combination of the aforementioned techniques is exploited, can vary from 60% up to 200% of the original design [38]. It is clear that, given the low amount of resources required by SA-FEMIP, fault tolerance techniques can be freely implemented within the selected device. Moreover, even after the implementation of fault tolerance techniques, space is also available to integrate in the same device additional FPGA-based IP-cores useful to accelerate other computational intensive tasks performed during the descending phase (e.g., *Hazard map computation* [39]). This is very important considering the limited resources available in space applications.

As mentioned at the beginning of this section SA-FEMIP has been synthesized avoiding the use of DSPs. This decision can now be better motivated. DSPs have the advantage of further reducing the area occupation of FEMIP especially when multipliers are implemented. With the use of DSPs the SA-FEMIP occupation would be reduced to 9,029 (5.06%) LUTs, 66 (68.75%) DSPs, while the occupation of registers and BRAMs remains the same. Nevertheless, DSPs are limited resources. With 66 DSPs required out of the 96 available in the Virtex 4 VLX200 FPGA, TMR techniques for this portion of the design would not be possible. Moreover, the intensive use of DSPs increase the routing complexity introducing a 30% frequency penalty in the design.

SA-FEMIP has not been compared to [19], since [19] implements the multi-scale Harris detector (i.e., a rotation-invariant version of the Harris detector). [19] consumes a lot of hardware resources, and implements a feature that is not actually required in EDL applications since rotations between two consecutive images are limited [40].

The proposed architecture has been evaluated in terms of accuracy and robustness, exploiting an image dataset, provided by *Thales Alenia Space Italia s.p.a.* company, that covers different landing zones (i.e., portions of the Mars surface), environmental conditions (i.e., image quality), and camera movement types, in a synthesized Mars environment. Camera movement types include displacements, up to 30 meters, at different altitudes (from 1,000 meters to 5,000 meters), and angular speed (up to 2.5 $^\circ/s$, in accordance to [40]), while image quality types include the injection of different levels of Gaussian noise, blur, brightness and contrast variations.

According to [40], the robustness has been evaluated exploiting two parameters: (i) *Number of Extracted Matches* (NEM), that identifies the number of matching points, and (ii) *Spatial Distribution of Points* (SDP), that measures how much the extracted matching points are uniformly spread in the image, defined as:

$$SDP = \frac{\sum_{i=1}^{N} -p_i \log p_i}{\log N} \tag{4}$$

where p_i is computed as the number of matching points within an image cell (see Section V) over the total number of extracted matching points in the frame, and N is the number of image cells (i.e., 64).

Fig. 8 shows the SDP results obtained from FEMIP [8] and SA-FEMIP by providing in input the images composing the aforementioned dataset. Thanks to the adaptive cell-based

thresholding approach, the proposed architecture outperforms FEMIP results in every test case (i.e., Test Index). In particular, the improvements are very high (from Test Index 0 to 76) when the input images represent a landing zone characterized by few small rugged regions. This is visually highlighted in Fig. 9 that depicts the matching points extracted by FEMIP (Fig. 9a) and SA-FEMIP (Fig. 9b). Each figure shows two consecutive input images with lines connecting the features that match in the two images.

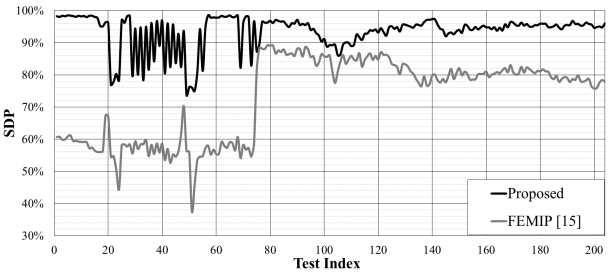
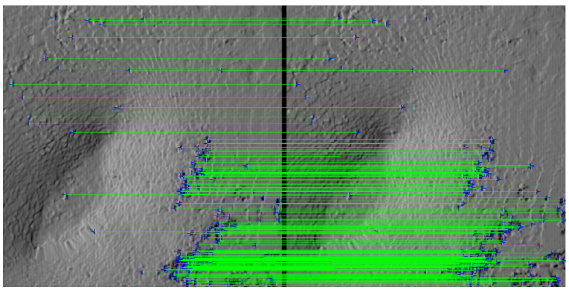


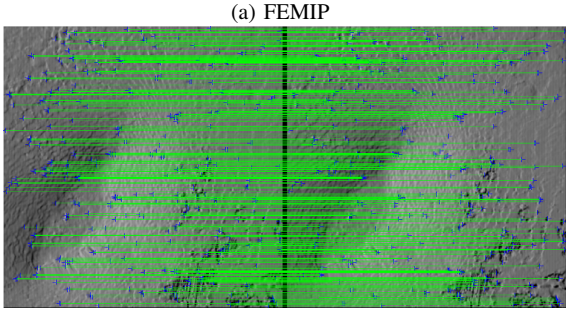
Fig. 8: SDP results for FEMIP and the proposed architecture

Fig. 10 shows the NEM versus different levels of injected Gaussian noise variance σ_f^2 (since FEMIP has a fixed $\sigma_f^2 = 2$, its NEM is represented by the dashed line).

A fixed σ_f^2 does not allow to reach the highest NEM for every noise level. Thus, exploiting the reconfigurable filter architecture (see Section IV) it is possible to highly increase the number of extracted matches, as shown by the *Optimal* line in Fig. 10. In order to follow the trend of this line, in the proposed architecture 5 configurations for the RF module have been chosen. In particular, these configurations are associated to σ_f^2 equal to 0.5, 0.75, 1, 1.5 and 2, for the noise level



(a) FEMIP



(b) Proposed architecture

Fig. 9: Example of extracted matches

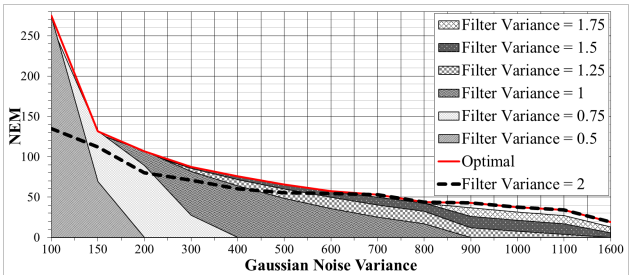


Fig. 10: NEM results for different levels of injected Gaussian noise, varying the Gaussian Filter variance

ranges [0,100], [100,200], [200,300], [300,600], [600,1600], respectively. As can be seen in Fig. 10, the usage of a reconfigurable filter increases the NEM value w.r.t. FEMIP up to 2 times, especially for a σ_n^2 lower than 600.

Moreover, as described in Section IV, the usage of the DPR enables to save resources with respect to use a static hardware architecture including 49 multipliers, each one with a multiplexer to select the right Gaussian kernel value. In the proposed architecture, using the same fixed-point data representation adopted in [8], the RM and the *Reconfiguration Controller* (Fig. 3) require 5,320 LUTs and few registers. Instead, a static hardware architecture (as the one reported in [33]), with the same data parallelism, would require about 19,000 LUTs, leading to a save of 72% of hardware resources.

Since each bitstream for the RM module is 166 KB (for the selected FPGA device), to store the 5 configurations 830 KB are required in the external memory. Since the throughput of the *Reconfiguration Controller* is 400 MB/s (i.e., this value is limited by the maximum throughput of the *ICAP* [34]), the time required to reconfigure the RM is equal to 0.42 ms. This time fits the idle time of the external memory (i.e., t_{idle} in Fig. 4) that is equal to 1.2 ms (i.e., the time required by the *Matcher* to perform the *NMS* phase). For the sake of completeness, considering an operating frequency of SA-FEMIP chain equal to 60 MHz, the time required to perform the filtering and the matching tasks (i.e., $t_{filtering}$ and $t_{matching}$ in Fig. Fig. 4) is 21.5 ms and 7.6 ms, respectively.

Eventually, Fig. 11 shows the percentages of Correct Matches (CM) for the different filter configurations and injected noise levels. CM has been computed exploiting the knowledge about the camera movement between two consecutive images of the dataset. Starting from the position of a matching point in the first image, it is possible to compute its expected position in the second image by using a three dimensional roto-traslation model. For each couple of images in the dataset this process has been automated through a *MATLAB* script. Then, the CM values have been computed by comparing the outputs of the script with the ones of the proposed architecture.

It is worth noting that, the CM values are not computed for every σ_f^2 since, as shown in Fig. 10, with a filter characterized by a low variance it is not possible to extract matching points for very high noise levels.

As can be seen in Fig. 11, the accuracy of the different filter

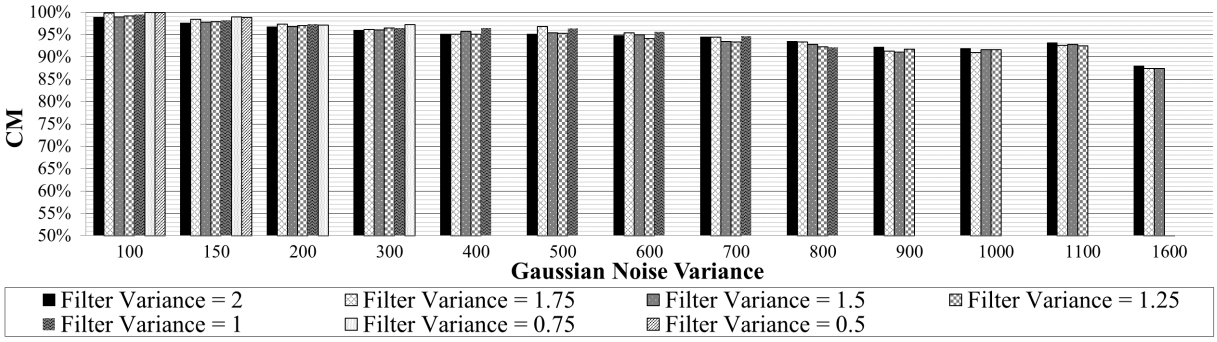


Fig. 11: Correct Matches (CM) results for different levels of injected Gaussian noise, varying the Gaussian Filter variance

configurations is higher than 85% for every noise level, and it is almost equal for a fixed noise level. These data demonstrate that the proposed filter is able to maximize the NEM, while preserving the correctness of its outputs.

VII. Conclusion

This paper presented a self-adaptive module for features extraction and matching designed to suit modern space-qualified FPGAs. In order to make the core completely self-adaptive, the Dynamic Partial Reconfiguration (DPR) feature of modern space-qualified FPGAs is exploited to design a self-reconfigurable Gaussian Filter module, while a novel adaptive algorithm and the associated hardware architecture, embedded in the features extraction module, increase the quality of the output results of the entire features extraction and matching processing chain.

Experimental results show the accuracy and the limited overhead of resources needed to implemented the proposed architecture, while maintaining the same throughput performances with respect to a state-of-the-art reference architecture, allowing to exploit the free hardware resources to apply fault mitigation strategies to increase the reliability of the design.

Acknowledgment

The authors would like to express their sincere thanks to the whole design team of *Thales Alenia Space Italia s.p.a* company for their helpful hints, guidelines, and fruitful brainstorming meetings.

References

[1] NASA, "Solar system exploration roadmap," [Accessed 28-July-2014]. [Online]. Available: http://solarsystem.nasa.gov/multimedia/downloads/SSE_RoadMap_2006_Report_FC-A_med.pdf

[2] ——, "NASA Curiosity Rover: Entry, Descent, and Landing," [Accessed 28-July-2014]. [Online]. Available: http://mars.jpl.nasa.gov/msl/mission/timeline/edl/

[3] A. Mourikis, N. Trawny, S. Roumeliotis, A. Johnson, A. Ansar, and L. Matthies, "Vision-aided inertial navigation for spacecraft entry, descent, and landing," *IEEE Transactions on Robotics*, vol. 25, no. 2, pp. 264–280, April 2009.

[4] J. Zhang, W. Liu, and Y. Wu, "Novel technique for vision-based UAV navigation," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 47, pp. 2731–2741, 2011.

[5] P. Nangtin, P. Kumhom, and K. Chamnongthai, "Video-based obstacle tracking for automatic train navigation," in *Proc. of 2005 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*, 2005, pp. 21–24.

[6] E. Loupias, N. Sebe, S. Bres, and J.-M. Jolion, "Wavelet-based salient points for image retrieval," in *Proceedings. 2000 International Conference on Image Processing*, vol. 2, 2000, pp. 518–521 vol.2.

[7] C. Harris and M. Stephens, "A combined corner and edge detector," in *Proc. of the 4th Alvey Vision Conference*, 1988, pp. 147–151.

[8] S. Di Carlo, G. Gambardella, P. Prinetto, D. Rolfo, P. Trotta, and P. Lanza, "FEMIP: A high performance FPGA-based features extractor and matcher for space applications," in *23rd International Conference on Field Programmable Logic and Applications (FPL)*, 2013, pp. 1–4.

[9] P. Beaudet, "Rotationally invariant image operators," in *Proc. of 4th International Joint Conference on Patter Recognition*, 1978, pp. 579–583.

[10] S. M. Smith and J. M. Brady, "SUSAN - a new approach to low level image processing," *International Journal of Computer Vision (IJCV)*, vol. 23, pp. 45–78, 1995.

[11] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, "SURF: Speeded up robust features," *Computer Vision and Image Understanding (CVIU)*, vol. 110, pp. 346–359, 2008.

[12] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ser. ICCV '99. Washington, DC, USA: IEEE Computer Society, 1999.

[13] N. Battezzati, S. Colazzo, M. Maffione, and L. Senepa, "SURF algorithm in FPGA: A novel architecture for high demanding industrial applications," in *Proc. of 2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2012, pp. 161–162.

[14] D. Bouris, A. Nikitakis, and I. Papaefstathiou, "Fast and efficient FPGA-based feature detection employing the SURF algorithm," in *18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2010, pp. 3–10.

[15] L. Yao, H. Feng, Y. Zhu, Z. Jiang, D. Zhao, and W. Feng, "An architecture of optimised SIFT feature detection for an FPGA implementation of an image matcher," in *International Conference on Field-Programmable Technology, 2009. FPT 2009.*, 2009, pp. 30–37.

[16] T. Tuytelaars and K. Mikolajczyk, *Local Invariant Feature Detectors: A Survey*. Now Publishers Inc., 2008.

[17] M. Dunstan, S. Parkes, and S. Mancuso, "Visual navigation chip for planetary landers," in *Proc. of 2005 Conference on DAta Systems In Aerospace (DASIA)*, 2005, pp. 1–7.

[18] A. Benedetti and P. Perona, "Real-time 2D feature detection on a reconfigurable computer," in *Proc. of 1998 Conference on Computer Vision and Pattern Recognition (CVP)*, 1998, pp. 586–593.

[19] C. Cabani and W. MacLean, "A proposed pipelined-architecture for FPGA-based affine-invariant feature detectors," in *Proc. of 2006 Computer Vision and Pattern Recognition Workshop (CVPRW)*, 2006, pp. 121–126.

[20] Cypress Semiconductor Corporation, "Star1000 1m pixel radiation hard cmos image sensor," [Accessed 28-July-2014]. [Online]. Available: http://www.onsemi.com/pub_link/Collateral/NOIS1SM1000A-D.PDF

[21] R. González and R. Woods, *Digital Image Processing*. Pearson/Prentice Hall, 2008.

[22] J. Fernández-Berni, R. Carmona-Galán, F. Pozas-Flores, Á. Zarándy, and Á. Rodríguez-Vázquez, "Multi-resolution low-power gaussian filtering

by reconfigurable focal-plane binning," in *SPIE Microtechnologies*. International Society for Optics and Photonics, 2011, pp. 806 806–806 806.

[23] Xilinx Corporation, "Virtex-4 FPGA User Guide - UG070," [Online - accessed 28-July-2014]. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug070.pdf

[24] S. Di Carlo, G. Gambardella, M. Indaco, D. Rolfo, G. Tiotto, and P. Prinetto, "An area-efficient 2-D convolution implementation on FPGA for space applications," in *Proc. of 6th International Design and Test Workshop (IDT)*, 2011, pp. 88 –92.

[25] Thales Alenia Space, "Aerospace module speed and trajectory estimation - internal report," Tech. Rep., 2012.

[26] F. Russo, "A method for estimation and filtering of gaussian noise in images," *IEEE Transactions on Instrumentation and Measurement*, vol. 52, no. 4, pp. 1148 – 1154, 2003.

[27] Z.-B. Zhao, J.-S. Yuan, Q. Gao, and Y.-H. Kong, "Wavelet image de-noising method based on noise standard deviation estimation," in *Proc. of International Conference on Wavelet Analysis and Pattern Recognition (ICWAPR)*, vol. 4, 2007, pp. 1910 – 1914.

[28] G. Deng and L. Cahill, "An adaptive gaussian filter for noise reduction and edge detection," in *Proc. of Nuclear Science Symposium and Medical Imaging Conference*, 1993, pp. 1615 – 1619 vol.3.

[29] J. Tian and L. Chen, "Image noise estimation using a variation-adaptive evolutionary approach," *IEEE Signal Processing Letters*, vol. 19, no. 7, pp. 395 – 398, 2012.

[30] R. Dobai and L. Sekanina, "Image filter evolution on the xilinx zynq platform," in *Adaptive Hardware and Systems (AHS), 2013 NASA/ESA Conference on*, 2013, pp. 164–171.

[31] J. Mora, A. Gallego, A. Otero, B. Lopez, E. de la Torre, and T. Riesgo, "A noise-agnostic self-adaptive image processing application based on evolvable hardware," in *2013 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2013, pp. 351–352.

[32] J. Mora, A. Gallego, A. Otero, E. de la Torre, and T. Riesgo, "Noise-agnostic adaptive image filtering without training references on an evolvable hardware platform," in *2013 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2013, pp. 182–189.

[33] S. Di Carlo, P. Prinetto, D. Rolfo, and P. Trotta, "AIDI: An adaptive image denoising FPGA-based IP-core for real-time applications," in *Adaptive Hardware and Systems (AHS), 2013 NASA/ESA Conference on*, 2013, pp. 99–106.

[34] Xilinx Corporation, "Partial reconfiguration user guide - ug702," [Accessed 28-July-2014]. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_3/ug702.pdf

[35] Gaisler Research AB, "GR-CPCI-XC4V LEON PCI Virtex 4 development board - product sheet," [Accessed 28-July-2014]. [Online]. Available: http://www.pender.ch/docs/GR-CPCI-XC4V_product_sheet.pdf

[36] R. Bonamy, D. Chillet, S. Bilavarn, and O. Sentieys, "Power consumption model for partial and dynamic reconfiguration," in *2012 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, 2012, pp. 1–8.

[37] M. Dunstan and M. Souyri, "The FEIC development for NPAL project: A core image processing chip for smart landers navigation applications," in *MicroElectronics Presentation Days, ESA/ESTEC*, 2004.

[38] C. Urbina-Ortega, G. Furano, G. Magistrati, K. Marinis, and A. Menicucci, "Flash-based FPGAs in Space, design guidelines and trade-off for critical applications," in *Proc. of IEEE Conference on Radiation Effects on Components and Systems (RADECS)*, 2013.

[39] C. Villalpando, R. Werner, J. Carson, G. Khanoyan, R. Stern, and N. Trawny, "A hybrid FPGA/Tilera compute element for autonomous hazard detection and navigation," in *Aerospace Conference, 2013 IEEE*, 2013, pp. 1–9.

[40] EADS Astrium, "Navigation for planetary approach and landing - final report," [Accessed 28-July-2014]. [Online]. Available: http://www.scribd.com/doc/18759944/EADS-Navigation-for-Planetary-Approach-Landing

Stefano Di Carlo received the MS degree in computer engineering and the PhD degree in information technologies from the Politecnico di Torino, Italy, where he has been an assistant professor in the Department of Control and Computer Engineering since 2008. His research interests include DFT, BIST, and reliability. He is a golden core member of the IEEE Computer Society and a senior member of the IEEE.

Giulio Gambardella received the MS degree in electronic engineering from Politecnico di Torino, Italy, where he has been Ph.D student in the Department of Control and Computer Engineering since 2012. His research interests include FPGA dependability and memory testing. He is a student member of IEEE.

Paolo Prinetto Paolo Prinetto received the MS degree in electronic engineering from the Politecnico di Torino, Italy, where he is a full professor of computer engineering in the Department of Control and Computer Engineering. He is also a joint professor at the University of Illinois, Chicago. His research interests include testing, test generation, BIST, and dependability. He is a golden core member of the IEEE Computer Society.

Daniele Rolfo received the MS degree in electronic engineering from Politecnico di Torino, Italy, where he has been Ph.D student in the Department of Control and Computer Engineering since 2012. His research interests include high-performance FPGA-based accelerators, especially for image processing, and dependability issues on hardware accelerators, like CUDA GPUs and FPGA devices. He is a student member of IEEE.

Pascal Trotta received the MS degree in electronic engineering from Politecnico di Torino, Italy, where he has been Ph.D student in the Department of Control and Computer Engineering since 2013. His research interests include image processing hardware acceleration in critical applications, and dependability of reconfigurable devices. He is a student member of IEEE.